ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 12, December 2014

Statistical Analysis of Disk Drives Performance with Application Directed Prefetching Approach

Javed hussain[1], Ilyas Khan[2]

Research Scholar, Mewar University, Chittorgarh, Rajasthan, India[1].

Vidyapeeth Institute of Science and Technology, Bhopal, Madhya Pradesh, India[2].

ABSTRACT: Modern magnetic disks are, as is well known, dramatically slower at random reads than sequential reads. Technological progress has exacerbated the problem; disk throughput has increased by a factor of 60 to 85 over the past twenty-five years, but seek times have decreased by a factor of only 15. Disks are less and less like random access devices in terms of performance. Although flash memory reduces the cost differential of random accesses, disks continue to offer vast amounts of inexpensive storage. For the foreseeable future, it will remain important to optimize the performance of applications that access disk like devices that is, devices with much faster sequential than random access. The best solution to this performance problem is to avoid critical path disk access altogether and reading unneeded data as sequential access and discarding it later.
.

KEYWORDS: Infill, Libprefetch, Seek, Readahead, StridedAccess ,Sqlite .

I. INTRODUCTION

In application-directed prefetching systems, the application informs the storage system of its intended upcoming reads. (Databases, scientific workloads, and others are easily able to calculate future accesses. Previous work on application-directed caching and prefetching demonstrated relatively low speedups for single-process, single-disk workloads (average speedup 26%, maximum 49%). However, this work aimed to overlap CPU time and I/O fetch time without greatly increasing memory use, and thus prefetched relatively little data from disk (16 blocks) just before a process needed it. Our system, aims solely to minimize I/O fetch time, a better choice given today's widened gap between processor and disk performance. The prefetching system is aggressive, fetching as much data as fits in available memory. It is also relatively simple, fitting in well with existing operating system techniques; most code is in a user-space library. Small, but critical, changes in kernel behaviour help ensure that prefetched data is kept until it is used. A contention controller detects changes in available memory and compensates by resizing the prefetching window, avoiding performance collapse when prefetching applications compete for memory and increasing performance when more memory is available. Our measurements show substantial speedups on test workloads, such as a 20x speedup on a SQLite table scan of a data set that is twice the size of memory. Running concurrent instances of applications with libprefetch shows similar factors of improvement.

II. RELATED WORK

Fuelled by the long-growing performance gulf between disk and CPU speeds, considerable research effort has been invested in improving disk read performance by caching and prefetching. Prefetching work in particular has been based on predicted, application-directed, and inferred disk access patterns

Disk Modeling Ruemmler and Wilkes is the classic paper on disk performance modeling. Our seek time observations complement those of Schlosser et al; like them, we use our observations to construct more effective ways to use a disk.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

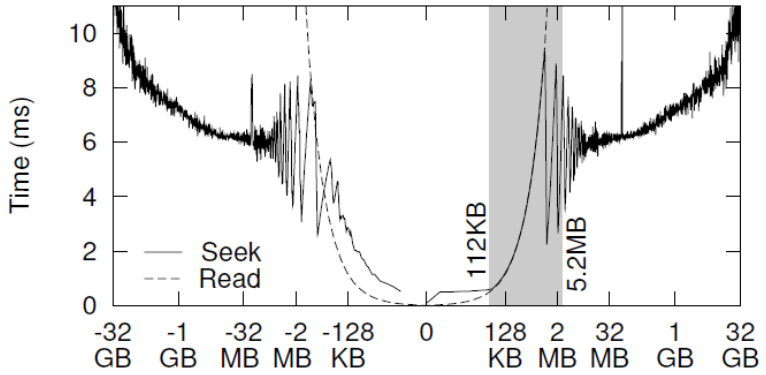
International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 12, December 2014

Predicting Accesses Operating systems have long employed predictive read-ahead algorithms to speed up sequential file access. This improves performance for many workloads, but can retard performance if future accesses are mis-predicted. As a result, read-ahead algorithms usually don't try to improve less predictable access patterns, such as sequential reads of many small files or non-sequential reads of large files.

Application-Directed Accesses Cao et al. and Patterson et al present systems like libprefetch where applications convey their access patterns to the file sys-tem to increase disk read performance.



Inferred Accesses Rather than requiring the application to explicitly supply a list of future reads, a prefetching system can automatically generate the list either from application source code, using static analysis [1], or from the running application, using speculative execution [3].

III. THE IMPACT OF MODERN DISK CHARACTERISTICS ON PREFETCHING

Disk prefetching algorithms aim to improve the performance of future disk reads by reading data before it is needed. Since our prefetching algorithm will use precise application information about future accesses, we need not worry about detecting access patterns or trying to predict what the application will use next. Instead, the chief goal is to determine the fastest method to retrieve the requested data from disk.

This section uses disk benchmarks to systematically build up a prefetching algorithm that takes advantage of the strengths, and as much as possible avoids the weak-nesses, of modern I/O subsystems. The prefetching algorithm must read at least the blocks needed by the application, so there are only a few degrees of freedom available. The prefetcher can reorder disk requests within the window of memory available for buffering, it can combine disk requests, and it can read non-required data if that would help. Different disk layout or block allocation algorithms could also lead to better performance, but these file system design techniques are orthogonal to the issues we consider.

Seek Performance

Conventional disk scheduling algorithms do not know what additional requests will arrive in the future, leading to a relatively small buffer of requests that can be reordered. In contrast, a prefetching algorithm that does know future accesses can use a reorder buffer as large as available memory. A larger buffer can substantially reduce average seek distance. In this section, we measure the actual cost of various seek distances on modern disks, aiming to determine where seek distance matters and by how much. We measured the average time to seek various distances, both forward and backward. Because the seek operation is below the disk interface abstraction, it is only possible to measure a seek in conjunction with a read or write operation. Therefore, the benchmarks start by reading the first block of the disk to establish the disk head location (or the last block, if seeking backward), then read several blocks from the disk, each separated by the seek distance being tested. With this test methodology, a seek distance of zero means that we read

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 12, December 2014

sequential disk locations with multiple requests, and a seek distance of 1 block re-reads the same block repeatedly. All the tests in this section use Direct I/O to skip the buffer cache, ensuring that buffer cache hits do not optimize away the effects we are trying to measure Seek time increases by roughly a factor of five from around 112KB to roughly 1 to 4MB, shown as in the graphs. In contrast, seek times for distances above 2.7MB increase slowly, by about a factor of one.Not considering the disk geometry effects visible as oscillations, a disk scheduling algorithm should minimize seek distance; though not all seek reductions are equal, reducing medium seeks far below 2MB will have more impact than reducing very large seeks to 2MB or more. Figure 1 shows the unexpected result that for distances up to 2.7MB, it may be cheaper to read that amount of data than to seek. This suggests that adjacent requests with small gaps might be serviced faster by requesting the entire range of data and discarding the uninteresting data.

IV. RESULT

Figure 2 suggested that reading and discarding small gaps between requests might be faster than seeking over those gaps.We modified the previous benchmark to add infill, varying the maximum infill allowed. Figure shows that in fill

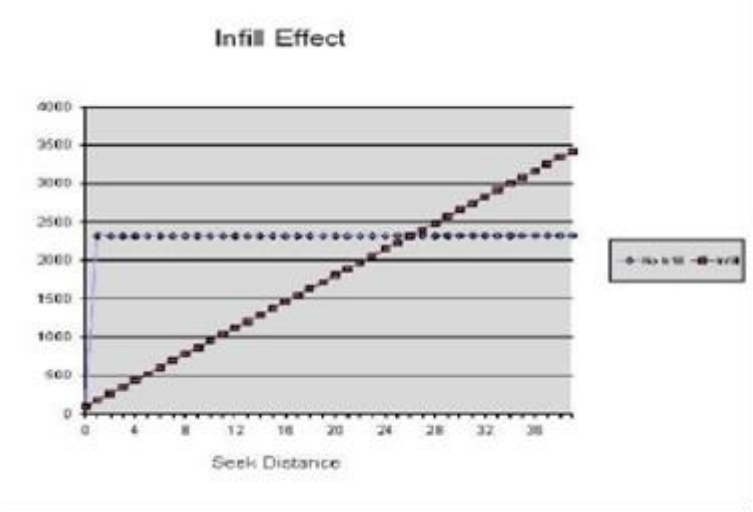


Figure 2.Infill effect

of up to 32KB reduces runtime on Disk 2. This corresponds to the region of Figure 1 where seeks take longer than similarly-sized maximum-throughput reads. Infill amounts between 32KB and 2MB have no additional effect, corresponding to the region of the seek graph where seek time is equal to read time for an equivalent amount of data. When infill is allowed to exceed 2MB, runtime increases. Itconfirming that reads of this size are more expensive than seeking. For the 16MB reorder buffer dataset, infill has very little effect: as Figure 2 shows, the average seek distance for this test is 128KB, above the threshold where we expect infill to help. On Disk 1, however, infill was performance-neutral. Apparently its firmware makes infill largely redundant.

V. CONCLUSION

An analysis of the performance characteristics of modern disks led us to a new approach to prefetching. Our prefetching algorithm minimizes the number of expensive seeks and leads to a substantial performance boost for non-sequential workloads. Libprefetch, a relatively simple library that implements this technique, can speed up real-world instances of non-sequential disk access, including image processing and database table scans, by as much as 4.9x and 20x, respectively, for workloads that do not fit in main memory. Furthermore, a simple contention controller enables this new prefetching algorithm to peacefully coexist with multiple instances of it as well as other applications. An analysis of the performance characteristics of modern disks led us to a new approach of infilling. Our infill algorithm minimizes seek time of the substantial expensive seeks and leads to performance boost for non-sequential workloads.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 12, December 2014

Although it need average of 3 MB of dedicated memory for operation but provides 1.049 xs-1.084 x improvements in seek time.

REFERENCES

1. Chris Ruemmler and John Wilkes, "An introduction to disk drive *modelling.",IEEE Computer 27(3):17-28*, 1994.
2. http://fox.wikis.com/wc.dll? Wiki~ Modern Hard Disk Specifications.
3. Thomas M.,Kroeger and Darrell D. E. Long, "The case for efficient file access pattern modeling." In Proc. 7th Workshop on Hot Topics in Operating Systems (HotOS-VII), pages 14-19, Rio Rico, AZ.
4. Steven W., Schlosser, Jiri Schindler, Stratos Papadomanolakis, Minglong Shao, Anastassia, Ailamaki, Christos, Faloutsos, and Gregory, R. Ganger, "On multidimensional data and modern disks. ", In Proc. 4th USENIX Conference on File and Storage Technologies (FAST '05), pages 225-238, San Francisco, CA, December.
5. James Griffioenand Randy Appleton, "Reducing file system latency using a predictive approach", In Proc. USENIX Summer 1994Technical Conference, pages 197-207, Boston, MA.
6. Dan Revel, Dylan McNamee, David Steere, and Jonathan Walpole "Adaptive Prefetching for Device-Independent File I/O", http://proceedings.spiedigitallibrary.org/ on 12/19/2013 Terms of Use: http://spiedl.org/terms.
7. Seung Woo Son, SaiPrashanthMuralidhara, OzcanOzturk, MahmutKandemir, Ibrahim Kolcu,Mustafa Karakoy, "Profiler and Compiler Assisted Adaptive I/O Prefetching for Shared Storage Caches" , PACT'08,October 25–29, 2008, Toronto, Ontario, Canada.