Appears in *Third Workshop on Workstation Operating Systems*, Key Biscayne, FL, April, 1992, pp. 126-131.

Disk Reads with DRAM Latency

Garth A. Gibson†, R. Hugo Patterson‡, M. Satyanarayanan†

†School of Computer Science
‡Dept. of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890
garth.gibson@cs.cmu.edu

1. Introduction

The gap between the access time of primary and secondary storage technologies, often called the access gap, has long constrained the performance of computer systems. Unfortunately, this gap is growing, not shrinking. While today's technologies improve the capacity of both of these storage media at a furious pace, they are not as successful at reducing access times. Instead, the speed of computations using only primary memory is increasing five to ten times faster than is the speed with which blocks on magnetic disk can be accessed [Katz89, Patterson88].

To overcome this relative deterioration of secondary storage performance, Input/Output systems are being designed around highly parallel disk arrays [ATC90, Gibson91, Kim86, Patterson88, Salem86], high bandwidth optical networks [Arnould89, Computer90], and cache-specialized file systems [Lazowska86, Nelson88, Ousterhout89, Rosenblum91, Satyanarayanan85]. Disk arrays increase secondary storage throughput by spreading data over many disks so that large accesses are striped for parallel operation and so that many small accesses can operate concurrently. Fast networks preserve throughput improvements derived from disk arrays on file servers as data is delivered to increasingly distant client systems. Secondary storage latency, on the other hand, is primarily addressed by large client and server file caches. Whenever data to be read is found in the cache or data to be written can be delayed in the cache, latency is as short as if secondary storage were constructed with semiconductor memory. Our research focuses on reducing the access latency for data to be read that is not already cached.

This material is based (in part) upon work supported by the National Science Foundation under grant number ECD-8907068. The government has certain rights in this material.

2. Focusing on Read Latency.

While file caching is usually effective at masking the long access latency of secondary storage, read misses can still be critical to performance. Caches, unfortunately, all too frequently experience misses. For example, large caches can have long transient states while the cache is being filled the first time or is being refilled because a new set of tasks have begun. This effect is particularly pronounced in frequently power-cycled portable computers [Kistler91] and heavily shared compute servers. Caches also experience high miss rates when some active files are large enough to flush most or all of the cache [Baker91].

Caches also fail to mask access latency when access patterns inherently peruse data only once. This is the case while reading bulletin boards on a remote workstation or portable computer. While this example may not comprise a large fraction of a system's cycles, its responsiveness may significantly effect user behavior. One technique for overcoming long response times while accessing a file for the first time is to "read ahead" later portions of the file. Where files are being read sequentially, caches employing readahead may be able to overlap latency for all blocks except the first. While this overlapping is largely successful, it does not help accesses to small files and it can hurt non-sequential accesses.

The primary/secondary access gap is not the only source of long read latencies. Network latencies sometimes add significantly to response time. This component of access time is likely to grow in importance as geographically distributed remote file systems and weakly connected, mobile file systems become more common.

3. Using Hints for Informed Prefetching.

We believe that access pattern hints are the key to avoiding long access latencies. Informed with accurate and timely hints about future accesses, a file system can prefetch data to overlap I/O latency with useful computation. We further believe that the extraction and collection of such accurate and timely hints is feasible. The rest of this extended abstract discusses the utility and feasibility of using hints for informed prefetching and our current research plans.

4. Exploiting Hints.

Hints enable informed prefetching to overlap those accesses for which caches are ineffective with useful computation. Hints specifying the set of files needed by a program to work effectively can minimize latency penalties by pre-loading the cache. Where resources are not available to entirely cache such a "working set," hints about access order and access pattern allow "readahead" of non-sequential and first referenced accesses as well as early release of precious memory resources.

When hints identify a sequence of accesses, sequential or otherwise, large enough to flush most or all of the file cache, caching can be bypassed in favor of simple double buffering. This limits the impact of large files on the performance of accesses into other files.

Of course, access pattern hints can also be effectively used to prefetch files that are only accessed once. Such files include the bulletin boards that will be read and the next message of interest within a bulletin board.

5. Obtaining Hints.

Critical to the success of very smart prefetching is the availability of accurate and timely hints. An important part of our research will be to expose these hints. However, we don't think this will be as hard as it might seem. After all, the success of sequential readahead is largely the product of "discovering" that the application is sequentially accessing its files; this is really known a priori because a programmer chosen to do so. It is a simple step to have programmers notify the I/O system, through a hint, of a sequential access pattern.

Using a hint based mechanism to determine what to read ahead, data can be prefetched in a programmer defined, nonsequential pattern. An important beneficiary of this approach will be the large scientific programs that execute alternating row and column access patterns on huge matrix data files [Miller91]. At least one of these access patterns will not be sequential in the file's linear storage, but is easily and obviously specified by a programmer. While this information could be used by the programmer to explicitly and asynchronously prefetch sequential and non-sequential data, fewer programs than we might expect actually do this [Miller91] because it expands virtual memory requirements in systems where memory is a critical resource, because it is not always portable, and because it requires more than a few lines of code. In contrast, system-level prefetching can be driven by very short and simple hints and can easily compensate for reluctance on the part of programmers.

With an interface for programmers to pass hints, there is good reason to extend optimizing compiler techniques to also issue hints. While file names and access patterns could be concealed in complex data structures, we believe that many programs access files simply; file names come as arguments, access calls are made from outer loops, and offsets are computed simply from previous offsets or loop indices. With such straightforward styles, we expect that precise hints can be extracted automatically.

Another source of early, if more speculative, hints is the process' or thread's parent code. For example, a software compilation tool such as "make" has extensive knowledge of the tasks that are going to be executed in the immediate (and sometimes not so immediate) future. To the degree that "make" can internally recognize the functions it spawns, it can give hints about the source, object, and library files that will be needed. Another example is a user shell. When a shell is required to do filename completion or wildcard expansion, it could pass these filenames as hints about what will soon be accessed.

For more accurate hints, a user shell could recognize certain program names on the command line it has constructed for execution. With recognized programs, file arguments can be named in hints before or in parallel with the program's invocation. To recognize and issue hints for specific programs, tools like a user shell will need program specific information. This could be available in a file of hints registered during program installation, or a specially formatted string in the first page of a program's binary. Alternatively, a program-specific "hint" binary whose name is constructed by simple rules could be invoked in parallel.

Finally, our last example method for generating hints exploits the efficient workload tracing tools developed for file system research [Mummert91]. A compactly represented trace of a program's access patterns could be collected and reported so that a programmer can gauge the accuracy of the hints already issued and construct new hints based, possibly literally, on previous traces.

6. Research Goals and Challenges.

Where hints are successfully used, deeper queues of anticipated accesses may develop. This will allow more efficient use of the I/O system by exploiting seek scheduling and grouping accesses into larger transfer units. To achieve these higher levels of efficiency without compromising on the latency of demand reads, we expect to pursue changes in the interfaces to disk controllers. For example, low priority prefetches of very large blocks of data should be preemptable if high priority demand fetches arrive. The impact of successful hints on peripheral interface design and on disk array design is an importance facet of our research.

While deep disk queues improve secondary storage throughput, they also forebode the rapid exhaustion of cache memory resources. Our research will have to take a careful look at the balance of memory demands from recently used data, anticipated data, and virtual memory.

One key to limiting the memory demands of hints will be to track the consumption of prefetched data. This may allow consumed data to be flushed from memory. It should also determine which data to prefetch next. However, with tasks giving hints for other tasks and programs constructed from modules coded (and, therefore, "hinted") separately, it may be hard to determine just which hint is retired by a particular access.

If matching correctly specified hints to demand access patterns is a problem, it will be small compared to identifying incorrectly and imprecisely specified hints. Yet, allowing for imprecise and incorrect hints is necessary. We must find ways to terminate or adjust such hints so that prefetching does not adversely effect performance.

Finally, and perhaps most importantly, we must find a good mechanism to take hints from high levels of abstraction and deliver them to lower levels. Some hints may be best exploited by database code, some best processed by cache management code, some targeted at device drivers, and some useful at many or all lower levels. We need interfaces and abstractions for distributing this type of performance optimization information without disturbing the benefits of layering and modularity in modern applications and operating systems. From a practical point of view, establishing the right interfaces is essential if tools as invasive of other parts of the system as are ours are to survive in today's standards driven environments.

7. Future Plans.

We begin our research simply; we are looking at a subset of applications available in our local Mach/Coda Unix environment that exploit different hint mechanisms and are likely to benefit substantially. The first two target applications have already been mentioned: the compilation tool "make" and large scientific applications. Hints from "make" are about the files accessed by other programs, primarily compilers and loaders (and sed and awk). On the other hand, hints from large scientific applications are likely to be given by the programmer or compiler and apply to a few large files possibly accessed many times. These applications will provide an early test of the opportunity and utility of our ideas.

As our experience grows, we expect to apply our ideas to a much wider domain of systems and applications. We plan to modify compilers to automatically extract access patterns and issue hints from application code. We will also modify a Unix cache manager to expose and exploit issued hints. Eventually, we plan to integrate the extraction and exploitation of hints into an efficient, hierarchical interface for optimizing secondary storage performance.

8. References

[Arnould89] Emmanuel A. Arnould, Francois J. Bitz, Eric C. Cooper, H. T. Kung, Robert D. Sansom, Peter A. Steenkiste, "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers," *Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III),* Boston MA, April 1989, pp. 205-216.

[ATC90] Product Description, RAID+ Series Model RX, Array Technology Corporation, Revision 1.0, Array Technology Corporation, Boulder CO, February 1990.

[Baker91] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, John K. Ousterhout, "Measurements of a Distributed File System," *Operating Systems Review (Proceedings of the 13th SOSP*), Volume 25 (5), October 1991, pp. 198-212.

[Computer90] "Gigabit Network Testbeds," *IEEE Computer*, Volume 23 (9), September 1990, Special Report.

[Gibson91] Garth A. Gibson, Redundant Disk Arrays: Reliable, Parallel Secondary Storage, Ph.D. Dissertation, University of California, Technical Report UCB/CSD 90/613, March 1991. To be published by MIT Press.

[Katz89] Randy H. Katz, G. A. Gibson, D. A. Patterson, "Disk System Architectures for High Performance Computing," *Proceedings of the IEEE*, Volume 77 (12), December 1989, pp. 1842-1858.

[Kim86] Michelle Y. Kim, "Synchronized Disk Interleaving," *IEEE Transactions on Computers*, Volume C-35 (11), November 1986.

[Kistler91] James J. Kistler, M. Satyanarayanan, "Disconnected Operation in the Coda File System," *Operating Systems Review (Proceedings of the 13th SOSP*), Volume 25 (5), October 1991, pp. 213-225.

[Lazowska86] Edward D. Lazowska, John Zahorjan, David R. Cheriton, Willy Zwaenepoel, "File Access Performance of Diskless Workstations," *ACM Transactions on Computer Systems*, Volume 4 (3), August 1986, pp. 238-268.

[Miller91] Ethan L. Miller, "Input/Output Behavior of Supercomputing Applications," University of California, Technical Report UCB/CSD 91/616, January 1991, Master's Thesis.

[Mummert91] Lily B. Mummert, M. Satyanarayanan, "Efficient and Portable File Reference Tracing in a Distributed Workstation Environment," Carnegie Mellon University, manuscript in preparation.

[Nelson88] Mike N. Nelson, Brent B. Welch, John K. Ousterhout, "Caching in the Sprite Network File System," *ACM Transactions on Computer Systems*, Volume 6 (1), February 1988.

[Ousterhout89] John K. Ousterhout, Fred Douglis, "Beating the I/O Bottleneck: A Case for Log-Structured File Systems," *ACM Operating Systems Review*, Volume 23 (1), January 1989, pp 11-28.

[Patterson88] David A. Patterson, Garth A. Gibson, Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD*), Chicago IL, June 1988, pp. 109-116.

[Rosenblum91] Mendel Rosenblum, John K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *Operating Systems Review (Proceedings of the 13th SOSP*), Volume 25 (5), October 1991, pp 1-15.

[Salem86] K. Salem, H. Garcia-Molina, "Disk Striping," *Proceedings of the 2nd IEEE International Conference on Data Engineering*, 1986.

[Satyanarayanan85] M. Satyanarayanan, John H. Howard, et al., "The ITC Distributed File System: Principles and Design," *Operating Systems Review (Proceedings of the 10th SOSP*), Volume 19 (5), December 1985, pp. 35-50.