Aggressive Prefetching: An Idea Whose Time Has Come*

Athanasios E. Papathanasiou and Michael L. Scott
University of Rochester
{papathan,scott}@cs.rochester.edu
http://www.cs.rochester.edu/{~papathan,~scott}

Abstract

I/O prefetching serves to hide the latency of slow peripheral devices. Traditional OS-level prefetching strategies have tended to be conservative, fetching only those data that are very likely to be needed according to some simple heuristic, and only just in time for them to arrive before the first access. More aggressive policies, which might speculate more about which data to fetch, or fetch them earlier in time, have typically not been considered a prudent use of computational, memory, or bandwidth resources. We argue, however, that technological trends and emerging system design goals have dramatically reduced the potential costs and dramatically increased the potential benefits of highly aggressive prefetching policies. We propose that memory management be redesigned to embrace such policies.

1 Introduction

Prefetching, also known as prepaging or read-ahead, has been standard practice in operating systems for more than thirty years. It complements traditional caching policies, such as LRU, by hiding or reducing the latency of access to non-cached data. Its goal is to predict future data accesses and make data available in memory before they are requested.

A common debate about prefetching concerns how aggressive it should be. Prefetching aggressiveness may vary in terms of timing and data coverage. The timing aspect determines how early prefetching of a given block should occur. The data coverage aspect determines how speculative prefetching should be regarding which blocks are likely to be accessed. Conservative prefetching attempts to fetch data incrementally, just in time to be accessed, and only when confidence is high [3]. Aggressive prefetching is distinguished in two ways. First, it prefetches deeper in a reference stream, earlier than would be necessary simply to hide I/O latencies. Second, it speculates more about future accesses in an attempt to increase data coverage, possibly at the cost of prefetching unnecessary data.

The literature on prefetching is very rich (far too rich to include appropriate citations here). Researchers have suggested and experimented with history-based predictors, application disclosed hints, application-controlled prefetching, speculative execution, and data compression in order to improve prefetching accuracy and coverage for both inter- and intra-file accesses. Various methods have also been proposed to control the amount of memory dedicated to prefetching and the possible eviction of cached pages in favor of prefetching.

Published studies have shown that aggressive prefetching has the potential to improve I/O performance for a variety of workloads and computing environments, either by eliminating demand misses on pages that a conservative system would not prefetch, or by avoiding long delays when device response times are irregular. Most modern operating systems, however, still rely on variants of the standard, conservative sequential read-ahead policy. Linux, for example, despite its reputation for quick adoption of promising research ideas, prefetches only when sequential access is detected, and (by default) to a maximum of only 128 KB.

Conservative algorithms have historically been reasonable: aggressive prefetching can have a negative impact on performance. We begin by reviewing this downside in Section 2. In Section 3, however, we argue that the conventional wisdom no longer holds. Specifically, the risks posed by aggressive prefetching are substantially reduced on resource-rich modern systems. Moreover, new system design goals, such as power efficiency and disconnected or weakly-connected operation, demand the implementation of very aggressive policies that predict and prefetch data far ahead of their expected use.

*This work was supported in part by NSF grants EIA-0080124, CCR-0204344, and CNS-0411127; and by Sun Microsystems Laboratories.

Finally, in Section 4 we present new research challenges for prefetching algorithms and discuss the implications of those algorithms for OS design and implementation.

2 Traditional concerns

Under certain scenarios aggressive prefetching may have a severe negative impact on performance.

Buffer cache pollution. Prefetching deeper in a reference stream risks polluting the buffer cache with unnecessary data and ejecting useful data. This risk is particularly worrisome when memory is scarce, and when predictable (e.g. sequential) accesses to predictable (e.g. server disk) devices make it easy to compute a minimum necessary "lead time", and offer little benefit from working farther ahead.

Increased physical memory pressure. Aggressive prefetching may increase physical memory pressure, prolonging the execution of the page replacement daemon. In the worst case, correctly prefetched pages may be evicted before they have a chance to be accessed. The system may even thrash.

Inefficient use of I/O bandwidth. Aggressive prefetching requires speculation about the future reference stream, and may result in reading a large amount of unnecessary data. Performance may suffer if bandwidth is a bottleneck.

Increased device congestion. Aggressive prefetching leads to an increased number of *asynchronous* requests in I/O device queues. *Synchronous* requests, which have an immediate impact on performance, may be penalized by waiting for prefetches to complete.

Techniques exist to minimize the impact of these problems. More accurate prediction algorithms can minimize cache pollution and wasted I/O bandwidth. The ability to cancel pending prefetch operations may reduce the risk of thrashing if memory becomes too tight. Replacement algorithms that balance the cost of evicting an already cached page against the benefit of prefetching a speculated page can partially avoid the ejection of useful cached data (this assumes an on-line mechanism to accurately evaluate the effectiveness of caching and prefetching). Prefetched pages can be distinguished from other pages in the page cache so that, for example, they can use a different replacement policy (LRU is not suitable). Finally, priority-based disk queues that schedule requests based on some notion of criticality can reduce the impact of I/O congestion. All of these solutions, unfortunately, introduce significant implementation complexity, discouraging their adoption by general-purpose operating systems. In addition, most of the problems above are most severe on resource-limited systems. A general-purpose OS, which needs to run on a variety of machines, may forgo potential benefits at the high end in order to avoid more serious problems at the low end.

3 Why it makes sense now

Two groups of trends suggest a reevaluation of the conventional wisdom on aggressive prefetching. First, technological and market forces have led to dramatic improvements in processing power, storage capacity, and to a lesser extent I/O bandwidth, with only modest reductions in I/O latency. These trends serve to increase the need for aggressive prefetching while simultaneously decreasing its risks. Second, emerging design goals and usage patterns are increasing the need for I/O while making its timing less predictable; this, too, increases the value of aggressive prefetching.

3.1 Technological and market trends

Magnetic disk performance. Though disk latency has been improving at only about 10% per year, increases in rotational speed and recording density have allowed disk bandwidths to improve at about 40% per year [6]. Higher bandwidth disks allow—in fact require—the system to read more data on each disk context switch, in order to balance the time that the disk is actively reading or writing against the time spent seeking. In recognition of this fact, modern disks have large controller caches devoted to speculatively reading whole tracks. In the absence of memory constraints, aggressive prefetching serves to exploit large on-disk caches, improving utilization.

Large memory size at low cost. Memory production is increasing at a rate of 70% annually, while prices have been dropping by 32% per year [5], reaching a cost today of about 12 cents per megabyte. Laptop computers with 512 MB of memory are now common, while desktop and high-end systems may boast several Gigabytes. While application needs have also grown, it seems fair to say on the whole that today's machines have significantly more memory "slack" than their predecessors did, providing the opportunity to prefetch aggressively with low risk of pollution or memory pressure.

Figure 1 presents laptop memory availability and company recommended memory sizes for commercial operating systems and applications since 1995. Memory availability is shown using three possible configurations: the *maximum* and *minimum* available, and a "*low cost*" option obtained by filling every memory slot with the density of RAM that maximized MB/dollar in the technology of the day. A comparison between the recommended memory size for our most memory-intensive application, Photoshop, and the low-cost memory configuration shows that the available memory "slack" in a low-cost laptop has grown from less than 1 MB in 1995 to

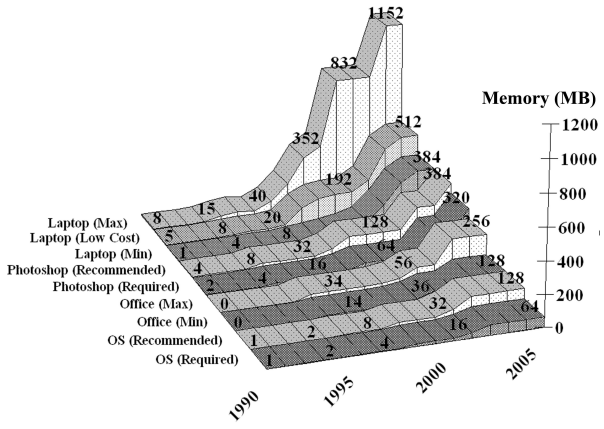

Figure 1: Memory "slack" trends. The figure presents how laptop memory availability and memory requirements of commercial applications and operating systems have changed during the last 15 years. Memory availability data are based on Macintosh laptop systems [1] and are represented using three possible configurations: the *maximum* and *minimum* available, and a "*low cost*" option obtained by filling every memory slot with the density of RAM that maximized MB/dollar in the technology of the day. Memory requirement data are based on company recommendations for Microsoft Windows operating systems and two applications, Adobe Photoshop and Microsoft Office. For Microsoft Windows and Adobe Photoshop company *recommended* and *required* memory sizes are provided. For Microsoft Office, *Min* represents the memory required to run a single office application and *Max* represents the memory required to run six office applications concurrently.

more than 100 MB today. The available "slack" is significantly higher for high-end laptops and desktops. As memory sizes and disk bandwidths continue to increase, and as multimedia applications continue to proliferate (more on this below), the performance benefit of aggressive prefetching will surpass that of caching policies.

I/O performance variability. In the past, prefetching algorithms have been developed under the assumption that storage devices are able to deliver data at relatively constant latencies and bandwidths. This assumption no longer holds. First, users access data through multiple devices with different performance characteristics. Second, the performance of a given device can vary greatly. Increases in areal densities of magnetic disks have led to bandwidth differences of 60% or more [7] between inner and outer tracks, and this gap is expected to grow. Li *et al.* [10] demonstrate a 15% to 47% throughput improvement for server-class applications through a competitive prefetching algorithm that takes into account the performance variability of magnetic disks. Similarly, wireless network channels are prone to noise and contention, resulting in dramatic variation in bandwidth over time. Finally, power-saving modes in disks and other devices can lead, often unpredictably, to very large increases in latency. To maintain good performance and make efficient use of the available bandwidth, prefetching has to be sensitive to a device's performance variation. Aggressive prefetching serves to exploit periods of higher bandwidth, hide periods of higher latency, and generally smooth out fluctuations in performance.

The processor–I/O gap. Processor speeds have been doubling every 18 to 24 months, increasing the performance gap between processors and I/O systems. Processing power is in fact so far ahead of disk latencies that prefetching has to work multiple blocks ahead to keep the processor supplied with data. Moreover, many applications exhibit phases that alternate between compute-intensive and I/O-intensive behavior. To maximize processor utilization, prefetch operations must start early enough to hide the latency of the *last* access of an I/O-intensive phase. Prefetching just in time to minimize the impact of the *next* access is not enough. Early prefetching, of course, implies reduced knowledge about future accesses, and requires both more sophisticated and more speculative predictors, to maximize data coverage. Fortunately, modern machines have sufficient spare cycles to support more computationally demanding predictors than anyone has yet proposed. In recognition of the increased portion of time spent on I/O during system startup, Microsoft Windows XP employs history-based informed prefetching to reduce operating system boot time and application launch time [12]. Similar approaches are being considered by Linux developers [8].

3.2 Design goals and usage patterns

Larger data sizes. Worldwide production of magnetic content increased at an annual rate of 22% from 1999 to 2000 [11]. This increase has been facilitated by annual increases in disk capacity of 130% [19]. Multimedia content (sound, photographs, video) contributes significantly to and will probably increase the growth rate of on-line data. As previous studies have shown [2, 14], larger data sets diminish the impact of larger memories on cache hit rates, increasing the importance of prefetching. Media applications also tend to touch each datum only once, limiting the potential of caching.

Multitasking. The dominance and maturity of multitasking computer systems allow end users to work concurrently on multiple tasks. On a typical research desktop, it is easy to imagine listening to a favorite MP3 track while browsing the web, downloading a movie, rebuilding a system in the background, and keeping half an eye on several different instant messaging windows. The

constant switching among user applications, several of which may be accessing large amounts of data, reduces the efficiency of LRU-style caching; aggressive prefetching allows each application to perform larger, less frequent I/O transfers, exploiting the disk performance advances described above.

Energy and power efficiency. Energy and power have become major issues for both personal computers and servers. Components of modern systems—and I/O devices in particular—have multiple power modes. Bursty, speculative prefetching can lead to dramatic energy savings by increasing the time spent in non-operational low-power modes [15]. As shown in our previous work [16], we can under reasonable assumptions[1] save as much as 72% of total disk energy even if only 20% of what we prefetch actually turns out to be useful. At the same time, *operational* low-power modes, as in recent proposals for multi-speed disks [4], suggest the need for *smooth* access patterns that can tolerate lower bandwidth. Prefetching algorithms for the next generation of disks may need to switch dynamically between smooth low-bandwidth operation and bursty high-bandwidth operation, depending on the offered workload.

Reliability and availability of data. Mobile systems must increasingly accommodate disconnected or weakly-connected operation [9], and provide efficient support for several portable storage devices [13]. Reliability and availability have traditionally been topics of file system research. We believe, however, that memory management and specifically prefetching must play a larger role. Mobile users may need to depend on multiple legacy file systems, not all of which may handle disconnection well. But while computers may have multiple file systems, they have a *single* memory management system. The file system functionality that provides access under disconnected or weakly-connected operation is based on aggressive, very speculative prefetching (with caching on local disk). This prefetching can be moved from the low-level file system to the memory management and virtual file system layers, where it can be used in conjunction with arbitrary underlying file systems. Aggressive prefetching that monitors access patterns through the virtual file system and is implemented at the memory management level might prefetch and back up data to both RAM and local peripheral devices.

[1] We assume 50 MB of memory dedicated to prefetching and an application data consumption rate of 240 KB/s (equivalent to MPEG playback). Energy savings are significant for several combinations of memory sizes used for prefetching, data rates and prefetching accuracy ratings.

4 Research challenges

The trends described in Section 3 raise new design challenges for aggressive prefetching.

Device-centric prefetching. Traditionally, prefetching has been application-centric. Previous work [17] suggests a cost-benefit model based on a constant disk latency in order to control prefetching. Such an assumption does not hold in modern systems. To accommodate power efficiency, reliability, and availability of data under the varying performance characteristics of storage devices, prefetching has to reflect both the application *and* the device. Performance, power, availability, and reliability characteristics of devices must be exposed to prefetching algorithms [10, 13, 16].

Characterization of I/O demands. Revealing device characteristics is not enough. To make informed decisions the prefetching and memory management system will also require high level information on access patterns and other application characteristics. An understanding of application reliability requirements, bandwidth demands, and latency resilience can improve prefetching decisions.

Coordination. Non-operational low-power modes depend on long idle periods in order to save energy. Uncoordinated I/O activity generated by multitasking workloads reduces periods of inactivity and frustrates the goal of power efficiency. Aggressive, coordinated prefetching can be used in order to coordinate I/O requests across multiple concurrently running applications and several storage devices.

Speculative predictors that provide increased data coverage. Emerging design goals, described in Section 3, make the case to prefetch significantly deeper than traditional just-in-time policies would suggest. In addition to short-term future data accesses, prefetching must predict long-term user intention and tasks in order to minimize the potentially significant energy costs of misses (e.g. for disk spin-up) and to avoid the possibility of application failure during weakly-connected operation.

Prefetching and caching metrics. Traditionally, cache miss ratios have been used in order to evaluate the efficiency of prefetching and caching algorithms. The utility of this metric, however, depends on the assumption that all cache misses are equivalent [18]. Power efficiency, availability, and varying performance characteristics lead to different costs for each miss. For example, a miss on a spun-down disk can be significantly more expensive in terms of both power and performance than a miss on remote data accessed through the network. We need new methods to evaluate the effectiveness of proposed prefetching algorithms.

In addition, several traditional prefetching problems may require new or improved solutions as prefetching becomes more aggressive. Examples include:

- Separate handling of prefetched and cached, accessed pages.
- Algorithms that dynamically control the amount of physical memory dedicated to prefetching.
- Monitoring systems that evaluate the efficiency of predictors and prefetching algorithms using multiple metrics (describing performance, power efficiency, and availability) and take corrective actions if necessary.
- Priority-based disk queues that minimize the possible negative impact of I/O queue congestion.
- Mechanisms to cancel in-progress prefetch operations in the event of mispredictions or sudden increases in memory pressure.
- Data compression or other techniques to increase data coverage.

To first approximation, the memory management system of today assumes responsibility for caching secondary storage without regard to the nature of either the applications above it or the devices beneath it. We believe this has to change. The "storage management system" of the future will track and predict the behavior of applications, and prioritize and coordinate their likely I/O needs. At the same time, it will model the latency, bandwidth, and reliability of devices over time, moving data not only between memory and I/O devices, but among those devices as well, to meet user-specified needs for energy efficiency, availability, reliability, and interactive responsiveness.

References

[1] Apple Computer, Inc. Apple Specifications. Available: http://www.info.apple.com/support/applespec.html.

[2] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a Distributed File System. In *Proc. of the 13th ACM Symp. on Operating Systems Principles*, 1991.

[3] P. Cao, E. W. Felten, and K. Li. A Study of Integrated Prefetching and Caching Strategies. In *Proc. of the 1995 ACM Joint Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS'95/PERFORMANCE'95)*, 1995.

[4] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *Proc. of the 30th Int. Symp. on Computer Architecture (ISCA'03)*, June 2003.

[5] J. Handy. Will the Memory Market EVER Recover?, Sept. 1999. http://www.reed-electronics.com/electronicnews/article/CA47864.html.

[6] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman, 3^{rd} edition, 2003.

[7] Hitachi Global Storage Technologies. Hard Disk Drive Specification; Hitachi Travelstar 5K80 2.5 inch ATA/IDE hard disk drive, Nov. 2003.

[8] Kerneltrap.org. Linux: Boot Time Speedups Through Precaching, Jan. 2004. http://kerneltrap.org/node/2157.

[9] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Trans. on Computer Systems*, 10(1), Feb. 1992.

[10] C. Li, A. E. Papathanasiou, and K. Shen. Competitive Prefetching for Data-Intensive Online Servers. In *Proc. of 1st Workshop on Operating Sys. and Arch. Support for the on-demand IT InfraStructure*, Oct. 2004.

[11] P. Lyman and H. R. Varian. How Much Information, 2003. School of Information Management and Systems, University of California at Berkeley. Available: http://www.sims.berkeley.edu/how-much-info-2003/.

[12] Microsoft Corporation. Kernel Enhancements for Windows XP, Jan. 2003. http://www.microsoft.com/whdc/driver/kernel/XP_kernel.mspx.

[13] E. B. Nightingale and J. Flinn. Energy-Efficiency and Storage Flexibility in the Blue File System. In *Proc. of the 6th USENIX Symp. on Operating Systems Design and Implementation*, Dec. 2004.

[14] J. K. Ousterhout, H. D. Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson. A Trace-driven Analysis of the UNIX 4.2 BSD File System. In *Proc. of the 10th ACM Symp. on Operating Systems Principles*, Dec. 1985.

[15] A. E. Papathanasiou and M. L. Scott. Energy Efficiency Through Burstiness. In *Proc. of the 5th IEEE Workshop on Mobile Computing Systems and Applications*, Oct. 2003.

[16] A. E. Papathanasiou and M. L. Scott. Energy Efficient Prefetching and Caching. In *Proc. of the USENIX 2004 Annual Technical Conf.*, June 2004.

[17] R. H. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed Prefetching and Caching. In *Proc. of the 15th ACM Symp. on Operating Systems Principles*, Dec. 1995.

[18] M. Satyanarayanan. Mobile Computing: Where's the Tofu? *ACM SIGMOBILE Mobile Comp. and Comm. Review*, 1(1), Apr. 1997.

[19] G. Zeytinci. Evolution of the Major Computer Storage Devices, 2001. Available: http://www.computinghistorymuseum.org/teaching/papers/research/StorageDevices-Zeytinci.pdf.