Scheduling for Modern Disk Drives and Non-Random Workloads

Bruce L. Worthington, Gregory R. Ganger, Yale N. Patt
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor 48109-2122

March 1, 1994

Abstract

Disk subsystem performance can be dramatically improved by dynamically ordering, or *scheduling*, pending requests. Modern disk drives have several features, such as complex logical-to-physical mappings and large prefetching caches, that can influence scheduling effectiveness. Via strongly validated simulation, we examine the impact of these features on various scheduling algorithms. Using both synthetic workloads and traces captured from six different user environments, we arrive at three main conclusions: (1) Incorporating complex mapping information into the scheduler provides only a marginal (less than 2%) decrease in response times for seek-reducing algorithms. (2) Algorithms which effectively utilize a prefetching disk cache provide significant performance improvements for workloads with read sequentiality. The cyclical scan algorithm (C-LOOK), which always schedules requests in ascending logical order, achieves the highest performance among seek-reducing algorithms for such workloads (five of the six examined). (3) Algorithms that reduce overall positioning delays produce the highest performance provided that they recognize and exploit a prefetching cache.

Acknowledgements

We would like to thank Jim Browning, John Fordemwalt, David Jaffe, Rama Karedla, Richie Lary, Mike Leonard, Joe Pasquale, Rusty Ransford, Chris Ruemmler, and John Wilkes for their insights on various I/O issues. We thank Chris Ruemmler and John Wilkes (of HP Labs) and Richie Lary and Rama Karedla (of DEC) for the traces. We thank Jim Pike of NCR Columbia for providing the initial funding for our research into the I/O subsystem, and Roy Clark of NCR Columbia for funding much of the research upon which this report is based. We also wish to acknowledge the other members of our research group at the University of Michigan. Finally, our research group is very fortunate to have the financial and technical support of several industrial partners, including NCR Corporation, Intel, Motorola, Scientific and Engineering Software, HaL, Digital Equipment Corporation, Micro Technology Inc., and Hewlett-Packard.

TABLE OF CONTENTS

1 Introduction **1**

2 Modern Disk Drives **2**

2.1 Host Interface 3

2.2 Data Layout 3

2.2.1 Zoned Recording 3

2.2.2 Track and Cylinder Skew 3

2.2.3 Defect Management 3

2.3 On-Board Cache 4

2.4 Overhead and Positioning Delays 5

3 Disk Scheduling Algorithms **5**

3.1 Seek Delay Reduction 5

3.2 Positioning Delay Reduction 6

3.3 Batch and Aging Algorithms 6

4 Methodology and Validation **6**

4.1 Disk simulator 6

4.2 Workloads 8

4.3 Disks 8

4.4 Metrics 9

5 Synthetic Workloads **9**

5.1 Scheduling by Logical Block Number 10

5.2 Scheduling with a Known Mapping 10

5.3 Scheduling with Full Knowledge 12

6 Traced Workloads **15**

6.1 Scheduling by Logical Block Number 15

6.2 Scheduling with a Known Mapping 23

6.3 Scheduling with Full Knowledge 30

7 Conclusions and Future Work **38**

A Simulation Parameters **40**

LIST OF FIGURES

1 Disk Drive Internals . . . 2

2 Sample Validation Workload Response Time Distributions . . . 8

3 LBN-Based Algorithm Performance using a Synthetic (Random) Workload . . . 11

4 Full-Map and LBN-Based Algorithm Performance using a Synthetic (Random) Workload . . . 13

5 Full-Knowledge Algorithm Performance using a Synthetic (Random) Workload . . . 14

6 LBN-Based Algorithm Performance using the *Cello* Trace . . . 16

7 LBN-Based Algorithm Performance using the *Snake* Trace . . . 17

8 LBN-Based Algorithm Performance using the *Air-Rsv* Trace . . . 18

9 LBN-Based Algorithm Performance using the *Sci-TS* Trace . . . 19

10 LBN-Based Algorithm Performance using the *Order* Trace . . . 20

11 LBN-Based Algorithm Performance using the *Report* Trace . . . 21

12 Full-Map and LBN-Based Algorithm Performance using the *Cello* Trace . . . 24

13 Full-Map and LBN-Based Algorithm Performance using the *Snake* Trace . . . 25

14 Full-Map and LBN-Based Algorithm Performance using the *Air-Rsv* Trace . . . 26

15 Full-Map and LBN-Based Algorithm Performance using the *Sci-TS* Trace . . . 27

16 Full-Map and LBN-Based Algorithm Performance using the *Order* Trace . . . 28

17 Full-Map and LBN-Based Algorithm Performance using the *Report* Trace . . . 29

18 Full-Knowledge Algorithm Performance using the *Cello* Trace . . . 31

19 Full-Knowledge Algorithm Performance using the *Snake* Trace . . . 32

20 Full-Knowledge Algorithm Performance using the *Air-Rsv* Trace . . . 33

21 Full-Knowledge Algorithm Performance using the *Sci-TS* Trace . . . 34

22 Full-Knowledge Algorithm Performance using the *Order* Trace . . . 35

23 Full-Knowledge Algorithm Performance using the *Report* Trace . . . 36

A.1 HP C2247 Disk Drive Seek Curve . . . 42

A.2 Validation Workload Response Time Distributions: 95% Random Reads . . . 44

A.3 Validation Workload Response Time Distributions: 95% Random Writes . . . 44

A.4 Validation Workload Response Time Distributions: 95% Sequential Reads . . . 45

A.5 Validation Workload Response Time Distributions: 95% Sequential Writes . . . 45

LIST OF TABLES

1 HP C2247 Disk Drive Basic Parameters 7

2 Basic Characteristics of the Traces 15

3 Sample Average Response Times for LBN-Based Algorithms 22

4 On-board Disk Cache Hit Rates for LBN-Based Algorithms 23

5 On-board Disk Cache Hit Rates for SPTF-based Algorithms 37

A.1 Parameters for modeling the HP C2247 on the NCR 3550 system. 41

A.2 Zone specifications for the HP C2247. 41

A.3 Seek parameters for modeling the HP C2247. 42

A.4 Overhead delays for modeling the HP C2247 on the NCR 3550 system. 43

A.5 Additional write overhead delays for modeling the HP C2247. 43

1 Introduction

Disk subsystems must be carefully managed to compensate for the growing performance disparity between processing and storage components. Disk workloads are often characterized by intense bursts of activity, creating long queues of pending requests [McNu86, Ruem93]. When such queues develop, the disk scheduler is responsible for dynamically ordering the pending requests to decrease service times. A significant portion of request service time may consist of mechanical positioning delays, which are highly dependent on the relative positions of the various request blocks and the current disk arm position. By taking into account the various delays associated with disk accesses, a scheduler can produce a request stream that minimizes the total positioning overhead but provides reasonable response times for individual requests.

Over the past 25 years, a variety of scheduling algorithms have been proposed and implemented in both academia and industry. Their relative merit is dependent on the exact workload of disk requests, the scheduler's computational resources, and the available knowledge of disk drive configuration and current status. Most previous studies of scheduling algorithms use simplistic disk models, lacking several important features present in modern disk drives. On-board logic has expanded, enabling a high-level interface to hide the the mapping of logical data blocks to physical media, and consequently make this information unavailable to any external scheduling entity. Small speed-matching buffers have been replaced with large prefetching caches. In this report, we investigate how these features affect the performance of various disk scheduling algorithms.

Previously published work has been limited to synthetic workloads. Most of these studies assume that request starting locations are uniformly distributed over the entire disk. While much can be learned using such *random* workloads, the validity of the results remains in question until verified using more "realistic" workloads. We use random workloads to allow for comparison with previous work, but most of our experiments use extensive traces taken from various real-world systems. The traces and synthetic workloads are used to drive a very detailed, well-validated disk simulator.

For random workloads, our results correlate well with previous studies. Algorithms that attempt to reduce seek delays improve performance for all but the lightest workloads. Algorithms that target both seek and rotational latency reduction provide even higher levels of performance. In addition, we show that little benefit is obtained by incorporating full logical-to-physical mappings into seek-reducing scheduling algorithms. Scheduling based purely on logical block numbers falls less than 2% behind full-map scheduling for all seek-reducing algorithms examined.

With the real-world traces, we show that full logical-to-physical mapping information does not provide a significant performance advantage for seek-reducing algorithms (matching the results for random workloads). However, the relative merit of the various algorithms is quite different from that observed for random workloads. The significant sequentiality and locality that characterize most of the traces cause the on-board, prefetching disk cache to play a major role. Scheduling algorithms that utilize the cache effectively achieve higher performance. The cyclical scan algorithm (C-LOOK), which always schedules requests in logically ascending order, generally outperforms other seek-reducing algorithms. Algorithms that reduce both seek and rotational latency delays attain even higher performance, provided they explicitly recognize and exploit the prefetching cache.

Section 2 describes some of the features of modern disk drives and how they can affect scheduling accuracy. Section 3 discusses previous scheduling algorithm research. Section 4 describes our methodology, including our validation efforts and the origins of the six traces. Sections 5 and 6 present our results using random workloads and traces, respectively. Section 7 summarizes this report and suggests some avenues for future research.

2 Modern Disk Drives

A disk drive contains a set of rapidly rotating platters (on a common axis) coated on both sides with magnetic media (see figure 1). Data blocks are written in circular **tracks** on each **surface**. The minimum unit of data storage is usually a **sector**, which typically holds 512 bytes of data plus some header/trailer information (e.g., error correction data). A **cylinder** is a set of tracks (one from each surface) equidistant from the center of the disk. For example, the innermost cylinder contains the innermost track of each surface. Therefore, a physical location is defined by its cylinder, surface, and sector. The disk arm holds a set of read/write heads that can be positioned to access any cylinder. The heads are ganged together such that a given disk arm position corresponds to a specific cylinder of data. In most drives only one read/write head is active at any given time.

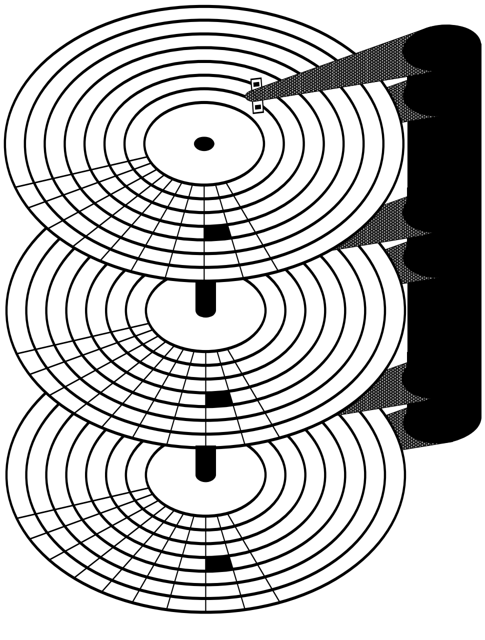


Figure 1: Disk Drive Internals

Requests serviced by the disk drive may suffer one or more mechanical delays. First, the disk arm must move (**seek**) to the target cylinder. Second, **rotational latency** is incurred while waiting for the target sectors to rotate into position under the read/write head. After these delays, the data transfer begins. Additional mechanical delays result if multiple tracks or cylinders must be accessed.

Current disk drive implementations have several features that can directly affect the accuracy of scheduling algorithms. Some features are the result of efforts to make disks self-contained or self-managed. Clever methods of increasing a disk drive's capacity and effective lifetime also have a major impact. One goal of our work is to determine exactly how these features should factor into scheduling algorithms.

2.1 Host Interface

Most disks possess sufficient on-board logic and processing power to present a relatively clean interface to the host system. Such protocols as the Small Computer System Interface (SCSI) and the Intelligent Peripheral Interface (IPI) are used by a wide variety of disk drive, adapter, and host system manufacturers. The host or intermediate controller presents a request to the disk drive in terms of the starting **logical block number** (LBN) and total request size. The details of the subsequent media access are typically hidden from the host. This approach offloads most of the management overhead associated with the actual data storage from the host or controller to the disk drive electronics. On the other hand, scheduling entities outside of the drive itself often have little or no knowledge of the data layout on the media, the status of any on-board disk cache, and the various overhead delays associated with each request.

2.2 Data Layout

Many systems use simple LBN-based approximations of seek-reducing algorithms. LBN-based scheduling relies on highly sequential logical block to physical block (LBN-to-PBN) mappings. More aggressive algorithms may require accurate knowledge of the data layout. As mentioned above, such mappings may be unavailable or very difficult to obtain. The complexity of LBN-to-PBN mappings is increased by zoned recording, track and cylinder skew, and defect management schemes.

2.2.1 Zoned Recording

To increase disk drive capacity per unit volume, most modern drives partition their set of cylinders into multiple **zones**. The number of sectors per track increases with the radius of the tracks in the zone. For example, the outermost zone may have twice as many sectors per track as the innermost zone. Within each zone, a subset of cylinders (or tracks or sectors) may be reserved for management purposes (see section 2.2.3). Without knowledge of the zone specifications and the locations of any reserved regions within each zone, a scheduler cannot determine the physical location of a given logical block.

2.2.2 Track and Cylinder Skew

Given that a positioning delay is incurred when switching heads or seeking between adjacent cylinders, the first logical block of each track or cylinder is generally offset (**skewed**) from the first logical block of the previous track or cylinder. Thus a request that crosses a track or cylinder boundary does not suffer unnecessary rotational delay due to "just missing" the next logical block and waiting almost a full rotation for it to come around again. Track and cylinder skews are often specified as an integral number of sectors and may therefore differ from zone to zone.

2.2.3 Defect Management

Disk drives are designed to survive the loss of hundreds of sectors or tracks to media corruption. In fact, new drives usually contain a list of defects identified at the factory. Defects **grown** during the course of the disk's lifetime are added to this list. LBN-to-PBN mappings are adjusted so that defective areas are not used for data storage. Such adjustments fall under the categories of **slipping** and **reallocation**.

Sector or track slipping takes place when a disk is formatted. In the case of sector slipping, the formatting process skips each defective sector when initializing the LBN-to-PBN mapping. If a defective sector is in the "middle" of a track, the mapping is adjusted so that the sectors immediately adjacent to the defective sector will be logically sequential. Track slipping follows the same routine, except that entire tracks are skipped if they contain any defective media. Each slipped region changes the mapping of subsequent logical blocks, reducing the accuracy of a scheduler that relies on static knowledge of the cylinder and/or the rotational position of each block. Extra space is reserved at the end of certain tracks, cylinders, or zones to contain the spillover from the slipping process. If these **spare** regions are organized as a set of cylinders at the end of each zone, the seek delay will increase whenever a zone boundary must be crossed.

Reallocation occurs when defective sectors (or tracks) are discovered during normal disk usage. The affected LBN's are dynamically remapped to spare regions. Any subsequent accesses are redirected to the appropriate spare sectors. The disks we examined contained very few defects, producing no noticeable effects on disk performance.

2.3 On-Board Cache

The use of memory within embedded disk drive control logic has progressed from small, speed-matching buffers to dynamically-controlled caches containing megabytes of storage. On-board disk logic can automatically prefetch data into the cache to more quickly satisfy sequential read requests. For example, a disk might take 20 ms to service a read request that requires media access. A subsequent sequential read might take only 2 ms if the requested data blocks have been prefetched into the on-board disk cache.

Some disks also support the ability to signal request completion as soon as all write data has been transferred to the cache (but not necessarily to the disk media). The on-board logic takes care of moving the data from the cache onto the media at some later point. Note that there are reliability issues that must be considered when performing such **fast writes**.

By utilizing a simple memory management algorithm in the cache control logic, disks with so-called **zero-latency** read or write capability can reduce their average response times by transferring data to or from the media in other than FIFO order. For example, if an entire track of data is to be read from the disk, a disk capable of zero-latency reads begins transferring data from the media as soon as any seek or head switch delays are complete.

Segmented disk caches contain multiple "cache lines," to use the common processor terminology. In addition, some disks employ algorithms that use the history of recent requests to dynamically modify the number of segments (and the segment size) in an attempt to improve cache performance.

The existence of an on-board cache affects scheduling activities in several ways. First, if the disk is configured to prefetch aggressively, the current position of the read/write head cannot necessarily be determined by knowledge of the location and length of the most recent request. The disk will continue reading beyond the end of a previous read request, possibly switching read/write heads or seeking to the next cylinder. Second, cached data can be accessed far more quickly than data obtained from the disk media. Requests that can be satisfied by the cache could therefore be given higher priority in order to reduce the average response time. Scheduling issues related to fast writes, zero-latency access, and cache segmentation are left for future work.

2.4 Overhead and Positioning Delays

There are several control and communication delays associated with servicing a disk request. Aggressive algorithms must plan for such delays or the resulting schedules may actually be worse than those produced by simpler algorithms. For example, if the initiation overhead is 1 millisecond and the rotational speed of the disk is 5400 RPM (11.1 ms per rotation), a media access that would begin in 0.5 milliseconds without any overhead (given the current rotational position) will actually begin in 11.6 milliseconds. If the initiation overhead were taken into account, a request with a shorter delay might be chosen.

The seek profile for a disk drive is a complex function, usually containing an initial sawtoothed region for very short seeks, a middle region approximated by a square root function and a large linear region for long seeks (see appendix A). To accurately predict and compare the positioning times of individual pending requests, a scheduler must have a relatively accurate approximation of this complex function or suboptimal schedules will result.

Lastly, the rotational speed of individual disk drives will vary slightly from factory specifications, and may even vary from moment to moment. Most current drives can pass and receive rotational synchronization signals. By monitoring these signals, a scheduler external to the disk can keep track of the exact rotational position of each disk and make accurate rotational latency predictions.

3 Disk Scheduling Algorithms

Numerous studies have shown that a simple *First Come First Served* (FCFS) disk scheduling policy often results in suboptimal performance. Many scheduling algorithms have been proposed that achieve higher performance by taking into account information about individual requests and the current state of the disk subsystem.

3.1 Seek Delay Reduction

Over 25 years ago, [Denn67] analyzed the advantages of a *Shortest Seek Time First* (SSTF) policy. This algorithm chooses the next request to service by selecting the pending request that will incur the smallest seek delay given the current disk arm position. It is usually infeasible to predict the exact seek times between cylinders, but SSTF may be closely approximated by using seek distances. SSTF reduces the average response time over a wide range of workloads. However, the potential for starvation of individual requests increases with disk utilization, resulting in excessive response time variance. Given a heavy workload, the disk arm tends to hover over a subset of the cylinders in an attempt to exhaust all requests to that region, thereby starving any requests outside of that space.

[Denn67] also examined the SCAN or "elevator" algorithm, which provides a lower response time variance than SSTF with only a marginal increase in the average response time (for the random workloads studied). This algorithm is named for the way the disk arm shuttles back and forth across the entire range of cylinders, servicing all requests in its path. It only changes direction at the innermost and outermost cylinder of the disk. Because the disk arm passes over the center region of the disk at more regular intervals than the edges, requests to cylinders in the middle of the disk receive better service. However, every cylinder **is** reached during both phases of the scan. As a result, SCAN resists starvation more effectively (i.e., has lower response time variance) than SSTF.

Several variations of the SCAN algorithm have been proposed. The *Cyclical SCAN* algorithm (C-SCAN) replaces the bi-directional scan with a single direction of arm travel [Seam66]. When the arm reaches the last cylinder, a full-stroke seek returns it to the first cylinder of the disk without servicing any requests along the way. C-SCAN treats each cylinder equally, rather than favoring the center cylinders as SCAN does. The LOOK algorithm, another SCAN variation, changes the scanning direction if no pending requests exist in the current direction of travel [Mert70]. C-SCAN and LOOK can be combined, resulting in the C-LOOK algorithm.

VSCAN(R), proposed in [Geis87], creates a continuum of algorithms between SSTF and LOOK. The R parameter denotes how strongly biased the scheduler is towards maintaining the current direction of travel. VSCAN(0.0) is equivalent to SSTF, and VSCAN(1.0) reduces to LOOK. [Geis87] suggests that VSCAN(0.2) provides a good balance between the average response time of SSTF and the starvation resistance of LOOK. The CVSCAN(N,R) variation described in [Geis87a] augments VSCAN by examining more than just the "closest" request in either direction from the current disk arm position. Instead, the average seek distances to the "closest" N requests in each direction are used for making scheduling decisions (i.e., CVSCAN(1,R) = VSCAN(R)).

3.2 Positioning Delay Reduction

Reducing the average seek delay requires only knowledge about the relative seek distances between requested data. In order to also account for rotational latency, more complete information about the actual mapping of data blocks onto the media is necessary. In addition, the current physical location of the active read/write head must be known. Given this information, the scheduler can choose the request with the minimum *positioning* delay (i.e., combined seek and rotational latency). This policy was denoted as *Shortest Time First* (STF) in [Selt90] and *Shortest Access Time First* (SATF) in [Jaco91], but we use the term *Shortest Positioning Time First* (SPTF) to clarify the exact focus of the algorithm.

3.3 Batch and Aging Algorithms

Several modifications to the above algorithms have been proposed to reduce response time variance. The resulting algorithms can be generally classified as either *batch* or *aging* algorithms. Batch algorithms service a subset of requests satisfying certain criteria before moving on to other requests. Possible criteria include the position of the request in the queue and/or the specific cylinder corresponding to the request [Teor72, Jaco91, Selt90]. Aging algorithms give priority to requests that have been in the pending queue for excessive periods of time. If too much weight is given to the queue time (age) component, such algorithms degenerate into FCFS. The priority may slowly increase as the request ages, or a time limit may be set after which requests are given a higher priority [Selt90, Jaco91].

4 Methodology and Validation

4.1 Disk simulator

We have developed a detailed, strongly validated disk simulator to compare disk scheduling algorithms. The simulator accurately models zoned recording, spare regions, defect slipping and reallocation, disk buffers and caches,

HP C2247 Disk Drive	
Formatted Capacity	1.05 GB
RPM	5400
Data Surfaces	13
Cylinders	2051
Sectors	2054864
Zones	8
Sectors/Track	56-96
Interface	SCSI-2
256 KB Cache, 1-4 Segments	
Track Sparing/Reallocation	

Table 1: HP C2247 Disk Drive Basic Parameters

various prefetch algorithms, fast write, bus delays, and control and communication overheads. For this report, the simulator was configured to model the HP C2240 series of disk drives [HP92]. Some of the basic specifications for the HP C2247 drive [HP91] are provided in table 1.

In order to accurately model this drive, an extensive set of parameters was obtained from published documentation and by monitoring the SCSI activity generated by an HP C2247 disk. The experimental system used was an NCR 3550 multiprocessor using the NCR 53C700 SCSI I/O Processor. System activity was tightly controlled during the monitoring process to reduce extraneous host delays. We extracted a seek curve, control and communication overhead values, bus transfer rates, and cache management strategies. We also determined the exact LBN-to-PBN mappings for several of these disks, which provided information about zoning, sparing, and existing defects. Appendix A provides a complete description of the various parameters.

The simulator was validated by exercising an actual HP C2247 and capturing traces of all SCSI activity. After extracting appropriate information, each traced request stream was run through the simulator, using the observed request interarrival delays. This process was repeated for several synthetic workloads with varying read/write ratios, arrival rates, request sizes, and degrees of sequentiality and locality. The average response times of the actual disk and the simulator match to within 0.8% in all cases. Unpredictable (from the disk's view) host delays partially account for the difference. Greater insight can be achieved by comparing the response time distributions [Ruem94]. Figure 2 shows distributions of measured and simulated response times for a sample validation workload of 10,000 requests. As with most of our validation results, one can barely see that two curves are present. [Ruem94] defines the root mean square horizontal distance between the two distribution curves as a demerit figure for disk model calibration. The demerit figure for the validation run shown in figure 2 is 0.07 ms, or less than 0.5% of the average response time. The worst-case demerit figure observed over all validation runs was only 1.9% of the corresponding average response time.

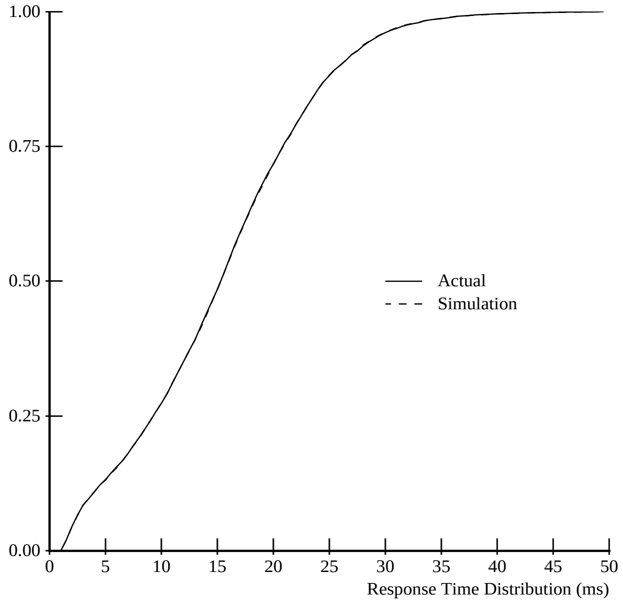


Figure 2: Response Time Distributions for a Validation Workload (50% reads, 30% sequential, 30% local [normal with 10000 sector variance], 8KB mean request size [exponential], 0-22 ms interarrival [uniform])

4.2 Workloads

We use synthetically generated random workloads and extensive traces captured from actual systems. We will describe the traces only briefly as they have been described elsewhere in more detail [Rama92, Ruem93]. The traced workloads span a broad range of environments, and each trace is at least a full workshift (8 hours) in length.

Two of the traces come from Hewlett-Packard systems running HP-UX^{TM}, a version of the UNIX^{TM} operating system [Ruem93]. *Cello* comes from a server at HP Labs used for program development, simulation, mail, and news. *Snake* is from a file server at the University of California, Berkeley used primarily for compilation and editing. While these traces are actually two months in length, we report data for a single week-long snapshot (5/30/92 to 6/6/92).

The other four traces are from commercial VAX^{TM} systems running the VMS^{TM} operating system [Rama92]. *Air-Rsv* is from a transaction processing environment in which approximately 500 travel agents made airline and hotel reservations. *Sci-TS* is from a scientific time-sharing environment in which analytic modeling software and graphical and statistical packages were used. *Order* and *Report* are from a machine parts distribution company. *Order* was collected during daytime hours, representing an order entry and processing workload. *Report* was collected at night, capturing a batch environment in which reports of the day's activities were generated.

4.3 Disks

We chose to simulate members of the HP C2240 series of drives for three reasons. First, our simulator correctly models the behavior of this line of drives. Second, these drives have most of the advanced features described in section 2. Third, we do not have detailed specifications for the disks used on the traced systems, nor access to such disks in order to obtain this information first-hand.

This disk substitution leads to two significant difficulties. First, our base disk (the HP C2247) has a different storage capacity than the disks used by the traced systems. To better match the size of the simulated disks to those in the traced systems, we modify the number of platters (5 for *Sci-TS*, *Order*, and *Report*; and 9 for *Cello*, *Snake*, and *Air-Rsv*) to create disks large enough to contain the active data without excessive unused media. We feel that this is a reasonable approach, as a production line of disks often differs only in the number of physical platters.

The second and more important difficulty is that HP C2240 disks service requests at a different rate than the disks used in the traced systems. An ideal model would incorporate appropriate feedback effects between request completions and subsequent arrivals. Unfortunately, information about the dependencies between individual request arrivals and previous request completions is not present in the traces. Developing a methodology for realistically modeling such feedback effects is an area for future research. The approach used here was to scale the traced interarrival times and thereby produce simulation results for a range of average arrival rates. When the scale factor is one, the simulated interarrival times match those traced. When the scale factor is two, the traced interarrival times are halved (doubling the average arrival rate). However, even with an identity scaling factor the workload would undoubtedly have been different if the system traced were using HP C2240 disk drives. This is a common problem with this type of trace-driven simulation (which behaves as an open system), but we believe that the qualitative results and analysis are valid.

4.4 Metrics

The average response time (across all disks) is the primary metric used for comparing the various scheduling algorithms. The squared coefficient of variation (σ^2/μ^2) is also used, as in [Teor72]. Given a constant average response time, a decrease in the coefficient of variation implies reduced response time variance (i.e., improved starvation resistance).

5 Synthetic Workloads

Synthetically generated random workloads were used in order to replicate previous work and obtain a starting point for further experiments. The request stream consisted of 8 KB accesses (a typical file system block size) uniformly distributed across the available logical space. The request interarrival times were exponentially distributed, with the mean varied to generate lighter or heavier workloads. The ratio of reads to writes was set to 2:1. Unless otherwise noted, the disk model was configured with 38 slipped tracks (matching the largest defect list found in our experimental HP C2247 disk drives). Each data point is the average of at least three separate runs of 250,000 disk requests, corresponding to simulated workloads of 50 to 400 hours of activity.

The results are partitioned into three groups based on the level of information available to the scheduling algorithms: scheduling based on LBN, scheduling given an accurate LBN-to-PBN mapping, and scheduling with full knowledge (including current read/write head position, all overhead delay values, cache contents, etc.).

5.1 Scheduling by Logical Block Number

Even if the scheduler has little or no knowledge of the LBN-to-PBN mapping for a given disk, it can approximate seek delays via the "distance" between logical block numbers for individual requests. For example, if a request to logical block 100 has just completed, an LBN-based C-LOOK scheduler will select a pending request for logical block 150 over one for logical block 200. The accuracy of this approximation is determined by the choice of scheduling algorithm, the variance in sectors per track between zones, the defect slipping/reallocation schemes, and any cache prefetching activity.

Figure 3a graphs the average response times for FCFS, LOOK, C-LOOK, SSTF and VSCAN(0.2) for a range of average arrival rates. As expected, FCFS quickly saturates as the workload increases. Since C-LOOK is designed to reduce the variance of response times (as well as the mean), its average response times runs 5-10% higher than those of LOOK, SSTF and VSCAN(0.2) for medium and heavy workloads. For the sub-saturation workloads, SSTF provides a slightly lower mean response time (up to 1% lower than VSCAN(0.2) and 2% lower than LOOK). For workloads at the edge of saturation, SSTF loses to VSCAN(0.2) and LOOK by similar margins.

The 95% confidence intervals shown in figure 3b lend support to the above observations about FCFS and C-LOOK. However, the intervals for LOOK (which are typical for SSTF and VSCAN(0.2) as well) prevent us from predicting the relative positions of LOOK, SSTF and VSCAN(0.2) for a given arrival rate.

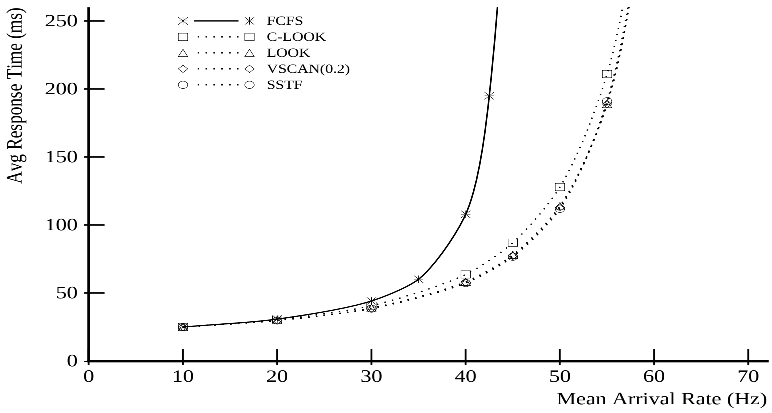
Figure 3c shows the squared coefficients of variance (σ^2/μ^2) for the same set of experiments. FCFS has the lowest coefficient for lighter workloads, as would be expected. As FCFS begins to saturate and its response time variance expands, C-LOOK emerges as the algorithm with the best starvation resistance. SSTF, while providing the best average response time in most cases, is highly susceptible to starvation (as is evident by its larger coefficient of variation).

Similar results are reported in Teorey and Pinkerton's comparison of FCFS, LOOK, C-LOOK, and SSTF [Teor72]. They determined that SSTF and LOOK (SCAN) provide the best average response times, while C-LOOK has superior starvation resistance for the heavier workloads. The simulation study in [Geis87] shows that the mean response times of SSTF, SCAN, and VSCAN(R) match to within a few percent when requests are distributed across the disk in either a uniform, unimodal, or bimodal manner. The observed starvation resistance of these algorithms also compares well with our results. In [Selt90], SSTF (referred to as SSF) and C-SCAN are shown to provide roughly equivalent disk utilization, but SSTF again provides inferior starvation resistance.

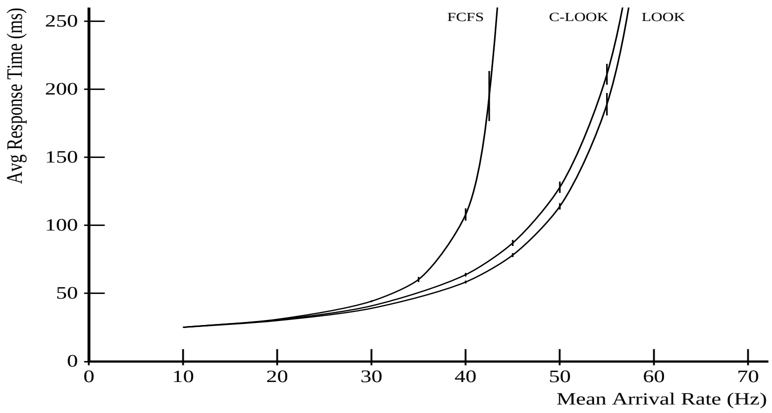
5.2 Scheduling with a Known Mapping

If the scheduler has knowledge of LBN-to-PBN mappings, it can more accurately predict seek delays and thereby produce better schedules. However, a heuristic for choosing among requests within a cylinder must be designed. We use a C-LOOK scheme in order to take advantage of the HP C2247's prefetching cache (see section 6). Thus each scheduling algorithm uses C-LOOK as long as there are requests within the current cylinder to satisfy, and then one of the various experimental algorithms to select the next cylinder to service.

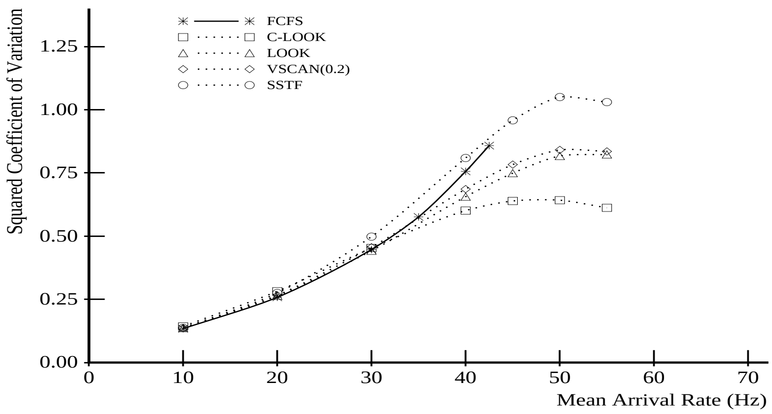
The LBN-based and full-map versions of each algorithm produce almost identical performance, as measured by both average response times and coefficients of variation. As random workloads contain few sequential requests, prefetching provides little or no performance improvement. The cylinder heuristic also has little effect, as individual



(a) Average Response Time



(b) 95% Confidence Intervals (Average Response Time)



(c) Squared Coefficient of Variation

Figure 3: LBN-Based Algorithm Performance using a Synthetic (Random) Workload

cylinders rarely contain multiple pending requests. Figures 4a and 4b display the average response times and squared coefficients of variation for both the LBN-based algorithms and the algorithms utilizing the full LBN-to-PBN mapping. The only visible deviation is a slight increase in SSTF's coefficient of variance for the full-map algorithm. This is reasonable, as SSTF utilizes mapping knowledge strictly to reduce response times, often at the expense of increased request starvation.

To determine the influence of excessive defects on the scheduler's accuracy, we modified the simulator to model an HP C2247 with half of its spare tracks (450 out of 900) filled due to randomly grown defects. To maximize the perturbation in the LBN-to-PBN mapping, all of the defects were dynamically reallocated. Even with over 2% of the disk's tracks remapped, full mapping information provides little improvement in performance. SSTF, the algorithm most sensitive to the LBN-to-PBN mapping, improves by less than 1%.

We conclude that maintaining a full LBN-to-PBN mapping is not justified for seek-reducing algorithms when scheduling random workloads.

5.3 Scheduling with Full Knowledge

Given sufficient computational resources, a full LBN-to-PBN mapping, accurate values for various control and communication overheads, and some indication of the drive's current rotational position, the scheduler can select the pending request which will incur the smallest total positioning delay (seek and rotational latency). As with SSTF, SPTF is highly susceptible to request starvation. For this reason, we also examined an aging algorithm based on SPTF. Denoted as *Aged Shortest Positioning Time First*, this algorithm is equivalent to the ASATF algorithm proposed in [Jaco91]. [1] ASPTF adjusts each positioning delay prediction (T_{pos}) by subtracting a weighted value corresponding to the amount of time the request has been waiting for service (T_{wait}). The resulting effective positioning delay (T_{eff}) is used in selecting the next request to issue:

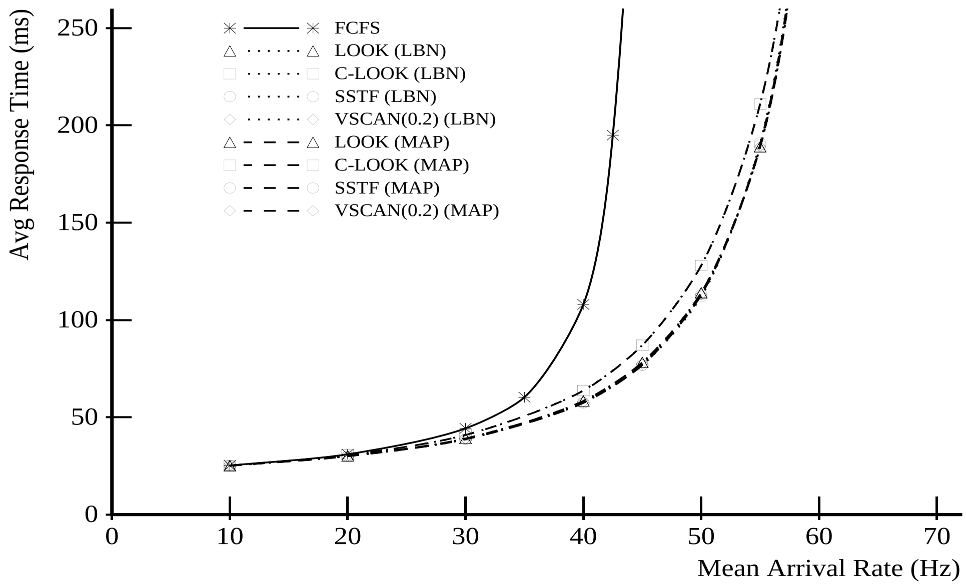
$$T_{eff} = T_{pos} - (W * T_{wait})$$

The weight suggested in [Jaco91] translates approximately to ASPTF(6.3). Figures 5a and 5b present the performance metrics for values of W ranging from 0 (SPTF) to 30. As W increases, the average response time slowly grows, but the response time variance drops significantly. In all cases, the algorithms that schedule based on predicted positioning times have lower average response times than those based on predicted seek times. For higher values of W, ASPTF suffers a sharp increase in average response time when the disk begins to saturate. For a sufficiently large W, ASPTF degenerates into FCFS.

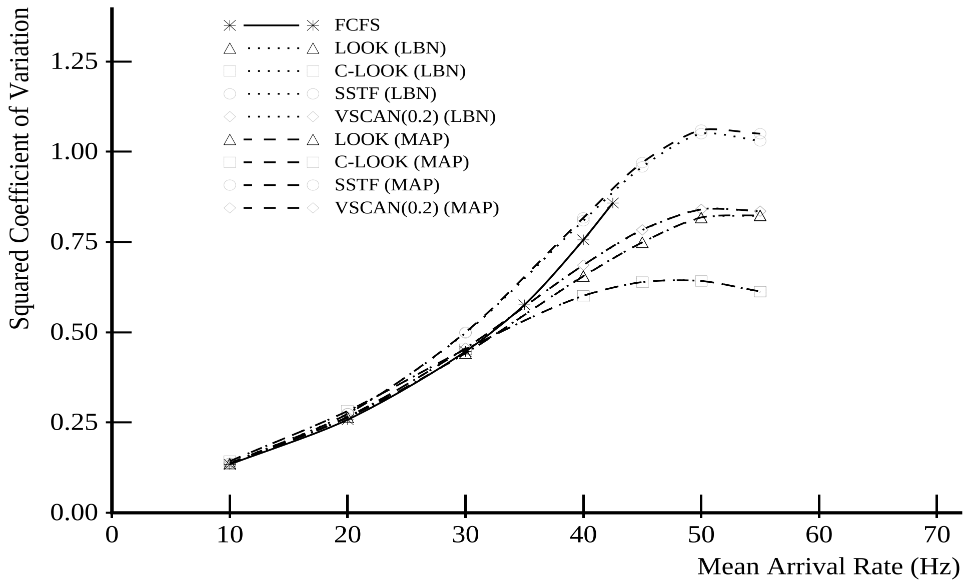
An appropriate aging factor may result in an algorithm with the average response time of SPTF and the starvation resistance of C-LOOK. For example, ASPTF(6) and ASPTF(12) match the performance of SPTF for all but the heaviest workloads, yet have much better starvation resistance. In fact, ASPTF(12) has lower response time variance than C-LOOK, even though its coefficient of variation is slightly higher.

Selecting the request with the smallest positioning delay entails significant computational effort (especially for large queues), cf. [Jaco91].

[1] We chose this algorithm over the WSTF algorithm suggested in [Selt90] because WSTF is somewhat insensitive to differences in predicted positioning times when comparing requests whose waiting times are near the aging limit.

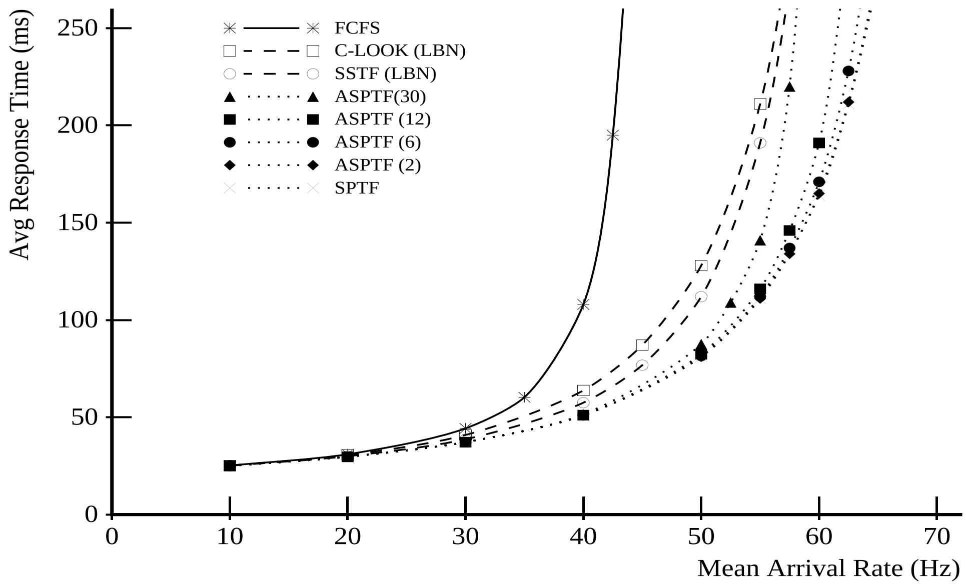


(a) Average Response Time

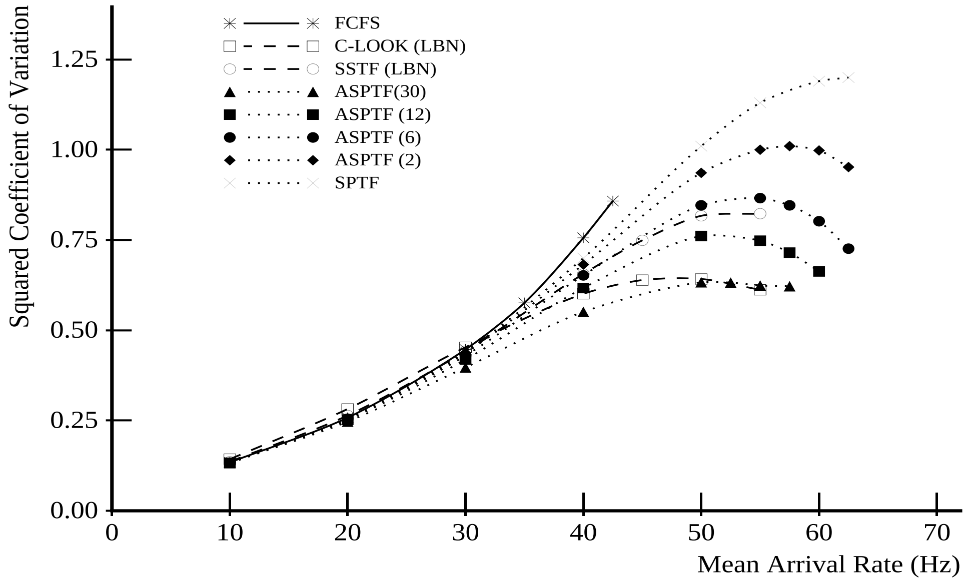


(b) Squared Coefficient of Variation

Figure 4: Full-Map and LBN-Based Algorithm Performance using a Synthetic (Random) Workload



(a) Average Response Time



(b) Squared Coefficient of Variation

Figure 5: Full-Knowledge Algorithm Performance using a Synthetic (Random) Workload

Trace Name	Length (hours)	Number of Disks	Number of Requests	Average Size (KB)	Total Read Percentage	Seq. Read Percentage
Cello	168	8	3,262,824	6.3	46.0%	2.5%
Snake	168	3	1,133,663	6.7	52.4%	18.6%
Air-Rsv	9	16	2,106,704	5.1	79.3%	7.8%
Sci-TS	19.6	43	5,187,693	2.4	81.5%	13.8%
Order	12	22	12,236,433	3.1	86.2%	7.5%
Report	8	22	8,679,057	3.9	88.6%	3.8%

Table 2: Basic Characteristics of the Traces

6 Traced Workloads

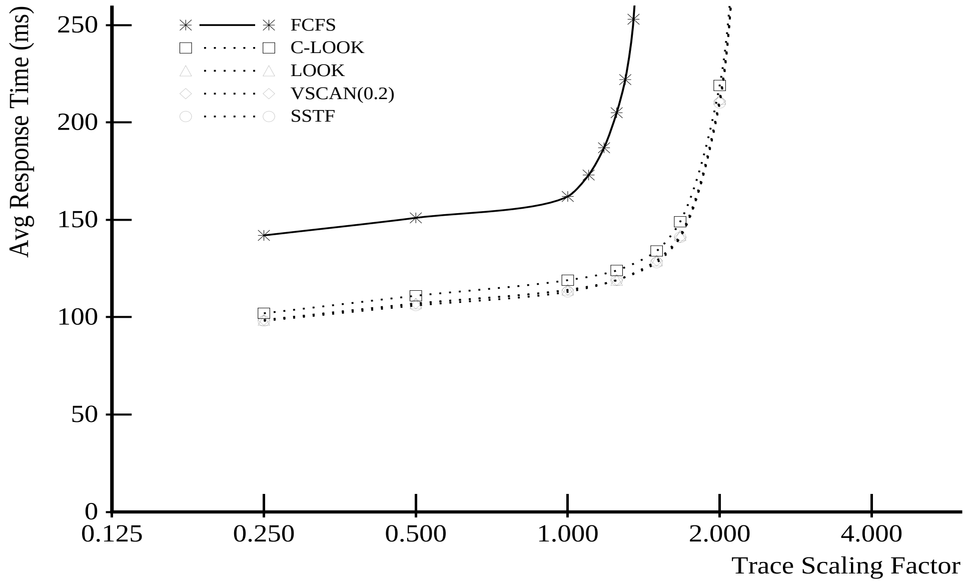
To gain an understanding of how various scheduling algorithms perform under actual user workloads, we used traces of disk activity collected from systems in use at various industrial and research installations. The environments in which the traces were captured are described briefly in section 4. Some basic characteristics of the traces are given in table 2. The traces vary widely in read/write ratios, access sizes, arrival rates, degrees of sequentiality, and burstiness. The environments and characteristics of these traces are described more thoroughly in [Rama92, Ruem93].

The performance graphs in this section use a range of trace scaling factors. Note that the scaling factor (the X-axis) is shown in $\log_2$ scale. The workloads with an identity scaling factor correspond to the traced request streams.

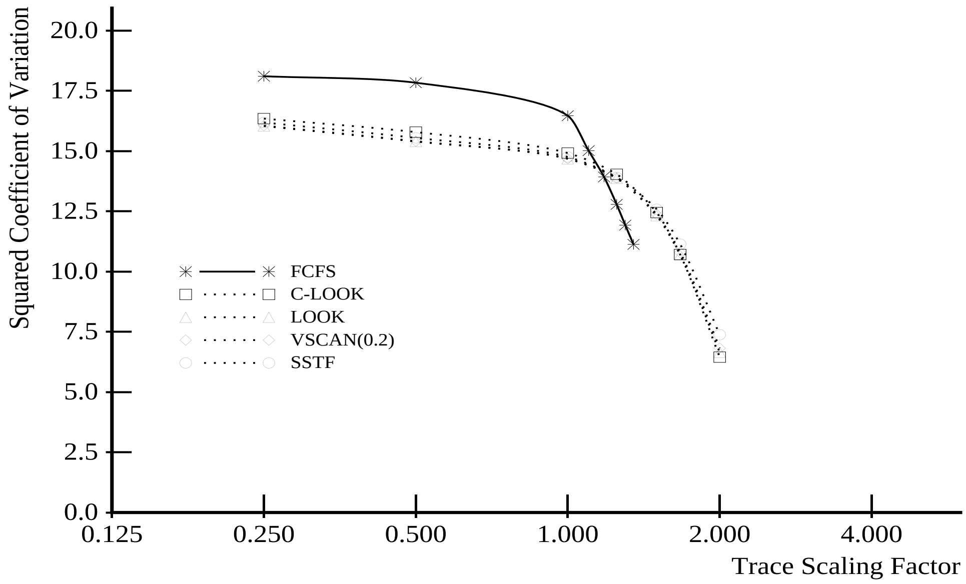
6.1 Scheduling by Logical Block Number

Figures 6-11 show the average response times and the corresponding squared coefficients of variation for LBN-based scheduling of the six traces. As above, FCFS scheduling performs poorly for all but the lightest workloads. The relative performance of the other algorithms, however, differs from that observed for the random workloads. The results indicate that the HP C2247's prefetching cache plays a large role in determining algorithm effectiveness.

Read requests for cached data can be serviced much faster than those which must access the disk media. If read requests arrive at the disk in logically ascending order, some portion (or all) of the data for each sequential request may already be in the prefetching cache when the request arrives. Therefore, algorithms that preserve any existing read sequentiality achieve higher performance for workloads containing a significant fraction of sequential or highly local read requests.

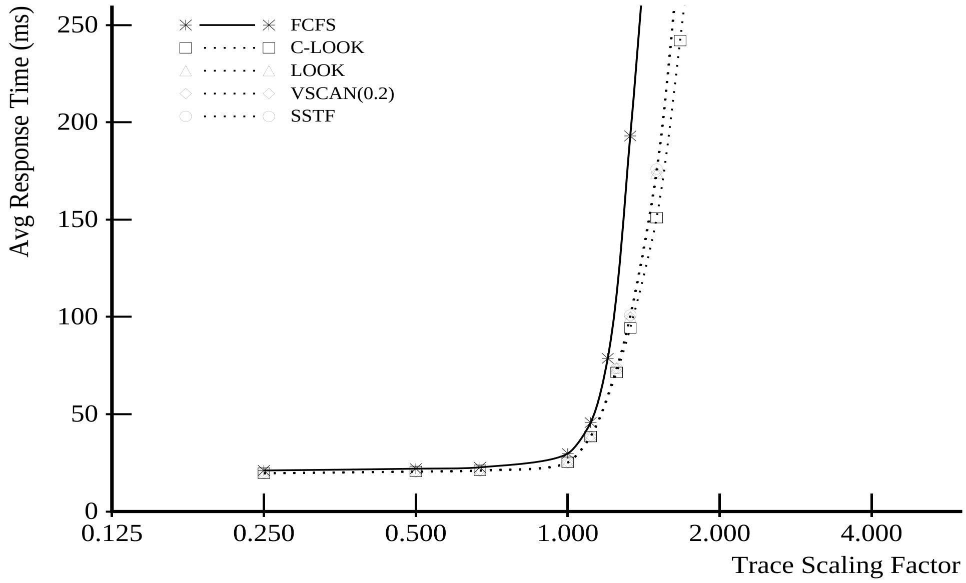


(a) Average Response Time

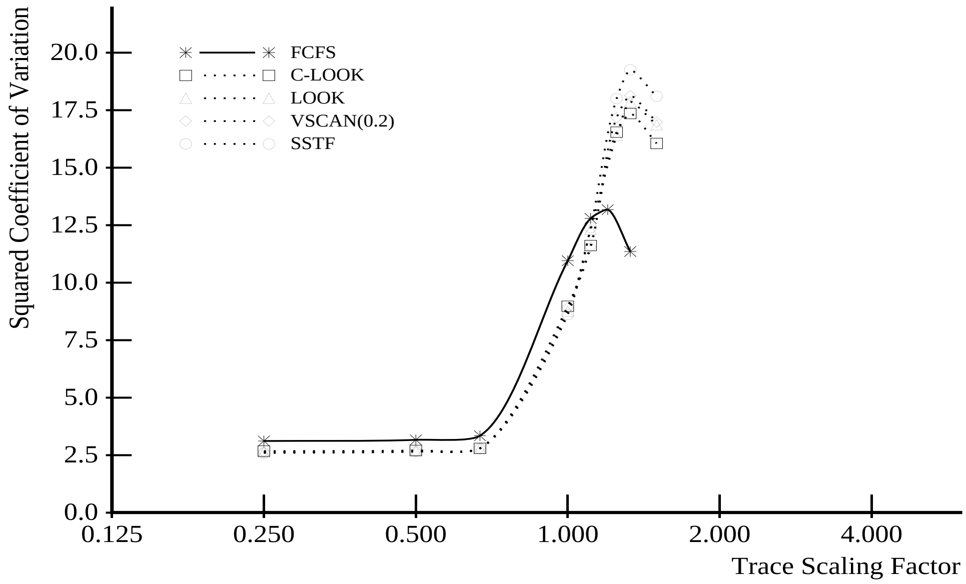


(b) Squared Coefficient of Variation

Figure 6: LBN-Based Algorithm Performance using the *Cello* Trace

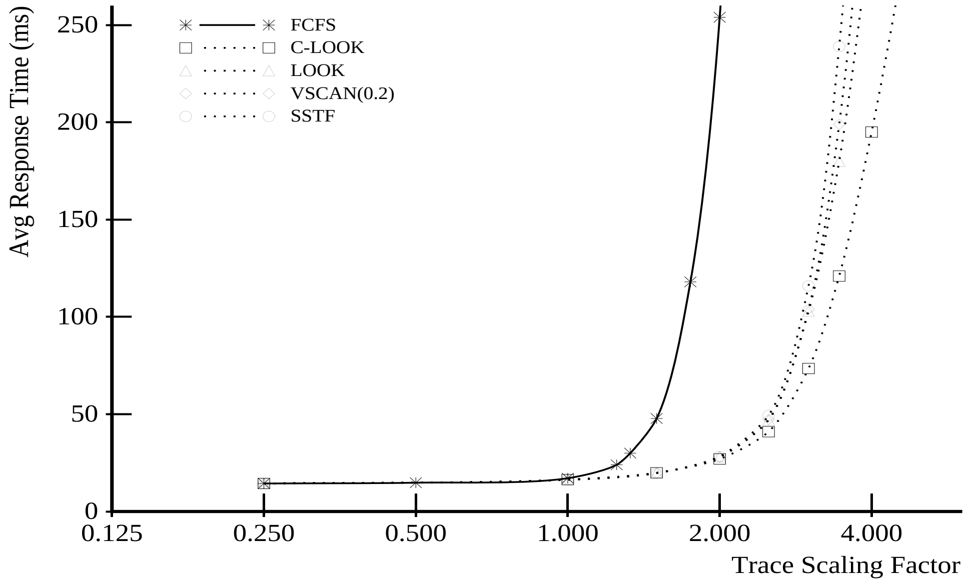


(a) Average Response Time

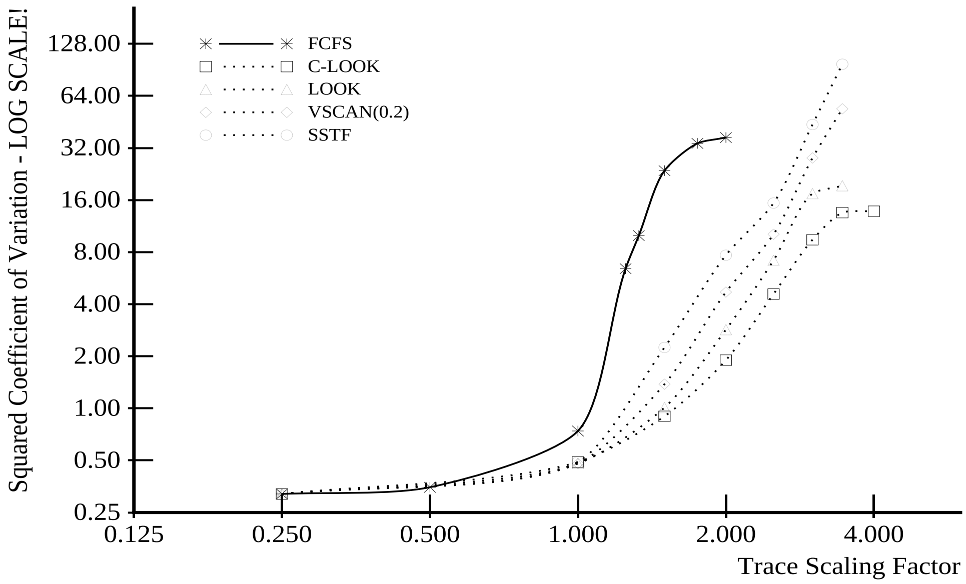


(b) Squared Coefficient of Variation

Figure 7: LBN-Based Algorithm Performance using the *Snake* Trace

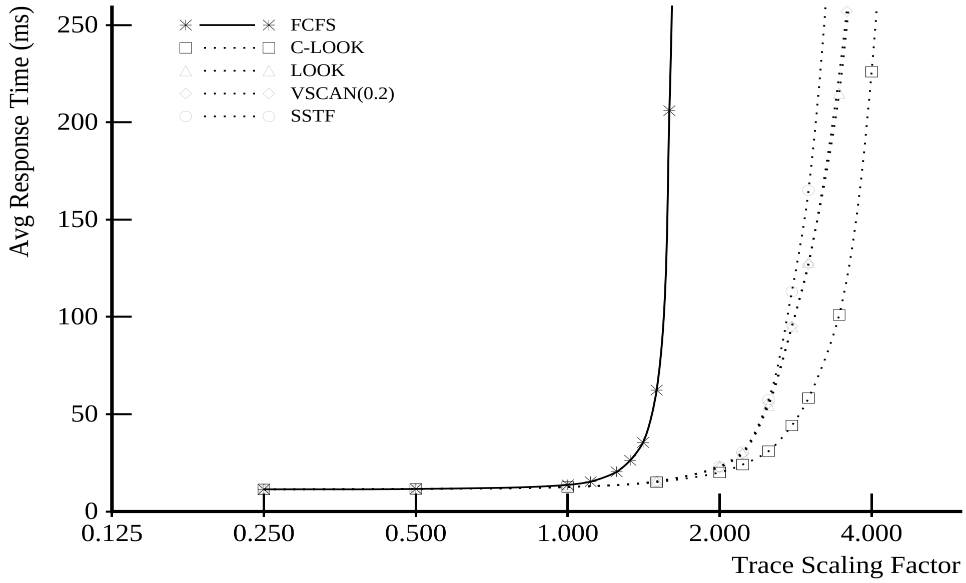


(a) Average Response Time

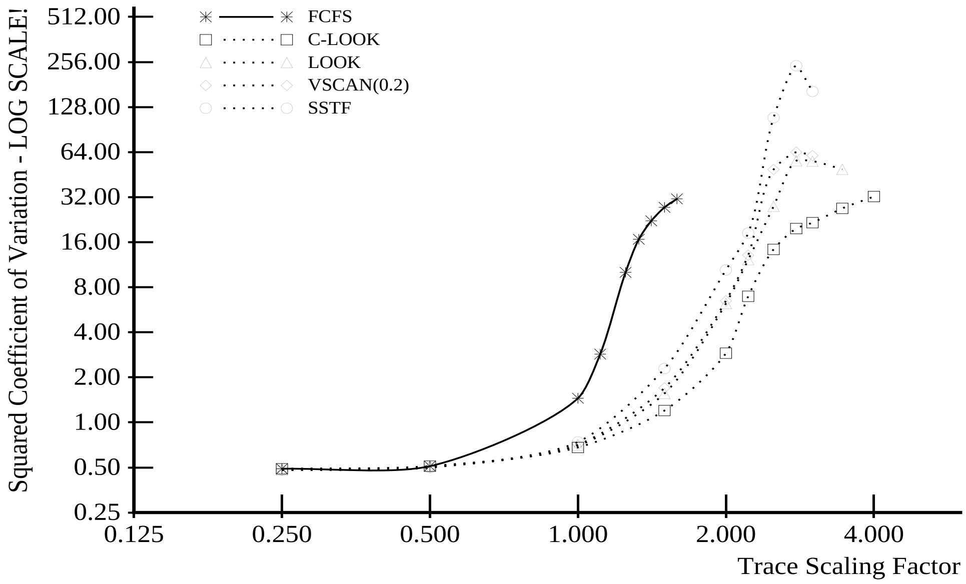


(b) Squared Coefficient of Variation

Figure 8: LBN-Based Algorithm Performance using the *Air-Rsv* Trace

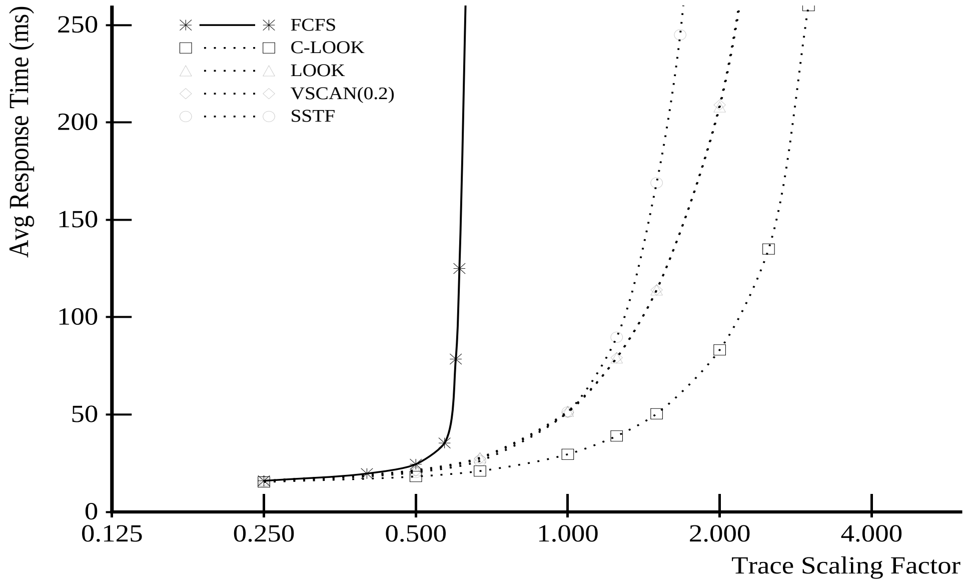


(a) Average Response Time

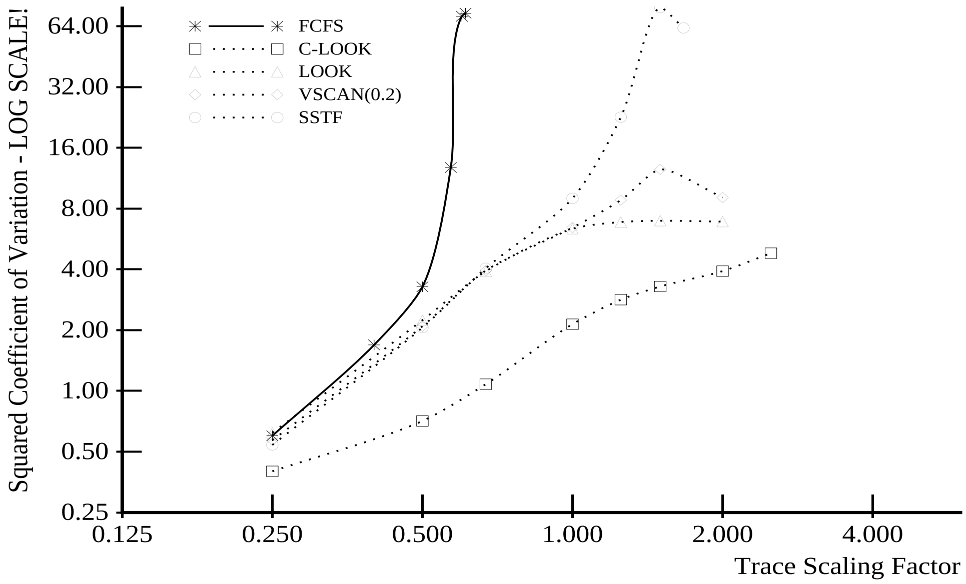


(b) Squared Coefficient of Variation

Figure 9: LBN-Based Algorithm Performance using the *Sci-TS* Trace

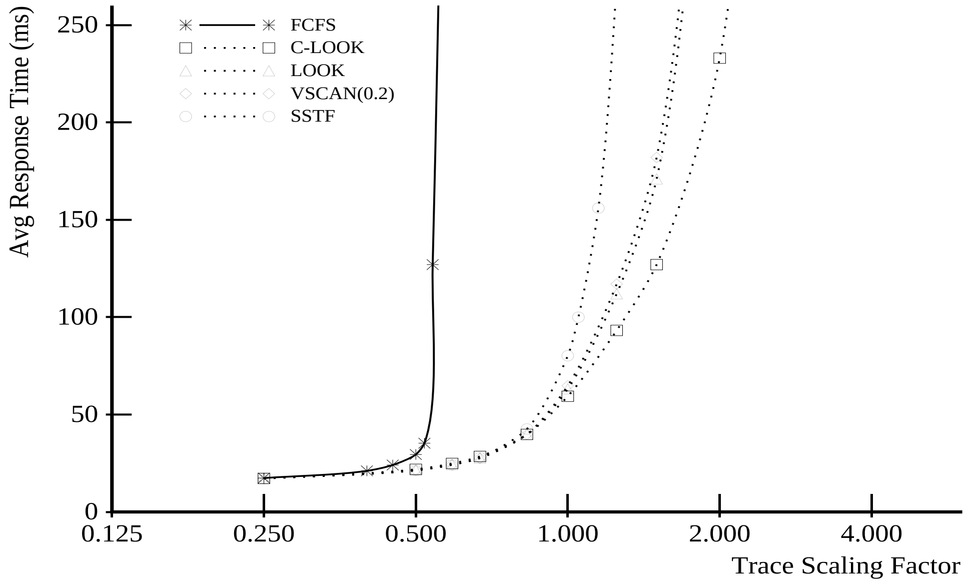


(a) Average Response Time

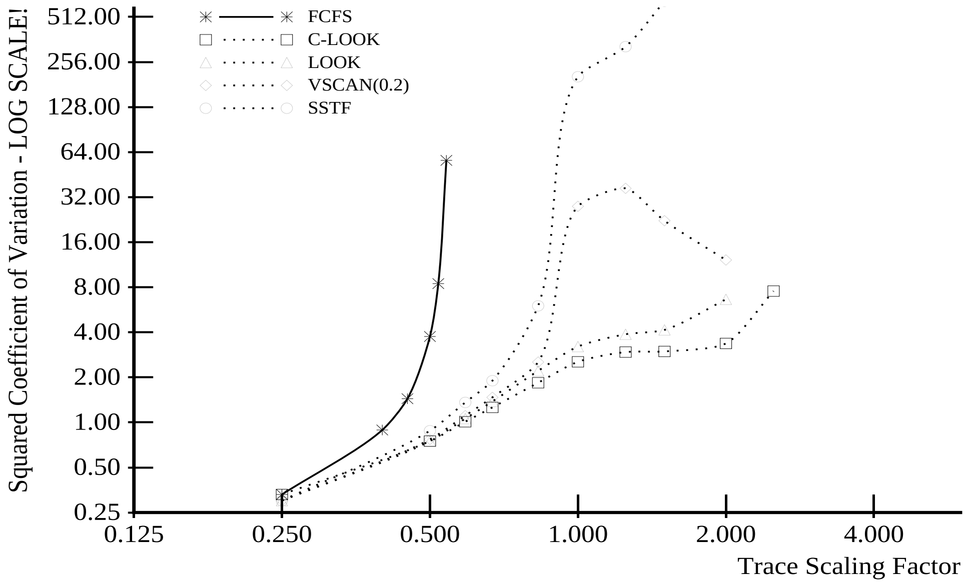


(b) Squared Coefficient of Variation

Figure 10: LBN-Based Algorithm Performance using the *Order* Trace



(a) Average Response Time



(b) Squared Coefficient of Variation

Figure 11: LBN-Based Algorithm Performance using the *Report* Trace

Trace	Scale Factor	C-LOOK	LOOK	VSCAN(0.2)	SSTF
Cello	1.5	134	129	129	128
Snake	1.25	71.5	73.7	73.7	73.6
Air-Rsv	2.5	44.2	51.2	51.2	53.9
Sci-TS	2.5	31.0	54.7	55.7	57.5
Order	1.0	29.6	51.7	51.1	51.3
Report	1.0	59.4	63.2	64.8	80.2

(a) Average Response Times (ms) using a Prefetching On-Board Disk Cache

Trace	Scale Factor	C-LOOK	LOOK	VSCAN(0.2)	SSTF
Cello	1.25	145	134	134	133
Snake	0.34	65.5	65.5	65.4	65.2
Air-Rsv	2.0	44.8	43.1	42.9	43.1
Sci-TS	1.69	92.6	64.4	63.6	61.8
Order	0.67	57.1	52.8	52.2	51.4
Report	0.71	60.2	55.3	55.5	54.6

(b) Average Response Times (ms) using a Speed-Matching On-Board Disk Buffer

Table 3: Sample Average Response Times for LBN-Based Algorithms (taken from the "knees" of the Average Response Time curves for the six traces)

Table 3a lists a sample point from the knee of each average response time graph. The LBN-based C-LOOK algorithm, which always schedules requests in logically ascending order, provides higher performance than LOOK, VSCAN(0.2) and SSTF for all traces except *Cello*. LOOK and VSCAN(0.2) generally outperform SSTF, as they schedule sequential requests in logically ascending order during the "ascending" half of the scan cycle. However, LOOK, VSCAN(0.2), and SSTF all exhibit similar performance when the last of a sequence of requests is scheduled before the first. When this occurs, the requests are scheduled to the disk in logically **descending** order, negating the performance advantage of the prefetching cache.

Trace	Scale Factor	C-LOOK	LOOK	VSCAN(0.2)	SSTF
Cello	1.0	11.5%	11.5%	11.5%	11.5%
Snake	1.25	21.8%	21.6%	21.6%	21.7%
Air-Rsv	2.5	15.0%	13.8%	13.7%	13.7%
Sci-TS	2.5	26.9%	25.7%	25.5%	25.3%
Order	1.0	17.6%	14.8%	14.8%	14.8%
Report	1.0	14.5%	14.2%	14.1%	13.7%

Table 4: On-board Disk Cache Hit Rates for LBN-Based Algorithms

For *Cello*, C-LOOK produces a higher average response time than the other seek-reducing algorithms. The *Cello* environment is dominated by large bursts of write requests to a single disk [Ruem93]. Almost half of the requests are serviced by this disk, with maximum queue lengths approaching 1000 at the identity scaling factor. In addition, this trace contains the smallest fraction of sequential read requests, thus benefiting the least from the prefetching cache. Table 4 lists the disk cache hit rates for each of the sample points given in table 3a. For all workloads except *Cello*, the C-LOOK algorithm exhibits a higher hit rate in the on-board disk cache. Note that even a small improvement in cache hit rate can significantly affect performance, due to the long mechanical delays incurred when accessing the media (versus accessing the cache). A decrease in service time for even a single request can result in decreased queue times for numerous pending requests.

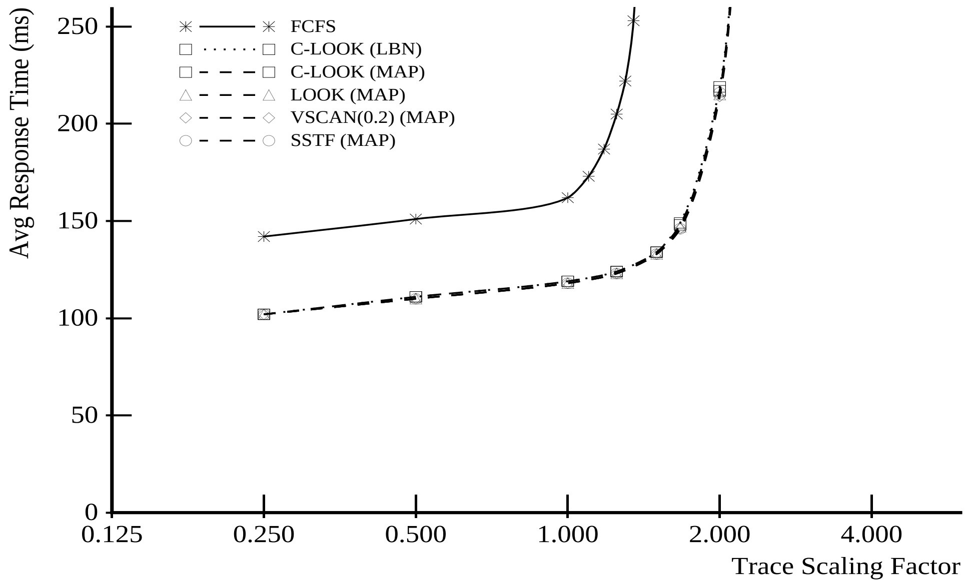
Additional evidence of the benefits of optimizing for a prefetching cache can be found by comparing tables 3a and 3b. The latter table lists sample points from a set of experiments where the disk "cache" is used only as a speed-matching buffer. Under this restriction, the performance of C-LOOK drops dramatically in relation to the other seek-reducing algorithms. This is best seen in the *Sci-TS* trace: C-LOOK **decreases** the average response time by better than 40% over LOOK, VSCAN(0.2), and SSTF when the cache is enabled, but it **increases** the response time by over 40% when the on-board memory is used as a simple buffer.

For some traces, such as *Report*, the performance improvement of C-LOOK over LOOK and SSTF is greater than would be expected from the improvement in cache hit rate. The ascending order of scheduling maintained by C-LOOK matches the prefetching nature of the disk. That is, the disk continues reading data beyond the end of a read request, often moving the read/write head forward one or more surfaces (or even to the next cylinder). So, requests which are logically forward of an immediately previous read will suffer shorter seek delays. This may account for some of the additional performance achieved by C-LOOK scheduling.

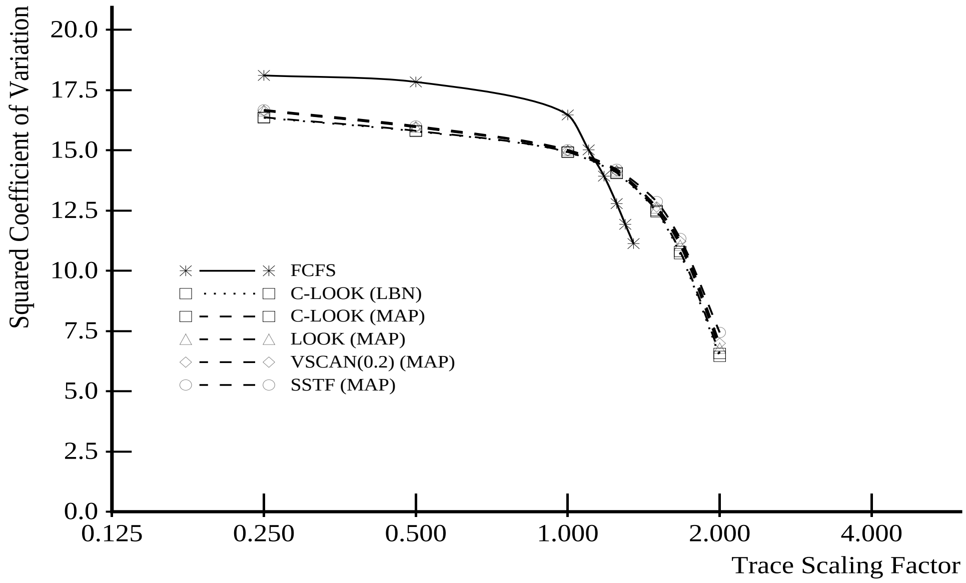
For all traces, C-LOOK provides starvation resistance equivalent or superior to the other seek-reducing algorithms (as evidenced by its squared coefficient of variation). This is not surprising, as C-LOOK was invented for this purpose. Predictably, LOOK has the next highest coefficient for most workloads, followed by VSCAN(0.2) and SSTF.

6.2 Scheduling with a Known Mapping

Figures 12-17 display the average response times and squared coefficients of variation for full-map scheduling of the six traces. FCFS and the LBN-based C-LOOK algorithm are also shown for comparision purposes. Performance

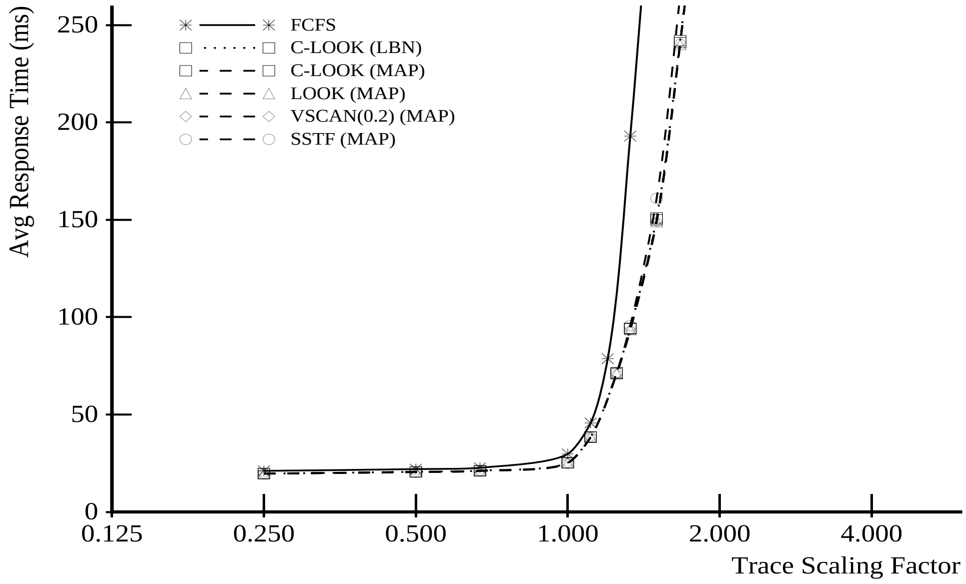


(a) Average Response Time

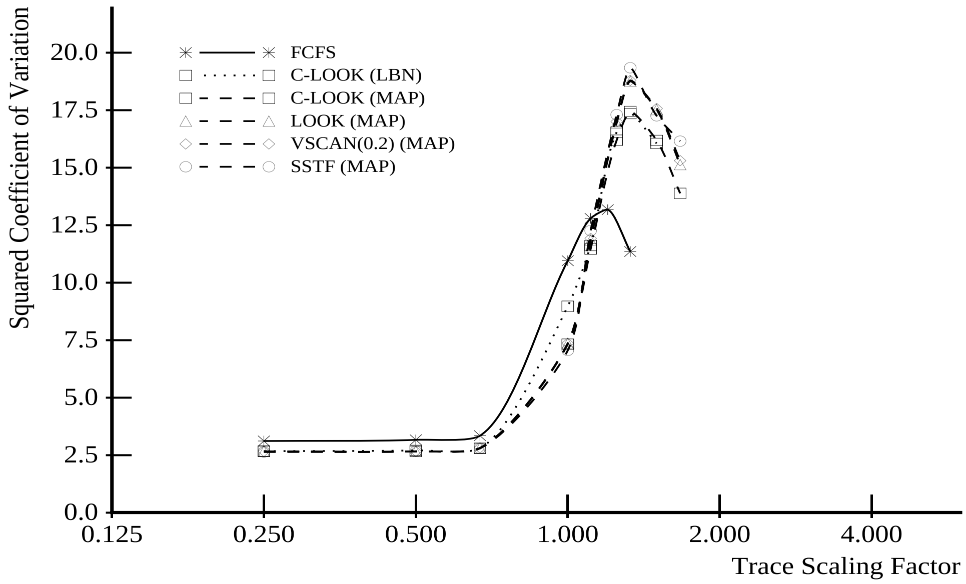


(b) Squared Coefficient of Variation

Figure 12: Full-Map and LBN-Based Algorithm Performance using the *Cello* Trace

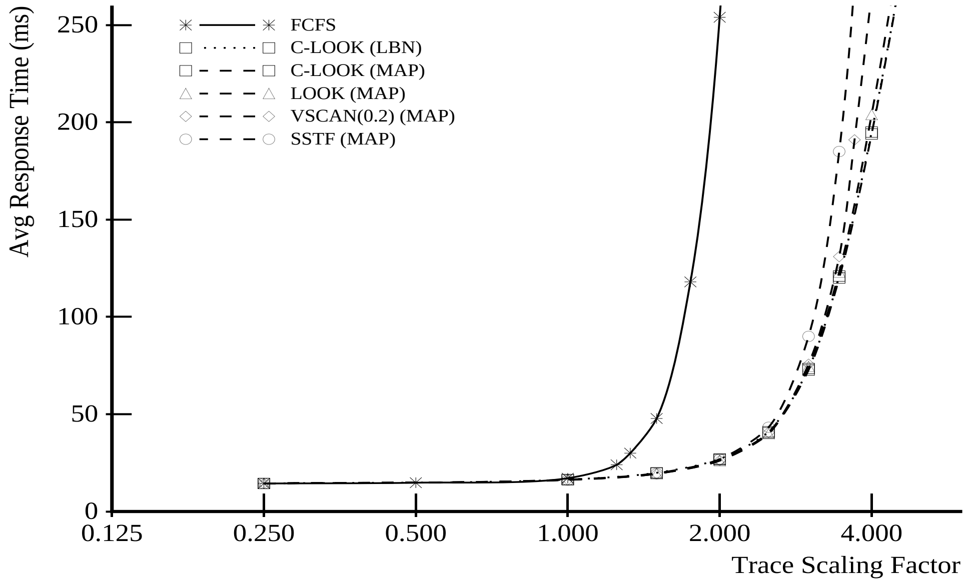


(a) Average Response Time

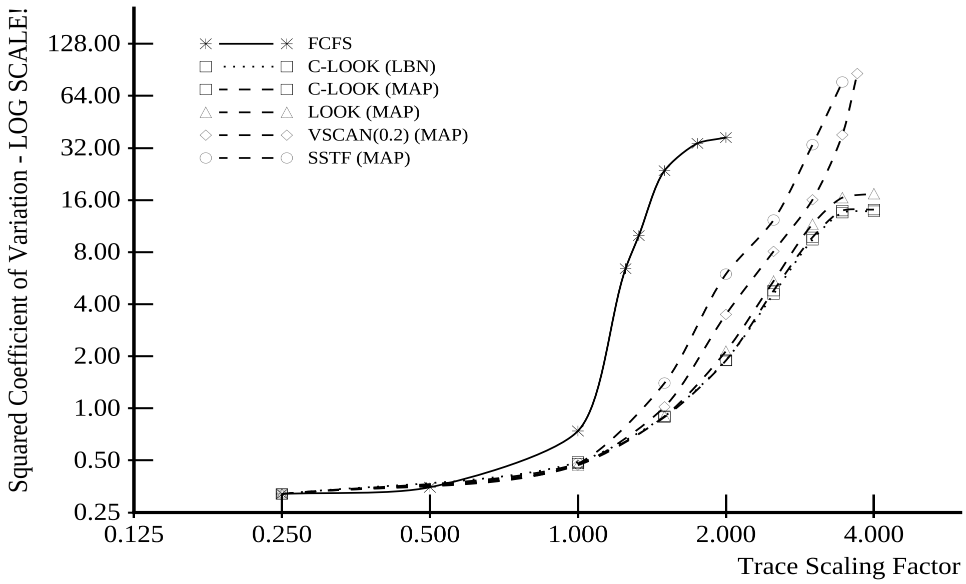


(b) Squared Coefficient of Variation

Figure 13: Full-Map and LBN-Based Algorithm Performance using the *Snake* Trace

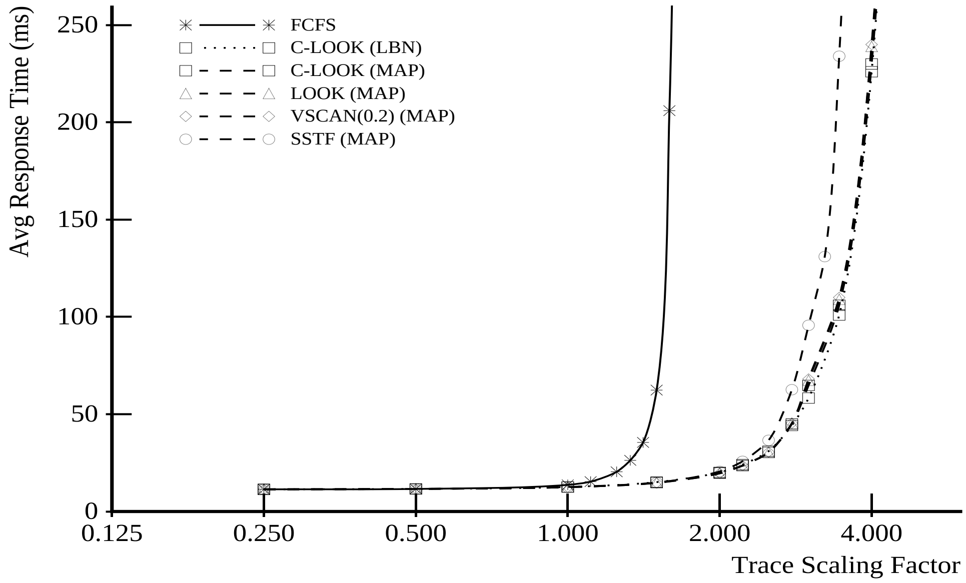


(a) Average Response Time

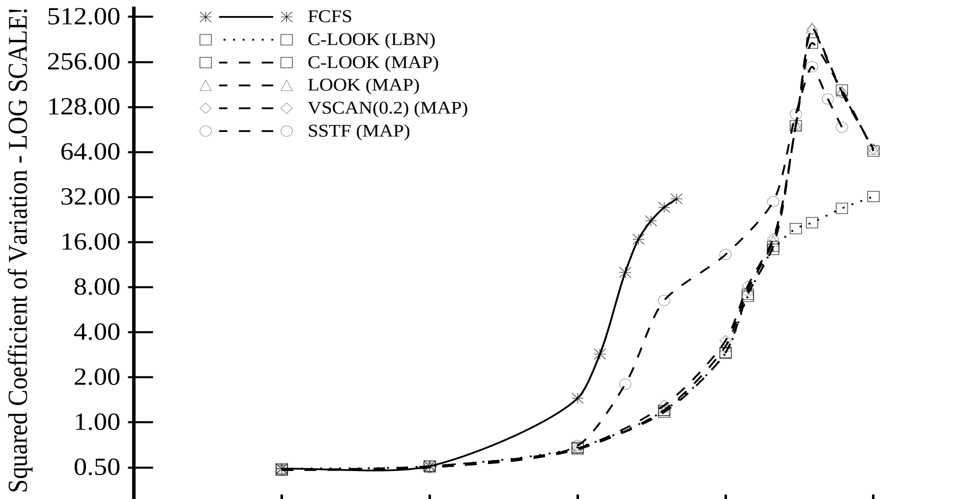


(b) Squared Coefficient of Variation

Figure 14: Full-Map and LBN-Based Algorithm Performance using the *Air-Rsv* Trace

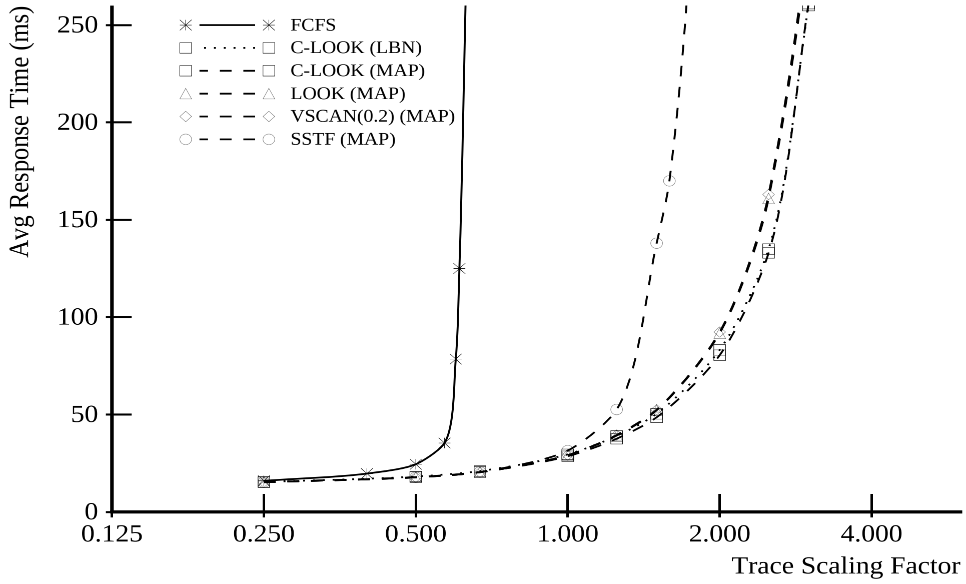


(a) Average Response Time

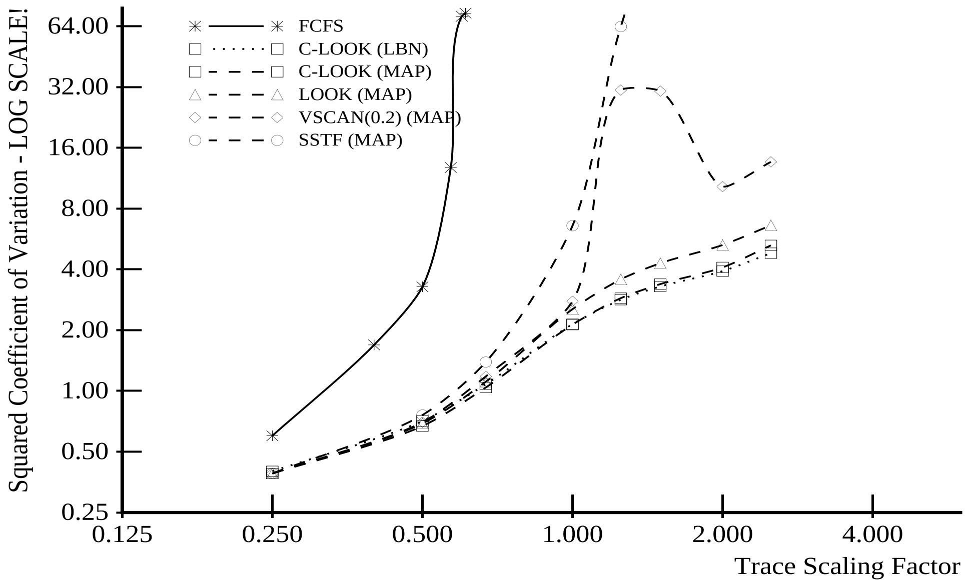


(b) Squared Coefficient of Variation

Figure 15: Full-Map and LBN-Based Algorithm Performance using the *Sci-TS* Trace

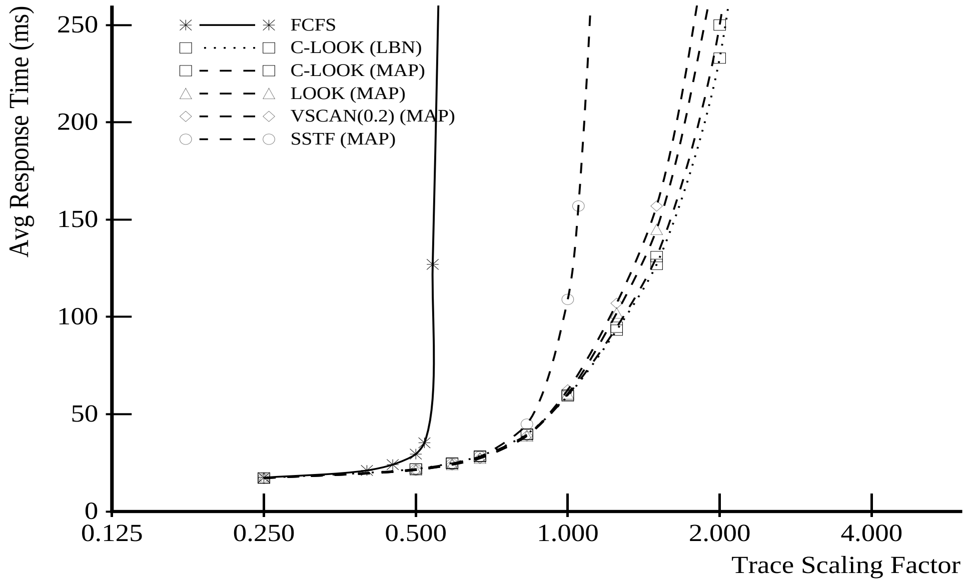


(a) Average Response Time

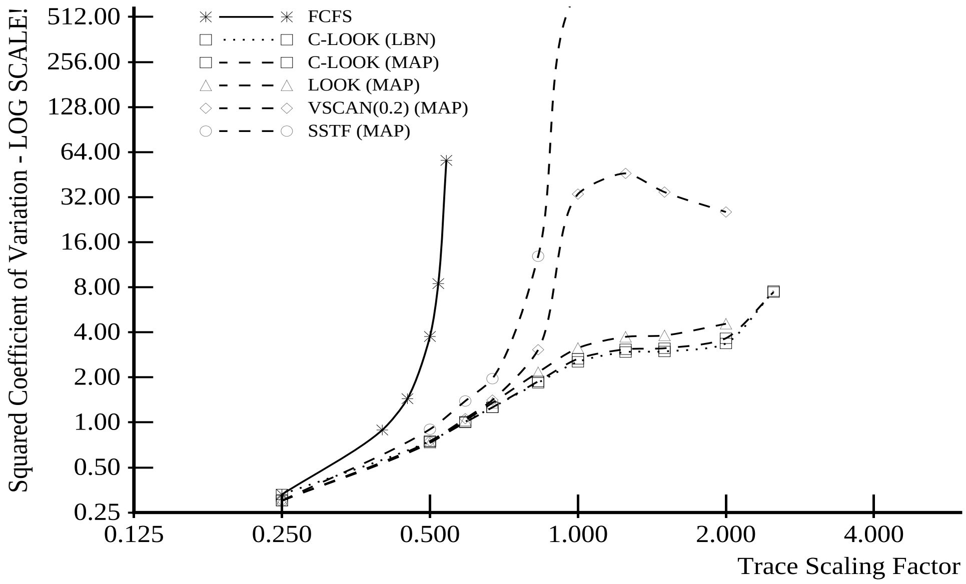


(b) Squared Coefficient of Variation

Figure 16: Full-Map and LBN-Based Algorithm Performance using the *Order* Trace



(a) Average Response Time



(b) Squared Coefficient of Variation

Figure 17: Full-Map and LBN-Based Algorithm Performance using the *Report* Trace

improvements are observed for LOOK, VSCAN(0.2), and SSTF under all workloads except *Cello*. This is not the result of reduced seek times via better mapping information, but rather is caused by the heuristic we use to schedule requests within a single cylinder. As described in section 5.2, the scheduler uses C-LOOK, LOOK, VSCAN(0.2), or SSTF when moving between cylinders, but uses C-LOOK "within" a cylinder. Therefore, the cylinder-based versions of LOOK, VSCAN(0.2) and SSTF use the prefetching cache much more effectively than their LBN-based counterparts. Further research into heuristics for scheduling within a cylinder may be necessary.

To verify the relative importance of the C-LOOK heuristic compared to the full LBN-to-PBN mapping, we modified the scheduling algorithms to use a fixed number of sectors per cylinder (1024) rather than the actual specifications (which are zone dependent). The resulting performance for all traces and algorithms is within 1% of the performance obtained using the actual mappings. We conclude, as was the case for random workloads, that the complexity of maintaining a full LBN-to-PBN mapping is not justified for seek-reducing algorithms.

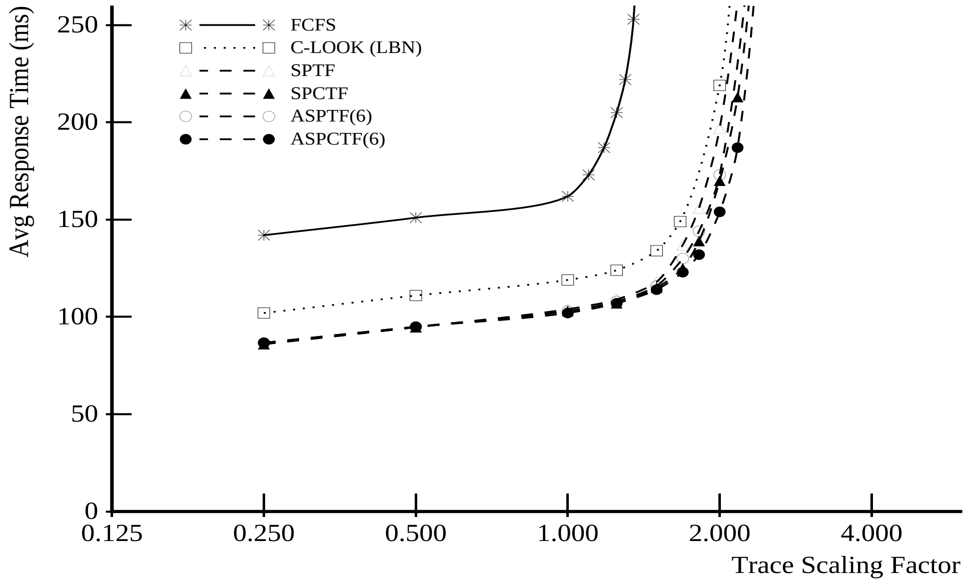
In general, the performance of full-map LOOK, VSCAN(0.2), and SSTF more closely match that of C-LOOK than was the case for the LBN-based algorithms. For the five workloads where C-LOOK provides the best average response times, this results in improved performance for LOOK, VSCAN(0.2), and SSTF. However, C-LOOK itself obtains only a marginal reduction in average response time when the LBN-to-PBN mapping is known.

SSTF improves less than LOOK or VSCAN(0.2) when the mapping and cylinder heuristic are incorporated. In the case of *Report*, the LBN-based SSTF scheduler is actually superior to the full-map version. This may be explained by examining the way in which our SSTF full-map implementation differs from C-LOOK, LOOK, and VSCAN(0.2). SSTF takes the "last known position" of the read/write head as being just beyond the **last** block of the immediately previous request. C-LOOK, LOOK, and VSCAN(0.2) use the **first** block of the previous request instead. This is required in order to keep multiple requests to the same location (or overlapping requests) from being unintentionally starved by the three scanning algorithms. When the SSTF algorithm implementation is modified to use the first block, it achieves performance improvement equivalent to LOOK and VSCAN(0.2). This indicates that repeat or overlapping requests exist in the experimental workloads. The fraction of such requests that are directly attributable to our methodology (scaling of traced interarrival times) has not yet been determined.

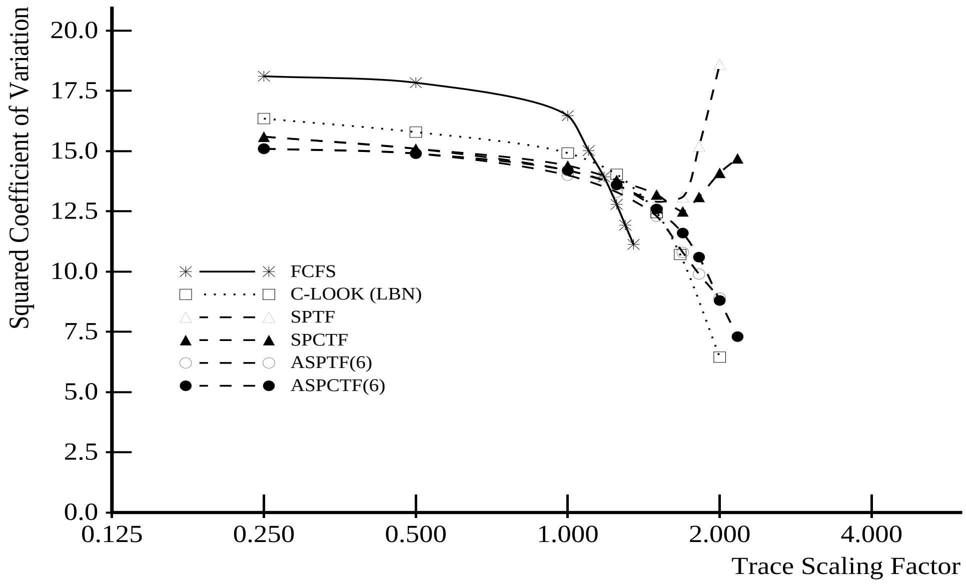
The relative starvation characteristics of the scheduling algorithms generally match the results found for the LBN-based schemes. For the *Sci-TS* trace, however, the full-map C-LOOK algorithm has only a marginal advantage in response time variance over the other algorithms. In fact, SSTF has a smaller coefficient of variance than C-LOOK for the heaviest workloads, but its response time variance is still greater (due to the difference in the algorithms' average response times). By satisfying all requests within a cylinder before moving to another cylinder, all algorithms become somewhat less resistant to starvation.

6.3 Scheduling with Full Knowledge

Figures 18-23 show results for experiments using SPTF-based algorithms, as described in section 5.3. Because these algorithms do not acknowledge and exploit the cache directly, we also present data for modified versions which track the contents of the on-board cache and estimate a positioning time of zero for any request that can be satisfied (at least partially) from the cache. The resulting algorithms are denoted as *Shortest Positioning (w/Cache) Time First* (SPCTF) and *Aged Shortest Positioning (w/Cache) Time First* (ASPCTF).

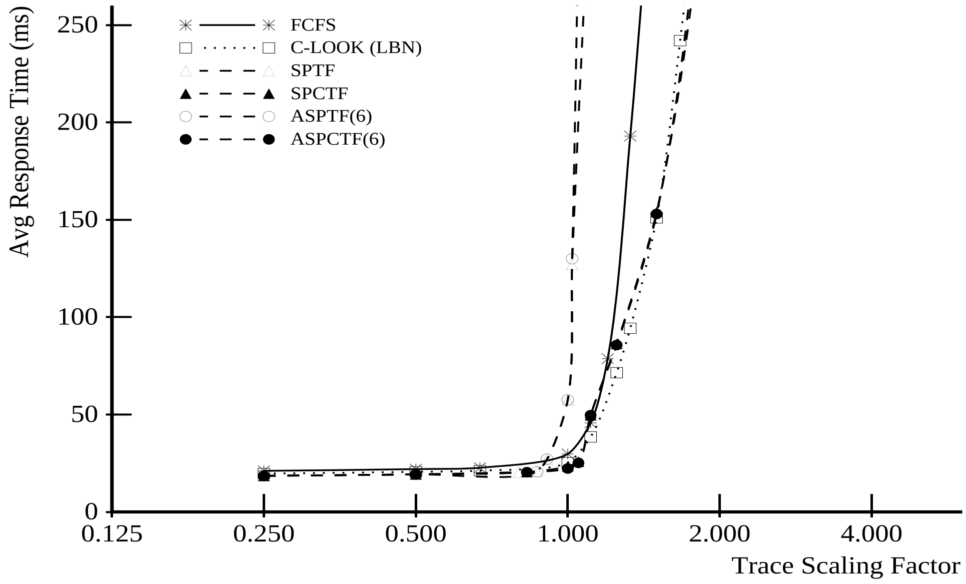


(a) Average Response Time

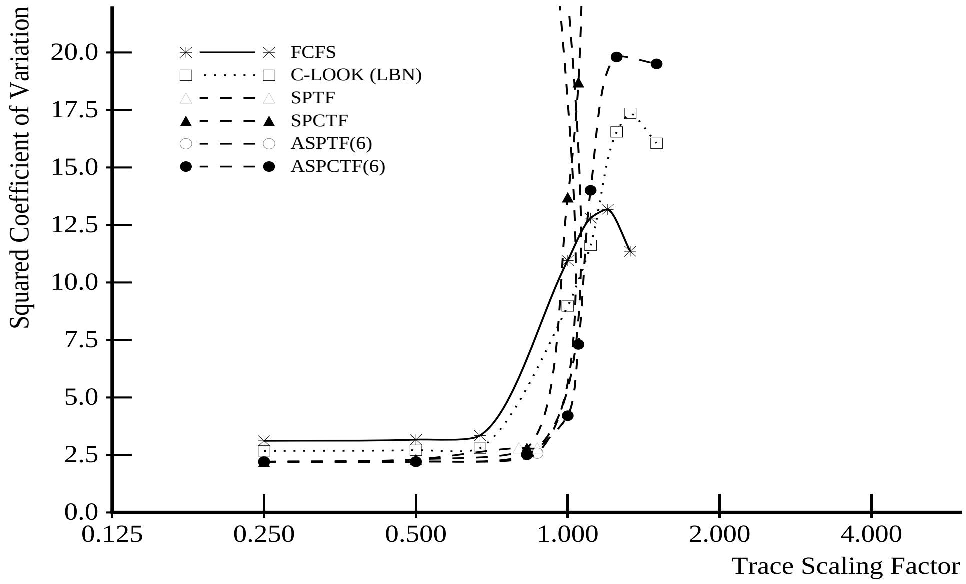


(b) Squared Coefficient of Variation

Figure 18: Full-Knowledge Algorithm Performance using the *Cello* Trace

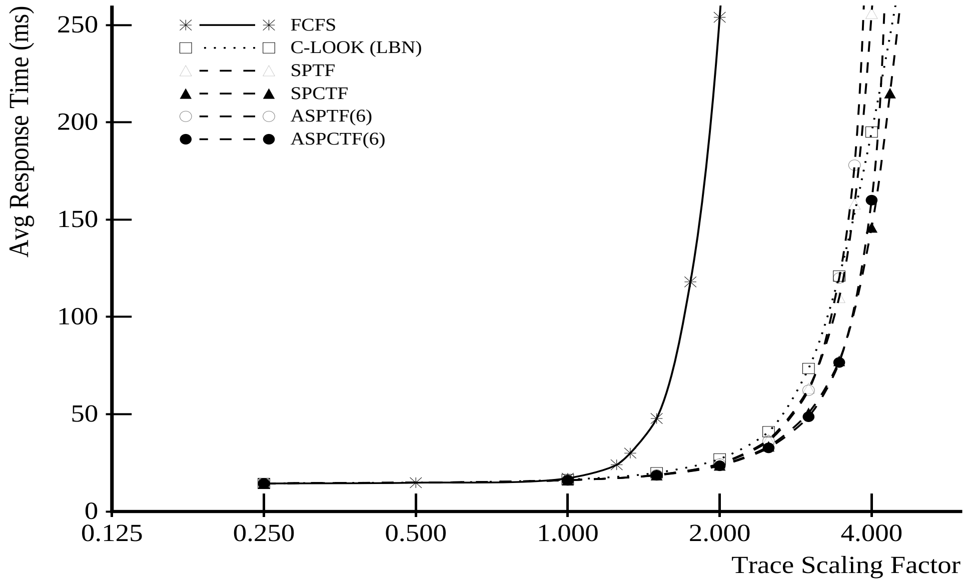


(a) Average Response Time

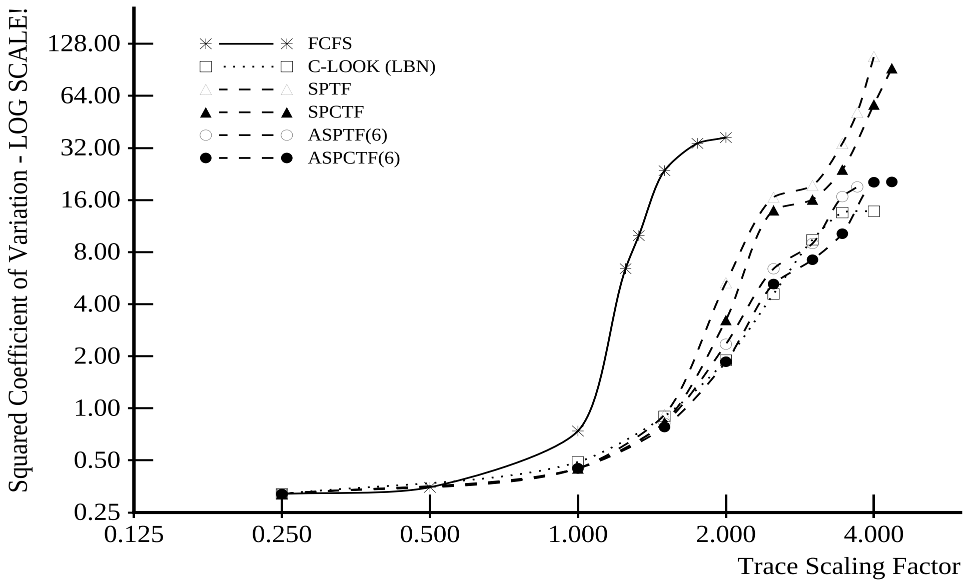


(b) Squared Coefficient of Variation

Figure 19: Full-Knowledge Algorithm Performance using the *Snake* Trace

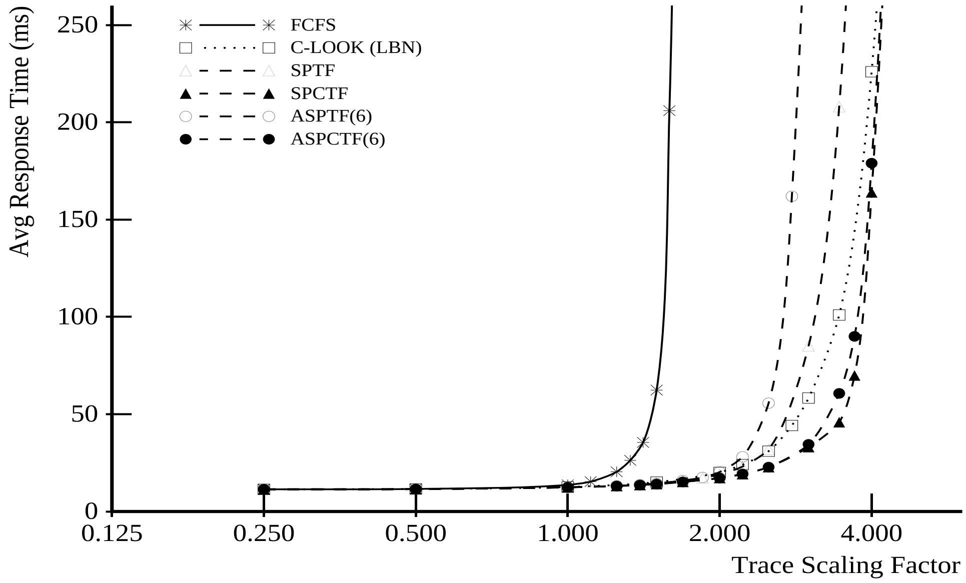


(a) Average Response Time

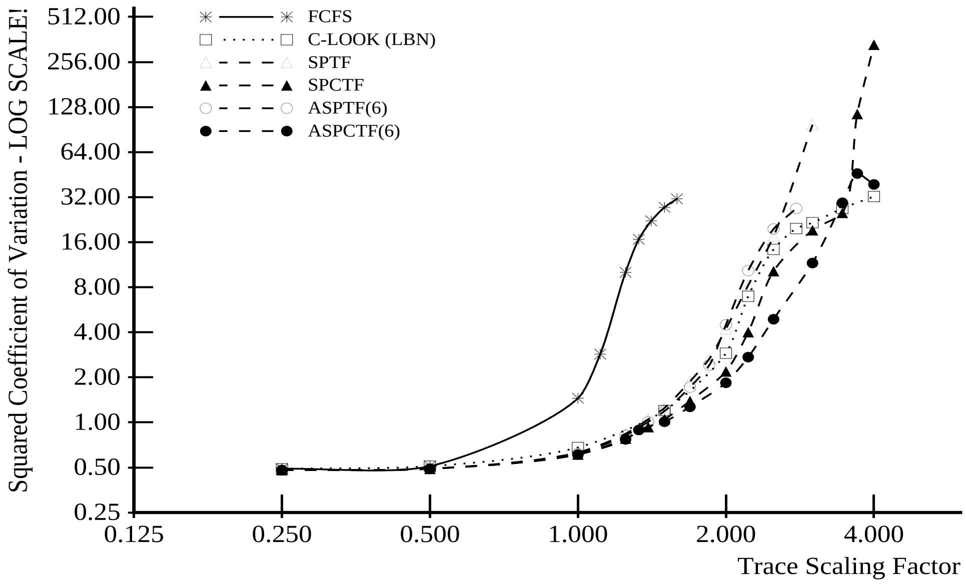


(b) Squared Coefficient of Variation

Figure 20: Full-Knowledge Algorithm Performance using the *Air-Rsv* Trace



(a) Average Response Time



(b) Squared Coefficient of Variation

Figure 21: Full-Knowledge Algorithm Performance using the *Sci-TS* Trace