Aggressive Centralized and Distributed Scheduling of Disk Requests

by

Bruce L. Worthington

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1995

Doctoral Committee:
Professor Yale N. Patt, Chair
Professor Edward S. Davidson
Professor Trevor N. Mudge
Professor Kensall D. Wise
Richie Lary, Storage Architect, Digital Equipment Corporation

ABSTRACT

Aggressive Centralized and Distributed Scheduling
of Disk Requests

by
Bruce L. Worthington

Chair: Yale N. Patt

Disk request scheduling is a critical element of high-performance computer system design. Previous disk scheduling research is insufficient because it uses simplistic or outdated disk models, unrepresentative workloads, and inadequate performance metrics. This dissertation explores the disk scheduler design space and provides guidelines for computer system engineers. Host-based and disk-based centralized scheduling are compared with distributed scheduling, a new paradigm with cost, complexity, and performance advantages. In distributed scheduling, individual disk schedulers located at multiple points along the I/O path cooperate to make scheduling decisions.

Scheduler implementations are evaluated using a detailed simulator containing validated models of state-of-the-art host systems, controllers, buses, and disk drives. The simulator is driven with extensive traces of real-world system and disk activity, and it reports system performance metrics (e.g., application run times) in addition to traditional disk subsystem metrics (e.g., mean request response times).

It is shown that for the best system performance, scheduling algorithms must incorporate system-level information (e.g., request priorities). Scheduling algorithms that focus on reducing seek delays do not need extensive disk-specific information. Finally, algorithms must exploit a disk's on-board cache to achieve the lowest disk service times.

Additional guidelines are provided for dealing with disk drive command queues, sequential disk request optimizations, and different on-board cache configurations.

To my father, Donald, for teaching me to always finish what I start;
to my mother, Margie, for her example of a life filled with compassion;
to my sister, Elizabeth, for being my truest friend;
to my aunts, Marilyn and Beverly, for their continual guidance and enthusiasm;
and to my wife, Janet, for holding my hand until the end of time.

ACKNOWLEDGEMENTS

My friends and labmates at the University of Michigan have been the single best aspect of my graduate school life. I am deeply thankful to my advisor, Dr. Yale Patt, for making me a part of his tightly-knit research group. I have thoroughly enjoyed my personal and professional association with the members of the group, including Mike Butler, Po-Yung Chang, Chris Eberly, Carlos Fuentes, Greg Ganger, Eric Hao, Robert Hou, Lea-Hwang Lee, Dennis Marsa, Sanjay Patel, Eric Sprangle, Jared Stark, and Tse-Yu Yeh. In particular, Greg Ganger has been instructor, pupil, and friend to me throughout our joint PhD quest.

Our administrative assistant, Michelle Chapman, has provided excellent support and never-ending patience during the last few years. Paula Denton, Denise Du Prie, and Jeanne Patterson formed a skilled second line of defense against general confusion and red-tape.

Edward Davidson, Richie Lary, Trevor Mudge, and Kensall Wise have my sincere gratitude for serving on my doctoral committee. Their comments and suggestions undoubtedly improved the quality of my dissertation.

I would like to thank John Fordemwalt, Richard Golding, Richie Lary, Mike Leonard, Chris Ruemmler, Carl Staelin, Tim Sullivan, Doug Voight, and John Wilkes for their professional guidance and insights. I thank Jim Pike of AT&T/GIS for providing the initial funding for my research at the University of Michigan. I also thank Hewlett-Packard, AT&T/GIS, Scientific and Engineering Software, and Digital Equipment Corporation for providing additional equipment and financial support.

Finally, I would like to thank the various animated and comic characters that have kept me sane: The Tick, Arthur, and all the other superheroes in The City; Calvin and Hobbes; Dilbert, Dogbert, Catbert, Ratbert, and Wally; The Uncanny X-Men; Brue Spluce (my X-Pilot alter-ego); Opus, Bill the Cat, and the rest of the Bloom County/Outland gang; Bugs Bunny; Daffy Duck; and Two Stupid Dogs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure

CHAPTER 1

Introduction

1.1 Overview

Disk subsystems must be effectively utilized 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 dynamically reorders pending requests to decrease service times. A significant portion of disk service time consists of mechanical positioning delays, which are highly dependent on the relative positions of the requested blocks and the current read/write head position. By taking into account the various delays associated with disk accesses, a scheduler can produce a request sequence that minimizes the mean positioning time while providing equitable response times for individual requests. In many cases, better overall system performance can be achieved by a disk scheduler that prioritizes requests based on their "importance" to the system as a whole. A scheduler with access to application-level or system-level information can generate a request schedule that produces more nearly optimal system performance.

Traditional disk schedulers typically utilize simple algorithms that require little knowledge of system goals or the state of the disk subsystem. The increasing amount of computation and memory resources available for scheduling activities makes more sophisticated disk scheduling algorithms feasible. As the demands on I/O subsystems increase, future high-performance systems will require more efficient and effective disk access. Aggressive disk request scheduling will play a major role in achieving this goal.

Disk schedulers can be found in application programs (e.g., database storage managers), O/S device drivers, intermediate I/O controllers (e.g., disk array controllers), and on-board disk drive controllers. In most systems, however, a disk request encounters only one "active" scheduler in the path from the application program to the disk drive media. One approach to improving performance is to increase the amount of information available to the active scheduler and augment existing scheduling algorithms to take this information into account.

Effective **centralized scheduling** requires that useful information be identified, extracted, and transmitted along the I/O path to a single active scheduler. Popular intermediate and peripheral bus protocols do not facilitate the transfer of such control information (e.g., resource status/utilization and request priority, dependency, and aging information).

Also, not all file system and device driver interfaces have the flexibility to communicate pertinent scheduling information.

Ideally, the location of a centralized scheduler should be chosen to minimize the control information traffic while assuring timely access to the most useful control information. In addition, a scheduler should have sufficient computation and memory resources to efficiently process incoming information, maintain queues of pending requests, and perform scheduling activities (i.e., searching and sorting the pending request queues). The cost and complexity disadvantages of centralized scheduling, however, motivated the investigation of an alternative scheduling paradigm.

Distributed scheduling partitions the necessary activities among separate entities along the I/O path. The scheduling entities must cooperate (i.e., exchange additional control information) in order to prevent undesirable interactions. Achieving such coordination may require changes to existing O/S interfaces and bus protocols. Even with appropriate hardware and software modifications, distributed scheduling may not attain performance equivalent to that of "omniscient" centralized scheduling (i.e., scheduling with complete knowledge of all useful information coupled with sufficient resources to effectively utilize the information). For realistic implementations, however, distributed scheduling has cost, complexity, and performance advantages.

Over the past 25 years, a variety of scheduling algorithms have been proposed and studied in both academia and industry. Previous scheduling algorithm studies typically used simplistic disk models lacking several important features of modern disk drives (e.g., on-board caches). The benchmark workloads were usually synthetic, using simple probability distributions (e.g., uniform, unimodal, or bimodal) for request starting locations. Host feedback effects were ignored (e.g., open subsystem models with exponentially distributed request interarrival times) or oversimplified (e.g., closed subsystem models maintaining a constant number of outstanding requests). The conclusions therefore should be considered in an appropriate context. Most apply only to a small set of workloads, which are typically unrepresentative of real-world systems, serviced by outdated disk drive components. In contrast, the studies discussed in this dissertation use a simulator containing a very detailed, well-validated disk model. The simulator uses traces taken from systems at several industrial and research installations. Some of the experiments use an open subsystem model, while others use a realistic host model developed at the University of Michigan [Gang93]. The host model provides system performance metrics to measure the user-level impact of scheduler design choices.

1.2 Thesis Statement

Previous studies of disk scheduling algorithms are inadequate for computer system engineers concerned with optimizing disk subsystem performance. This dissertation provides guidelines for designing and implementing aggressive disk request schedulers in high-performance systems equipped with state-of-the-art storage components.

1.3 Accomplishments of this Dissertation

- It is shown that distributed scheduler implementations provide performance equal to or superior to that of equivalent centralized scheduler implementations for most workloads studied. Since distributed scheduling also has cost and complexity advantages, this new scheduling paradigm is recommended for future high-performance systems.

- It is shown that algorithms incorporating system-level information (e.g., disk request priorities) provide the highest levels of overall system performance for the real-world traces studied. This performance advantage is not necessarily reflected in (and sometimes runs counter to) standard disk subsystem performance metrics such as request response time mean and variance.

- It is shown that the quantity and detail of hardware-specific information required by a disk request scheduler is a function of the algorithm complexity. Simple scheduling algorithms gain little performance benefit from extensive disk drive specifications. Complex scheduling algorithms, on the other hand, require comprehensive disk drive configuration and state information.

- It is shown that disk request schedulers that effectively exploit a disk drive's on-board data cache provide superior performance. Mean response times for medium-to-heavy workloads decrease significantly when a disk request scheduler reorders or combines disk requests to improve on-board cache utilization.

- A wide variety of scheduling algorithms and implementations are compared and contrasted using detailed simulation driven by traces of real-world disk and system activity.

1.4 Organization of this Dissertation

This dissertation is organized in eight chapters. Chapter 2 describes a variety of information potentially useful to the disk scheduler. Chapter 3 discusses previous disk scheduling studies. Chapter 4 describes the methodology employed throughout this work, including a discussion of the traced workloads and the simulator. Chapters 5 and 6 compare various algorithms for centralized scheduling at the host and disk, respectively. Chapter 7 compares the same algorithms using distributed scheduling. Chapter 8 provides some concluding remarks and discusses future directions. The appendix lists parameter values for the disks modeled in this work.

CHAPTER 2

Useful Scheduling Information

A disk request scheduler reorders queues of pending requests to optimize some set of criteria (e.g., performance and reliability metrics). It must therefore have access to information about individual requests and the state of the components along the I/O path. This chapter identifies various types of information that are useful to a disk request scheduler.

The set of useful information is divided into two classes. Information from "below" the scheduler relates to the I/O path hardware, typically concerning the configuration, operation, and current state of the individual controllers, buses, and storage devices (e.g., disks). Information from "above" the scheduler specifies how requests relate to each other and to system goals (e.g., application runtimes and reliability guarantees).

2.1 Information From "Below" the Scheduler

A scheduler can use hardware-specific information to reduce disk request service times. In particular, aggressive disk schedulers can exploit information about disk drive configuration and state to reduce mechanical delays and make better use of on-board caches.

2.1.1 The Modern Disk Drive

A disk drive consists of a set of **platters** rotating on a common axis, or **spindle** (see figure 2.1). Both **surfaces** of each platter are coated with magnetic media. 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 outermost cylinder contains the outermost track of each surface. Each surface has an associated **read/write head**. In most drives only one read/write head is active at any given time. The heads are mounted on disk **arms** that are ganged together on a common **actuator**. Each cylinder of data is accessed from a specific actuator position.[1]

[1] With increasing track densities, it has become necessary to (slightly) reposition the actuator when switching between tracks on a cylinder. High-capacity drives use **servo bursts** embedded between sectors of data to align the read/write head "over" a track.

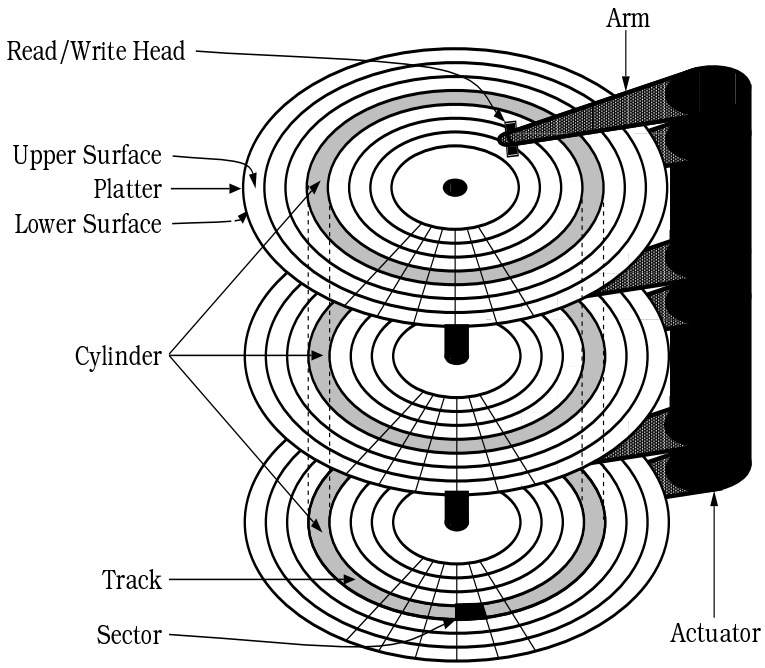


Figure 2.1: Disk Drive Terminology

Mechanical delays often dominate disk request service times. To access a requested media location, a disk actuator must first **seek** to the target cylinder and activate the appropriate read/write head. Then, a request incurs **rotational latency** while waiting for the target sectors to rotate into position. After these delays, the magnetic media is accessed as the sectors rotate "under" the read/write head. Further mechanical delays result if multiple tracks or cylinders must be accessed.

In addition to the mechanical characteristics, a number of modern disk features can directly affect scheduling algorithm performance. Some are the result of efforts to make disks self-contained or self-managed. Clever methods of increasing drive capacity (e.g., zones) and effective lifetime (e.g., automatic defect management) also have a major impact. The remainder of this section provides additional details on mechanical positioning delays, the on-board disk controller, the mapping of logical data blocks to physical media locations, and methods for obtaining disk drive state and configuration information.

Mechanical Positioning

Positioning a read/write head to access specific sectors of data involves several individual delays. To correctly predict the positioning times of pending requests, a scheduler must have accurate approximations of the seek curve, head switch time, write settling time, rotation speed, and current position of the disk actuator.

A **seek curve** maps seek distances to actuator positioning times. The seek time between two cylinders is a complex function of the positions of the cylinders (i.e., their distance from the spindle), the direction of the seek (inward or outward), various environmental factors (e.g., vibration and thermal variation), the type of servo information (dedicated, embedded,

or hybrid), the mass and flexibility of the head/arm assembly, the current available to the actuator, etc. A single curve can therefore only reflect mean or upper-bound estimations for seek times. Typical seek curves consist of an irregular initial region for very short seeks, a middle region approximated by a square root function, and a large linear region for long seeks (see figure 2.2 and the appendix).

The **head switch** time (i.e., the time necessary to electronically switch from one read/write head to another and realign "over" the new track) depends primarily on the type of servo information utilized by the disk drive (e.g., dedicated, embedded, or hybrid) and various environmental factors (e.g., vibration and thermal variation). Seek and head switch activity occur simultaneously, with seek time dominating.[2] A write request that requires a seek or head switch may need additional time to more closely align the read/write head (the **write settling** time).[3]

Disk drive specifications allow rotation speeds to vary slightly from disk to disk. The rotation speed of a disk may even vary from moment to moment. Modern drives can pass and receive rotational synchronization signals, and an external scheduler can track the speed and rotational position of a disk by monitoring these signals. If the necessary monitoring hardware is unavailable, an aggressive scheduler can estimate the current rotational position of a disk by tracking request completion times [Wort95].

Disk Controller

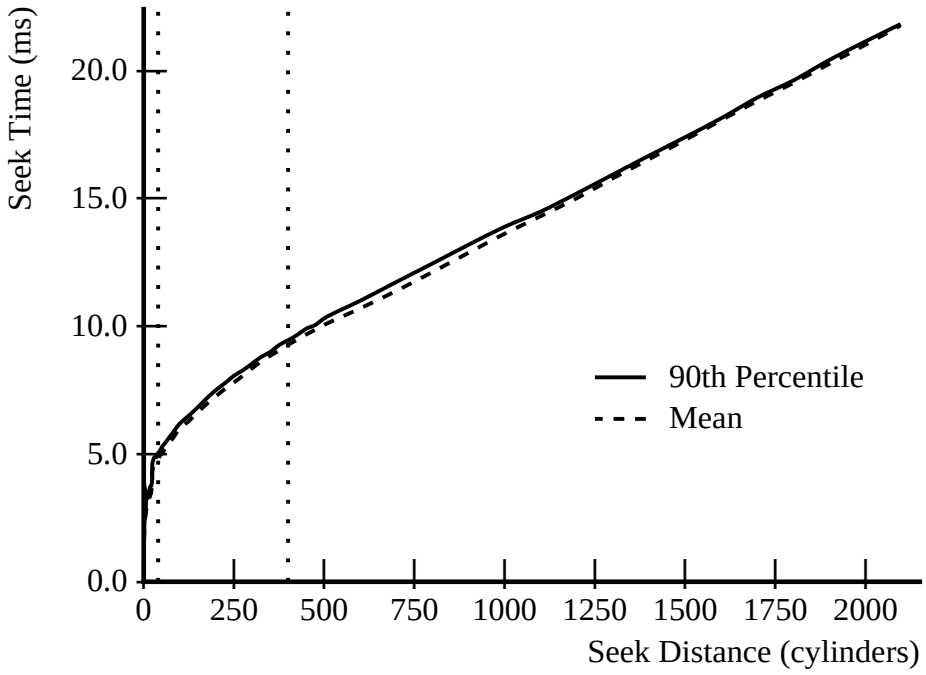
The resources available to on-board disk controllers increase with each new generation of drives. Current disk drive components include powerful embedded microprocessors and large quantities of memory. An on-board disk controller handles the peripheral bus interface, interprets and services disk requests, manages the on-board data cache, and (if appropriately configured) maintains and schedules queues of pending requests. Disk controller firmware typically contains a small operating system coupled with a number of processes to perform the various disk controller functions.

Host Interface

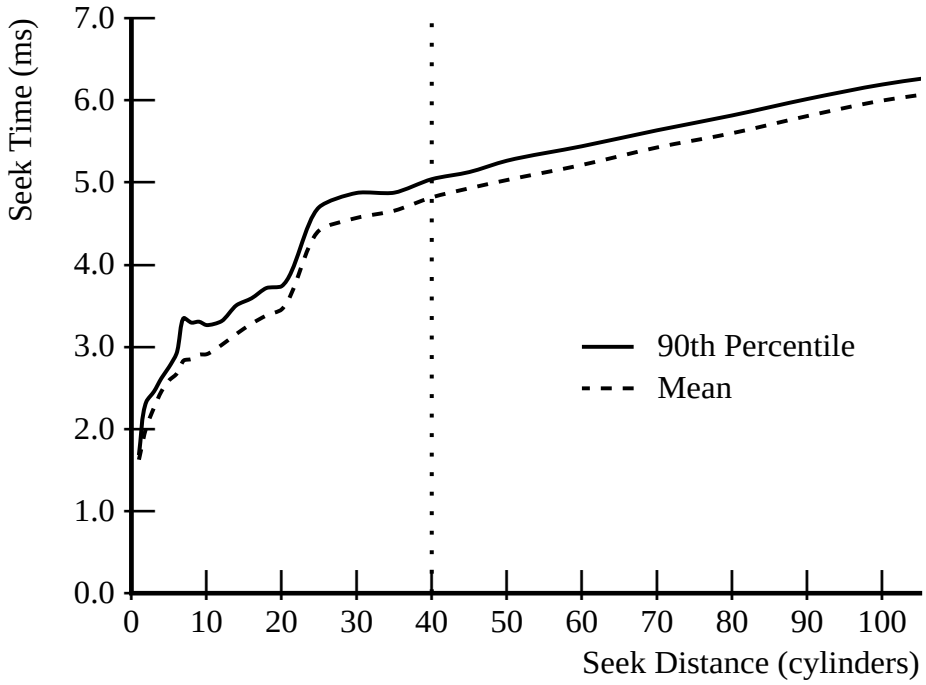
Most disks possess sufficient on-board logic and processing power to present a relatively clean interface to a host system. High-level protocols, such as the Small Computer System Interface (SCSI) and the Intelligent Peripheral Interface (IPI), are supported by a wide variety of disk drive, disk array, I/O adapter, and host system manufacturers. A host or intermediate controller issues a request to a disk drive in terms of a starting **logical block number** (LBN) and a total request size. The details of the subsequent media access are hidden from the host. This approach offloads the management overhead associated with the actual data storage from the host or controller to the disk drive electronics. As a result, scheduling entities external to a modern drive typically have little or no knowledge of the

[2]Improvements in magnetic media and servo technology continue to increase track densities. For future generations of disks, repositioning between adjacent tracks on a surface (i.e., a 1-cylinder seek) may actually require less time than repositioning between tracks on a cylinder.

[3]Data can be successfully read from the media even when a read/write head is slightly misaligned, but data must be written as close to the middle of a track as possible in order to prevent corruption of adjacent tracks of data.



(a) Full seek curve



(b) Expanded view of short seeks

Figure 2.2: Measured Seek Curves for the Seagate ST41601N Disk Drive

physical data layout, the status of the on-board data cache, and the various overhead delays associated with each request.

Control and Communication Overheads

A disk request incurs several control and communication delays during its lifetime. Aggressive scheduling algorithms must plan for such delays to be effective. For example, if a disk rotating at 5400 RPM (11.1 ms per rotation) requires 1 ms of command initiation overhead, a media access that would begin in 0.5 ms without any overhead (given the current rotational position) will "just miss" the target sectors and wait for a total of 11.6 ms. If the initiation overhead were taken into account, a request with a shorter delay might be chosen.

Disk request processing overheads depend on the request type (read or write), the request size, the state of the on-board cache (hit or miss), and the immediately previous request's characteristics. Protocol and bus transfer delays depend on the individual components along the I/O path (e.g., host processors, intermediate controllers, I/O adapters, buses, and on-board disk controllers). The appendix lists measured overhead values for the HP C2247 disk drive.

On-Board Data Cache

The use of memory within embedded disk drive control logic has progressed from small, speed-matching buffers to dynamically-managed caches containing several megabytes of storage. On-board disk logic can automatically prefetch data to more quickly satisfy subsequent read requests. For example, a disk might take 20 ms to service a read request that requires media access, whereas a subsequent sequential read request for prefetched data from the on-board cache might take only 1 ms.

The existence of an on-board cache affects scheduling activities in two ways. First, a scheduler may not be able to determine the actual position of the active read/write head after a read request; automatic prefetch activity may result in switching heads or seeking to the next cylinder. Second, cached data blocks can be accessed far more quickly than blocks obtained from the disk media. Schedulers may give priority to read requests that will hit in the cache in an attempt to reduce mean response times.

Some disks support the ability to signal write request completion as soon as all of the data blocks reach the on-board cache. The on-board controller moves the data from the cache to the media at some later point. Since current disk implementations use volatile memory components for the cache, the contents of the cache are lost if the power fails. **Fast writes** may therefore reduce the reliability of disk storage unless additional measures are taken to protect cached data (e.g., non-volatile memory components or uninterruptable power supplies).

Disks with **read-on-arrival** and/or **write-on-arrival** capability reduce request service times by transferring data to or from the media "out-of-order." That is, sectors are accessed in the order that the read/write head encounters them rather than in strictly ascending LBN order. For example, if an entire track of data is to be read, a disk with read-on-arrival begins transferring data from the media almost immediately after any seek or head switch delays are complete. This essentially eliminates rotational latency. The performance advantage is less for requests larger or smaller than a track.

Most on-board data caches contain multiple cache lines, or **segments**, to better service multiple streams of sequential requests. Some disks fine-tune cache performance by analyzing disk request patterns and dynamically modifying the number of segments (and the segment size).

Scheduling issues related to fast writes, read/write-on-arrival, and dynamic cache segmentation are beyond the scope of this dissertation.

Command Queueing

Modern disk drives can handle multiple outstanding requests for service. **Command queueing** overlaps processing overhead for incoming requests with media access for prior requests. Some command queueing implementations allow the disk controller to take an active role in scheduling pending requests (e.g., to reduce mechanical latencies). Disk controllers can also increase the concurrency between requests receiving service. Section 4.1.2 discusses several specific optimizations.

If a disk services requests in arrival order, external schedulers need only know the depth of the on-board command queue. If a disk reorders pending requests, external schedulers should coordinate their activity with the on-board disk scheduler (i.e., use distributed scheduling).

Data Layout

Many existing schedulers use simple LBN-based approximations of seek-reducing algorithms. LBN-based scheduling relies on highly sequential logical block number to physical block number (LBN-to-PBN) mappings. More aggressive algorithms require accurate knowledge of the data layout. Zoned recording, track/cylinder skew, and defect management schemes complicate LBN-to-PBN mappings.

Zoned Recording

To exploit the larger track circumference of cylinders farther from the spindle, a modern drive usually partitions its 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 might have twice as many sectors per track as the innermost zone. This increases both the capacity per unit volume and the mean media transfer rate of the disk. Within each zone, a disk may reserve a subset of cylinders (or tracks or sectors) for defect management (see below).

Track and Cylinder Skew

Since switching between tracks and seeking between adjacent cylinders requires non-zero time, the first logical block of each track or cylinder is offset (or **skewed**) from the first logical block of the previous track or cylinder. This prevents a request that crosses a track or cylinder boundary from "just missing" the next logical block and waiting almost a full rotation for it to come around again. Track and cylinder skews are generally specified as an integral number of sectors and therefore differ from zone to zone.

Defect Management

Disk drives can survive the loss of hundreds of sectors or tracks to media corruption. In fact, a new drive usually contains a list of defects identified at the factory. A disk updates this list with defects **grown** during its lifetime and removes the corrupt media locations from its LBN-to-PBN mapping.

Sector or track **slipping** occurs during disk format. In the case of sector slipping, the format process skips each defective sector and adjusts the LBN-to-PBN mapping so that the sectors on either "side" hold logically sequential data. Track slipping follows the same routine, except that the format process skips an entire track if it contains 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 rotational position of each block. The format process reserves extra space (**spare regions**) at the end of certain tracks, cylinders, or zones to contain the spillover caused by slipped sectors or tracks (to minimize the impact of defects on the static LBN-to-PBN mapping). In general, large spare regions affect seek behavior and small spare regions (distributed across the disk) interrupt sequential data transfers.

Reallocation occurs when a disk discovers defective sectors (or tracks) during normal disk use. Modern disks automatically remap the affected LBNs to spare regions and redirect subsequent accesses to the appropriate spare sectors. The disks used in this work contain very few reallocated defects, producing no discernible performance effects.

Extraction of Disk Drive Parameters

The previous sections describe a number of disk drive parameters useful to a scheduler. External schedulers must somehow obtain such information from a disk in order to exploit it. In particular, algorithms that reduce combined seek and rotational latencies require detailed information on command processing overheads, mechanical latencies, and data layout. To enable the use of such algorithms, a suite of quick and effective programs have been developed to extract the necessary information from modern SCSI disk drives. [Wort95] contains an overview of the extraction process.

2.1.2 Other I/O Path State

Although the most important hardware-specific data relates to the disk drives, other I/O path components can also supply useful scheduling information. Bus adapters can provide the current status of intermediate and peripheral buses. Intermediate controllers with sufficient computation and memory resources can provide both caching and scheduling functionality. In particular, disk array controllers can contribute significantly to scheduling performance.

Disk arrays are becoming commodity items for systems of all sizes. A variety of data placement and redundancy schemes are available to improve performance and/or data reliability [Gang94, Hou93, Holl92, Gray90, Katz89, Patt88, Kim86]. An intelligent disk array controller can reorder pending requests based on the state of any caches, disks, and buses under its control. For example, if an array employs a redundancy scheme that pro-

vides multiple paths for accessing data (e.g., disk mirroring), the array controller could dynamically route requests to the components that can most quickly service the requests [Poly93, Sugg93, Ng91, Chen90, Lee90, Bitt88]. Alternately, the array controller could schedule requests to the components with the shortest pending queues.

In this dissertation, all information from "below" the scheduler relates to disk drive configuration and state. Scheduling optimizations using bus and intermediate controller information are left as future work.

2.2 Information From "Above" the Scheduler

It is easy to lose sight of the "big picture" when developing scheduling algorithms that use large quantities of hardware-specific information. The end goal of disk scheduling is not to reduce the mean disk request response time, but to reduce (or eliminate) the performance impact of "slow" disk activity on application processes. Information about inter-request relationships and the impact of individual requests on system goals allows a scheduler to optimize for overall system performance rather than disk subsystem performance. System-level information originates from the operating system and the individual processes issuing disk requests.

2.2.1 Application and File System Information

A scheduler can prioritize pending requests based on the system-level information available for each request. For example, read requests for which application processes are waiting might receive high priority. Background write requests periodically generated by disk cache flush daemons[4] might receive low priority.

If an application (e.g., a database storage manager) schedules its own requests, it can use application-specific information directly.[5] If scheduling entities outside of the application (e.g., the device driver) are to exploit such information, an appropriate interface must be provided. For requests issued through a file system, similar interfaces and information-sharing techniques are necessary to allow application and file system information to reach schedulers farther down the I/O path.

File system metadata updates (e.g., requests generated by file creation and deletion) and ordered updates generated by applications (e.g., OLTP) often impose certain dependencies between requests. For example, a set of write requests may need to complete in a certain sequence to maintain data reliability or security guarantees. Schedulers can use dependency information not only to uphold such guarantees, but also to prioritize requests. For example, requests at the head of a long dependency chain might receive higher priority.

[4] A **daemon** is a special-purpose host process that wakes up periodically or on request. For example, daemons might be responsible for spooling printer requests, sending packets out over a network, and flushing disk block caches.

[5] The concept of **virtual devices** makes application-level request scheduling less prevalent in modern systems. That is, current operating systems present applications with an abstract view of secondary storage, hiding the mapping between virtual and physical devices.

Host system reliability guarantees can also constrain scheduling activities. For example, many UNIX[6] file systems guarantee that data updates written to the main memory disk cache will be issued to stable storage within a given period of time (e.g., 30 seconds). A background daemon periodically sweeps through the disk cache and writes dirty blocks to disk. Alternately, the file system could generate write requests with "deadlines," and the scheduler could guarantee that each request will be initiated before the specified deadline. Both scenarios require appropriate interfaces and information-sharing for effective disk request scheduling.

2.2.2 Resource Utilization

Request priorities should depend (to some degree) on the utilization of finite system resources, such as main memory disk cache blocks. For example, write requests generated because of disk cache pressure (i.e., few available clean cache blocks) should be expedited. Otherwise, the disk block cache may become a performance bottleneck [Benn94]. This example suggests an additional scheduling optimization. At the moment when cache pressure raises the priority of subsequent write requests, there may already be writes waiting for service. In order to quickly make space in the cache, the priorities of the outstanding write requests should also be raised (retroactively). This can be accomplished by using indirect or class-based priorities. That is, the control block associated with each request can contain a pointer to an entry in a table of request priorities. By changing a value in the priority table, an entire class of requests can be re-prioritized (including both existing and subsequent requests).

[6]UNIX is a registered trademark used under license from the X/Open Corporation.

CHAPTER 3

Related Work

3.1 Scheduler Implementations

A modern computer system contains a number of components that can potentially participate in scheduling activities. Device drivers embedded in the O/S traditionally contain scheduling algorithms that use only logical block numbers and/or request types (i.e., read or write). Schedulers found in intermediate controllers (e.g., on I/O cards or within storage enclosures) may utilize additional hardware-specific information (e.g., bus, controller, or disk drive state). Lastly, modern disk drives with active command queueing can use full knowledge of disk configuration, operation, and state.

Distributed schedulers make use of more than one entity along the I/O path to reorder disk requests. Traditional systems have little or no support for high-performance distributed scheduling. Simple high-level schedulers could act as filters that prioritize requests or enforce request dependencies by holding back low-priority or dependent disk requests. This degenerate form of distributed scheduling leaves much to be desired, although it is implementable on current systems without modification to existing interfaces and protocols. More aggressive distributed schedulers require active coordination between scheduling components.

3.2 Scheduling Algorithms

Numerous studies have shown that **First Come First Served** (FCFS) disk scheduling results in suboptimal performance for all but the lightest workloads.[1] More effective scheduling algorithms exploit information about individual requests and the current state of system components. Scheduling algorithms are partitioned based on whether they use scheduling information from "below" the scheduler (e.g., disk drive state) or information from "above" the scheduler (e.g., request priorities).

Although previous studies provide some insight into the general performance behavior of scheduling algorithms, many of their assumptions are not valid for modern high-performance computer systems. The complex characteristics of current disk drives cannot be duplicated with simple analytical or simulation models [Ruem94]. Synthetic or benchmark

[1] [Wilh76] constructs a synthetic workload with very high physical locality for which FCFS provides good performance for some subsaturation arrival rates.

workloads with simple probability distributions for request starting locations or request inter-arrival times are unrepresentative of real-world systems [Ruem93, Bate91, Henl89, McNu86, Scra83, Lync72]. Disk subsystem performance metrics cannot accurately predict the effects of scheduling design and implementation choices on overall system performance (i.e., the elapsed times or throughput of application-level tasks) [Gang93]. Each of the studies discussed below suffers from one or more of these inadequacies.

3.2.1 Scheduling with Information from "Below"

Traditional scheduling algorithms use hardware-specific knowledge to reduce mechanical delays. Some algorithms minimize seek times while others minimize overall positioning times (i.e., combined seek and rotational latencies). Modern SCSI disk drives have mean seek times of approximately 10 ms (for random disk accesses) and maximum (or **full-stroke**) seek times of approximately 20 ms. Modern disk platters rotate at 5000 RPM or more, resulting in mean rotational latencies of approximately 4–6 ms. Thus, both seek times and rotational latencies have a large impact on mean request service times.

Reducing Mean Seek Time

Over 25 years ago, Denning used a simple analytical model to study the advantages of a **Shortest Seek Time First** (SSTF) policy [Denn67]. This algorithm always schedules the pending request that will incur the smallest seek time given the current disk actuator position. Since it is infeasible to exactly predict seek times (see section 2.1.1), schedulers typically approximate SSTF by using seek distances (i.e., the number of cylinders to be traversed). SSTF reduces mean response times over a wide range of workloads. However, SSTF is a greedy scheduling algorithm; the potential for starvation of individual requests increases with workload intensity. With SSTF scheduling of a heavy workload, a disk's actuator 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 region. In particular, SSTF discriminates against requests to the innermost and outermost cylinders.

Denning also examined the **SCAN** or "elevator" algorithm, which provides lower response time variance than SSTF with only a marginal increase in the mean response time (for the synthetic workloads studied). This algorithm is named for the way that the disk actuator 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 SCAN passes over every cylinder during each **phase** of the scan, it resists starvation more effectively (i.e., has a lower response time variance) than SSTF. However, the disk arms pass through the center region of the disk at more regular intervals than the edges. Requests to the middle cylinders therefore receive better service.

The SCAN algorithm has a number of variations. The **Cyclical SCAN** algorithm (C-SCAN) replaces the bidirectional scan with a single direction of travel [Seam66]. When the actuator reaches the last logical cylinder, a full-stroke seek returns it to the first cylinder without servicing any requests along the way. C-SCAN treats each cylinder equally rather than favoring the center cylinders. The **LOOK** algorithm, another SCAN variant, reverses

the scanning direction when no pending requests exist in the current direction of travel [Mert70]. The **C-LOOK** algorithm combines C-SCAN and LOOK.

Teorey and Pinkerton used both analytical and simulation models to analyze the performance of several scheduling algorithms, including FCFS, SSTF, LOOK, C-LOOK, and some hybrid algorithms optimized to handle extremely heavy workloads [Teor72]. Their simple models assumed a linear seek curve, zero command and completion overheads, no bus activity, uniformly distributed request starting locations, and a constant number of outstanding requests (i.e., a closed model). They showed that SSTF provides the lowest response times at the cost of poor starvation resistance. They defined a new metric for evaluating scheduling algorithm performance that includes the mean service time, the mean queue time, and the response time variance. Using this metric, they concluded that the best scheduler implementation should use a dynamic algorithm policy that employs LOOK when the workload intensity is low and C-LOOK when it is high. They suggested an implementation with some hysteresis at the crossover point between the algorithms.

Daniel and Geist proposed **VSCAN(R)**, a continuum of scheduling algorithms between SSTF and LOOK [Dani83]. The R parameter indicates the algorithm's bias towards maintaining the current direction of actuator motion. VSCAN(0.0) is equivalent to SSTF, and VSCAN(1.0) is equivalent to LOOK. A subsequent study by the same authors attempted to identify the optimal value for R [Geis87]. They demonstrated that VSCAN(R) with a low value of R provides mean response times within a few percent of STTF and response time variances within a few percent of LOOK. However, they used simple synthetic workloads with either uniformly, unimodally, or bimodally distributed request starting locations. Furthermore, their simulator configuration matched that reported in [Teor72].

They also implemented VSCAN(R) in a UNIX device driver on a DEC PDP-11/70 with two 134 MB SMD Fujitsu disk drives. The system workload consisted of eight or more university researchers doing program development, text editing, and some text formatting. The results indicated that VSCAN(0.2) provides higher throughput (and lower or equivalent mean response times) than FCFS, SSTF, LOOK, and VSCAN(0.1). They concluded that VSCAN(0.2) provides a good balance between the response time mean and variance performance metrics. Although this conclusion was reasonable for previous computer systems, this dissertation shows that the VSCAN(R) algorithm does not effectively exploit the on-board data caches found in current disk drives.

The **CVSCAN(N,R)** algorithm described in [Geis87a] augments VSCAN(R) to consider more than just the "closest" request in either direction from the current actuator position. Instead, the algorithm computes the mean seek distance to the closest N requests in each direction. So, CVSCAN(1,R) is equivalent to VSCAN(R). Geist, et al., used both simulation and actual implementation to test the performance of CVSCAN(N,R) for various N and R values. Using the same simulator configuration as in [Geis87], they determined the mean and standard deviation for request queue and service times. Values of N greater than unity did not provide significant improvement for CVSCAN(N,R). They also compared scheduling algorithms by actual implementation and determined that SSTF provides the highest throughput and is the most susceptible to request starvation. LOOK is the least susceptible and provides 96% of the throughput of SSTF. Versions of VSCAN(R) and CVSCAN(N,R) all provide performance roughly equivalent to that of LOOK. Their study failed to demonstrate any significant advantage of either VSCAN(R) or CVSCAN(N,R) over the more traditional

scheduling algorithms. In addition, the benchmark workload was heavily weighted towards read requests for file system directory blocks clustered around a few points in the logical space. As modern operating systems typically cache frequently accessed directory blocks in main memory, this workload was quite unrealistic.

WSCAN(R), or **Window SCAN**, represents a slightly different continuum of scheduling algorithms between SSTF and LOOK [Sugg90]. WSCAN(R) emulates the LOOK algorithm while pulling a "service window" along behind. The R parameter indicates the size of the service window. When the algorithm schedules requests within the window, neither the "current" direction of travel nor the window boundaries change. WSCAN(0.0) is equivalent to LOOK, and WSCAN(1.0) is equivalent to SSTF. Suggs used the same simulator as [Geis87] to determine that WSCAN(R) performs best with a value of R between 0.01 and 0.03. He demonstrated that for several very short sequences of requests, WSCAN(R) provides a mean seek time closer to "optimal" than SSTF, SCAN, and VSCAN(0.2). Using a scheduler implementation and workload similar to [Geis87a], he showed that WSCAN(0.02) provides a 12.8% reduction in mean response time over VSCAN(0.2).

The WSCAN(R) algorithm shows more promise than VSCAN(R) for high-performance disk request scheduling, as it will temporarily "turn around" and service sequential requests in logically ascending order (during its descending phase). VSCAN(R) never reverses direction unless there are no requests within the nearest fifth of the logical space "below" the current location. And when it does turn around, it does so permanently. That is, it changes from the descending phase to the ascending phase. Experiments in this dissertation demonstrate that this behavior causes significant performance degradation when scheduling sequential requests for disks with prefetching on-board caches. Unfortunately, the WSCAN(R) algorithm is not studied in this dissertation.

XSCAN(R), a temporal variation of WSCAN(R), updates the "current" head position at the completion of each request rather than the initiation [Geis95]. That is, a request arriving at the scheduler may be chosen as the next to receive service if it is "closest" to the position of the last request completed (i.e., not the request currently being serviced). This algorithm gives priority to logically sequential (or nearly sequential) streams of synchronous requests when the compute time between request completions and subsequent initiations is small. Geist and Westall examined the performance of FCFS, WSCAN(R), and a read-prioritized version of C-LOOK using a synthetic Linux file system workload. This workload was almost pathologically worst-case for the C-LOOK algorithm, at least partially invalidating the results of their study. The file system workload consisted of eight processes issuing synchronous single-sector read requests to eight "files" in the synthetic workspace. Each process chose a new file to read after reading the last block of the previous file. For a C-LOOK scheduler and a 4-segment data cache, such a workload results in a very low cache hit rate. The scheduler services one request from each process during each scan across the logical space. When WSCAN(R) or XSCAN(R) is used to schedule this workload, the actuator often alternates between just two files (i.e., two cylinders) until one or the other has been completely read. In this scenario, the automatic prefetching mechanism of the disk provides a much higher cache hit rate — a hit rate of over 50% is reported. Given the poor choice of workload in [Geis95], it is unclear what advantage (if any) XSCAN(R) has over WSCAN(R) or any other scheduling algorithm.

Reducing Mean Positioning Time

Given detailed disk configuration and state information, a scheduler can choose the pending request that will incur the minimum positioning time. Several studies have examined the performance of such a scheduling algorithm. Denoted as **Shortest Time First** (STF) in [Selt90] and **Shortest Access Time First** (SATF) in [Jaco91], the term **Shortest Positioning Time First** (SPTF) is used in this work to clarify the exact purpose of this algorithm. Seltzer, et al., simulated a disk subsystem with a constant number of pending requests (up to 1000). Jacobson and Wilkes simulated a disk subsystem with a Poisson arrival process for requests. Both studies used workloads with uniformly distributed request starting locations.

The disk utilization numbers presented in [Selt90] indicate that SPTF provides up to 60% higher performance than C-LOOK or SSTF for the heaviest workloads. At the same time, the request response time variance for SPTF (a greedy algorithm) rises dramatically with increasing workload intensity. The maximum response time observed for SPTF at a queue length of 1000 is 4.5 times higher than that of C-LOOK or SSTF. Jacobson and Wilkes showed that SPTF saturates the studied disk subsystem at an arrival rate of approximately 70 requests per second, as compared to 50 requests per second for VSCAN(0.2) and SSTF. These two studies used better disk drive models than those used in previous simulation studies of scheduling algorithms. However, they also used unrealistic workloads with simplistic host models.

Sector-VSCAN(R), a variation on VSCAN(R), also takes rotational latency into account [Reyn88]. The R parameter indicates the algorithm's bias towards maintaining the current direction of actuator motion. Reynolds showed that Sector-VSCAN(R) provides up to a 5% improvement in mean response time over VSCAN(R) for a synthetic file system workload.

By detecting and scheduling sequential streams of requests in logically ascending order, most of the algorithms described in this chapter can be modified to further reduce service times. Given that logically sequential blocks typically map to physically sequential media locations, sequential read requests (and write requests in some cases) incur zero positioning delays when serviced in logically ascending order. A scheduler external to the disk drive also has the option of **concatenating** (combining) sequential requests into a single larger request. A concatenated request incurs less total processing and protocol overheads than would the set of contributing requests. On the other hand, none of the individual requests "complete" until the entire concatenated request completes. This may artificially increase the service times for some of the contributing requests.

3.2.2 Scheduling with Information from "Above"

This section describes scheduling algorithms that utilize system-level information (see section 2.2) to achieve performance and reliability goals. In the absence of such information, a scheduler cannot identify "high priority" requests. However, it can control response time variance and thereby reduce the possibility of starving high priority requests.

Reducing Response Time Variance

Most of the algorithms described in section 3.2.1 can be modified to reduce response time variance. The resulting algorithms are classified as either **batch** or **aging** algorithms. Batch algorithms service a subset of the pending requests, selected by some set of criteria, before moving on to other requests. Possible criteria include request arrival times and target cylinder numbers [Jaco91, Selt90, Coff72, Teor72, Fran69]. Aging algorithms give priority to requests that have been in the request queue for excessive periods of time. Some algorithms gradually increase the priority as a request ages [Jaco91, Selt90]. Alternately, a time limit may be set after which requests move to a higher priority. In either case, if an aging algorithm gives too much weight to the queue time (age) component, it degenerates into FCFS [Gotl73]. For this reason, aging algorithms must be carefully designed — preferably after a thorough analysis of the specific workload to be scheduled.

Scheduling for System Performance

If entities above the scheduler provide information about request inter-relationships and/or the role of individual requests in achieving system goals (e.g., performance and reliability metrics), the scheduler can more effectively focus on improving overall system performance. The resulting request orderings may actually be "suboptimal" as measured by disk subsystem metrics.

Ganger and Patt performed an initial study of disk scheduling using system-level information obtained from a UNIX file system [Gang93]. They classified requests as time-critical, time-limited, or time-noncritical based on whether or not host processes block (or will block) after issuing the requests. They modified a device driver executing the LOOK scheduling algorithm to give priority to time-critical and time-limited requests. For a file compression workload, application runtimes dropped 13–14% when compared to the conventional LOOK scheduler. At the same time, the mean request service time increased by 86%, the mean seek time increased by over 180%, and the mean response time increased by over an order of magnitude. Their work plainly demonstrated the potential benefits of making system-level information available to the disk scheduler.

Real-time system research has recently focused on disk scheduling algorithms that use transaction deadlines to reorder outstanding disk requests. Abbott and Garcia-Molina have studied scheduling algorithms that prioritize disk requests based on the priority of the issuing transactions [Abbo89] or a combination of transaction priority and seek distance [Abbo90]. They show that the former provide improved performance over FCFS and the latter provide improved performance over traditional seek-reducing algorithms. In particular, a scheduling algorithm that considers the "feasibility" of deadlines (i.e., whether or not a disk can service a request before the issuing transaction's deadline expires) is shown to provide the highest system performance (as measured by the number of missed deadlines). In the first study, disk service time was held constant at 25 ms per request. In the second study, they used a very simple disk model lacking most of the advanced features of a modern disk drive. A Poisson arrival process was used to generate single-track disk requests, and starting locations were uniformly distributed.

Carey, et al., used a similar methodology to examine a priority-based LOOK algorithm [Care89]. A more complex host model was used to more closely emulate real database activity. Their LOOK algorithm serviced all high priority requests before servicing any low priority requests. They recognized that the number of priority levels is inversely related to the mechanical delay reduction possible using such a multi-queue scheduler; a simple two-queue scheme allows significant scheduling flexibility, especially for heavier workloads. They concluded that it is essential to use a priority-based disk scheduler (in conjunction with a priority-based buffer management algorithm) whenever the disk subsystem is a performance bottleneck.

Chen, et al., compared the performance of several real-time scheduling algorithms, including the best algorithms from [Abbo90] and [Care89] along with two new real-time algorithms based on SSTF [Chen91]. The SSTF-based algorithms use both seek distances and request deadlines in the generation of request orderings. The algorithms can be biased toward either of the criteria by changing certain scheduling parameters. The comprehensive database model used in the experiments was validated against an actual real-time database testbed, but the disk model was very simplistic. In particular, the lack of an on-board data cache in the disk model undoubtedly affected the performance results for their experiments using workloads with high access locality (up to a 0.8 sequential access probability). Most of their experiments, however, assumed a uniform distribution of request starting locations. For their simulation environment, the two SSTF-based algorithms provided the best system performance.

Kim and Srivastava examined real-time disk scheduling algorithms that differentiate between reads and writes when assigning request priorities [Kim91]. The best performance resulted from an algorithm that assigns read request priorities based on the priority of the issuing transaction and write request priorities based on the priority of the transactions waiting for the release of the corresponding write locks. If no transaction is waiting for a write to complete, the write is set to the lowest priority possible. Their simulator did not include a disk model; they assumed a fixed disk service time of 25 ms.

In the absence of explicit system-level information, some schedulers make certain assumptions about request priorities. For example, a scheduler can give priority to read requests (over write requests) in an attempt to reduce application runtimes. This heuristic is driven by the fact that application processes often wait for read requests, while write requests are usually buffered in main memory using write-back disk caches. Unfortunately, such heuristics are only effective in certain situations and may actually degrade performance during bursts of heavy activity. It is easy to devise cases where optimal performance requires the scheduler to give priority to write requests (over read requests). For example, speculative prefetch requests issued by file systems or applications should not be given precedence over synchronous write requests (e.g., file system metadata updates). As a more complex example, if a disk block cache in main memory cannot hold any more dirty blocks, it is vital to quickly clean some fraction of the cache (see section 2.2.2). Giving write requests low priority slows the process of cleaning the disk cache, indirectly reducing application performance.

3.3 Significant Room for Improvement

This chapter describes the state-of-the-art in disk scheduling. Clearly, significant room for improvement exists. Interfaces and protocols should be augmented to allow useful scheduling information to reach a centralized scheduler or pass between distributed schedulers. Components along the I/O path should be modified to extract and pass such information to the scheduler(s). Centralized scheduling algorithms should be upgraded to utilize more of the available information, and distributed scheduling algorithms should be devised that enable cooperative scheduling.

Accurately identifying effective scheduling algorithms and scheduler implementations requires a better performance evaluation methodology than has been used in previous work. The studies discussed above all suffer from one or more failings. Most use disk models that lack important disk drive characteristics (e.g., request processing overheads, onboard data caches, and complex data layouts). The experiments typically use synthetic workloads with simple probability distributions for request starting locations (e.g., uniform, unimodal, or bimodal), which are unrepresentative of most real-world workloads [Ruem93, Bate91, Henl89, McNu86, Scra83, Lync72]. Those studies that did involve actual implementation used very simplistic benchmarks. With few exceptions, previous work ignored host feedback effects (e.g., by using a Poisson process for request arrivals) or oversimplified them (e.g., by maintaining a constant number of outstanding requests).

CHAPTER 4

Methodology

The experiments in this dissertation involve four elements: a trace-driven simulator comprised of detailed models of both the host and the disk subsystem; a set of traces (primarily taken from real-world systems) used as input to the simulator; the disk scheduling algorithms and scheduler implementations being studied; and the performance metrics used to evaluate scheduler designs. This chapter describes each of the four elements.

4.1 The Simulation Model

As part of a larger research effort into I/O subsystem design issues, an extensive trace-driven simulator was developed for studying disk subsystems. It contains multiple models of host systems, buses, caches, controllers, and disk drives. The simplest host model merely interprets traces of disk request activity. The most complex host model simulates the execution of application and O/S processes on one or more processors. The disk models range from simple statistical delay mechanisms to detailed, highly-validated models of modern SCSI disk drives.

4.1.1 Host Models

The simulator supports several host models that consume different types of traces. For experiments driven by **disk request traces**, request arrivals are modeled as in an open system. Requests are issued to the disk subsystem model using traced interarrival times (without regard for request completion times). The open system approach was chosen for two reasons. First, it retains the burstiness inherent in real workloads (as opposed to the fixed queue lengths of the closed system approach). Second, scaling of the traced interarrival times creates a smooth continuum of workloads.

For experiments driven by **full system traces**, a more detailed host model provides realistic feedback based on request completions. The **process-flow model** reproduces the execution of processes in the host system and their interaction with the disk subsystem [Gang93]. It recreates each process by consulting a timestamped trace of important events encountered during the lifetime of that process (e.g., fork, exit, sleep, and disk request generation). It also generates and handles clock and I/O interrupts. Because the process-flow model simulates how processes interact with their disk requests, it enables the partitioning of

HP C2247 Disk Drive	
Formatted Capacity	1.05 GB
RPM	5400
Mean Seek Time	≈10 ms
Data Surfaces	13
Cylinders	2051
Sectors	2054864
Zones	8
Sectors/Track	56-96
Interface	SCSI-2
Cache	256 KB

Table 4.1: Basic Characteristics of the HP C2247 Disk Drive

disk requests into multiple classes. Disk scheduler(s) can use this classification information to tune schedules for better overall system performance.

4.1.2 Disk Drive Models

The simulator contains all of the modern disk drive features described in section 2.1.1, including zoned recording, spare regions, defect slipping and reallocation, disk buffers and caches, various prefetch algorithms, fast writes, bus delays, control and communication overheads, and command queueing. For the studies reported in this dissertation, the simulator was configured to model the HP C2240 line of disk drives [HP92]. Table 4.1 lists some basic specifications for the HP C2247 [HP92a].

To accurately model this line of drives, an extensive set of parameters was obtained from published documentation and by monitoring SCSI activity. The experimental platform consisted of an NCR 3550 multiprocessor system equipped with NCR 53C700 SCSI I/O processors and HP C2247 disk drives. Extraneous system activity was minimized during the monitoring process to reduce host delays. Various mechanical positioning delays, control and communication overhead values, path transfer rates, and cache management strategies were extracted. Exact LBN-to-PBN mappings were determined for several disks, providing information on zoning, sparing, and defects. The appendix provides a complete list of the disk parameters and extracted values.

The disk model was validated by exercising an actual HP C2247 and capturing traces of all SCSI activity. The simulator replayed each traced request stream, using the observed 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 mean 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 measured and simulated response time distributions [Ruem94]. Figure 4.1 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

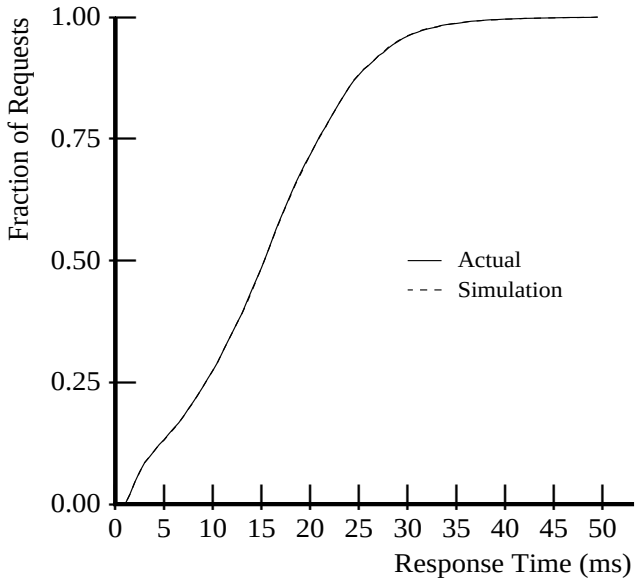


Figure 4.1: Response Time Distributions for a Validation Workload (50% reads, 30% sequential, 30% local [normal, 10000 sector variance], 8KB mean request size [exponential], 0-22 ms request interarrival time [uniform])

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 4.1 is 0.07 ms, which is less than 0.5% of the corresponding mean response time. The worst-case demerit figure observed over all validation runs was only 1.9% of the corresponding mean response time.

Experiments in this dissertation use the HP C2240 line of disk drives for three reasons. First, the simulator correctly models the observed behavior of these drives. Second, they have most of the advanced features described in section 2.1.1. Third, detailed specifications for the disks used in some of the traced systems are unavailable.

This disk substitution leads to two significant difficulties. First, the base experimental disk (the HP C2247) has a different storage capacity than the disks used by some of the traced systems. To better match the size of the simulated disks to those in the traced systems, the number of platters is reconfigured to create disks large enough to contain the active data without excessive unused media. This is not unreasonable, since a production line of disks often differs only in the number of platters [HP92a].

The second and more important issue is that HP C2240 disks service requests at a different rate than the disks in the disk request traces. To produce heavier or lighter workloads, the traced interarrival times are scaled to produce a range of mean 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 cut in half (doubling the mean arrival rate). Even with an identity scaling factor, however, the workloads would undoubtedly have been

different if the systems traced had been using HP C2240 disk drives; information about how individual request arrivals depend upon previous request completions is not present in these traces. This is a common problem with trace-driven simulation, but should not invalidate the qualitative results and insights. Also, the experiments driven by the full system traces generally provide corroborating evidence.

Command Queueing

The simulator contains a number of parameters related specifically to disk command queueing. Unfortunately, a methodology for validating disk command queueing models is beyond the scope of this dissertation. When a cache-sensitive algorithm is used by a disk-based scheduler, the disk model's cache is probed before selecting a new request for service. This represents a more aggressive approach than exists in current disk-based schedulers, which typically probe the cache (for scheduling purposes) only when a request arrives [Lary95]. Enabling command queueing does not affect the values for the command and completion overhead parameters, although some of the overheads may occur in parallel (due to request "pipelining" at the disk). Experiments in this dissertation use two command queueing configurations, denoted **Preseek** and **Full**. The three components of inter-request concurrency in the disk model are preseeking, read hits under misses, and write prebuffering.

Preseeking

After a disk finishes the media access for a request, it transfers any remaining cached data to the host (in the case of a read request) and performs cleanup/completion processing. At the same time, the disk can begin any mechanical positioning required for the next request in the on-board queue. This is denoted as **preseeking**. Both Preseek and Full experimental command queueing configurations allow preseeking. In the absence of pending requests, disks typically prefetch sequential data into the on-board cache after a read request.

Read Hits Under Misses

The Full command queueing configuration allows read requests that hit in the on-board cache to be serviced in parallel with any ongoing media access. There are three forms of **hits under misses** enabled in this configuration. First, any new read request whose data is entirely contained in the on-board cache (a **full read hit**) is serviced immediately (without any intervening disconnect) followed by any necessary cleanup/completion activity. Second, any new read request whose first sector is contained in the cache (a **partial read hit**) will have the available data transferred to the host prior to disconnecting. Third, whenever a disk disconnects from a bus, its pending queue is searched for read requests that have "enough" of their requested data available in the cache to warrant reconnection and subsequent data transfer (an **intermediate read hit**). The value of "enough" is determined by the Buffer Full Ratio, as specified by the SCSI peripheral bus protocol [SCSI93]. The request currently performing media access always has priority over intermediate read hits when bus activity is required.

Write Prebuffering

The Full command queueing configuration allows a disk to transfer all or part of a write request's data to the on-board cache in parallel with any ongoing media access (if there is an appropriate cache segment available). This is denoted as **prebuffering**. In addition, whenever a disk disconnects from a bus, its pending queue is searched for write requests that can be assigned to an available segment or have "enough" space available in their cache segment to warrant reconnection and subsequent data transfer (an **intermediate write hit**). The value of "enough" is determined by the Buffer Empty Ratio, as specified by the SCSI peripheral bus protocol [SCSI93]. The request currently performing media access always has priority over intermediate write hits when bus activity is required.

When command queueing is enabled, the default on-board cache configuration is modified to increase the number of cache segments. For experiments without command queueing, each on-board data cache contains two 256-sector (128 KB) segments, only one of which is usable for write request data.[1] For experiments with command queueing, each on-board cache contains four 128-sector (64 KB) segments, allowing greater inter-request concurrency at each disk. A maximum of two segments can be dirty (if prebuffering write data). Experiments in this dissertation point out the sensitivity of scheduler performance to on-board cache configuration, although a full study of scheduler/cache interaction is left for future work.

4.2 Workloads

A small number of the experiments use synthetically-generated disk request traces, primarily to allow comparison with previous work. The remainder use traces captured from actual systems. Hewlett-Packard and Digital Equipment Corporation have provided six extensive traces of disk request activity from various customer and research systems. In addition, system and disk activity was traced on an NCR workstation to enable experiments using the process-flow host model.

4.2.1 Disk Request Traces

The six HP and DEC traces are described only briefly as they have been discussed elsewhere in more detail [Ruem93, Rama92]. The traced workloads span a broad range of environments, and each trace is at least a full workshift (8 hours) in length. Table 4.2 provides some basic information on each trace.

Two of the traces come from Hewlett-Packard systems running HP-UX, a version of UNIX [Ruem93]. *Cello* comes from a server at Hewlett-Packard Laboratories (Palo Alto, CA) 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, the experiments in this dissertation use a single week-long snapshot (5/30/92 to 6/6/92), as in [Wort94, Ruem93].

The other four traces are from commercial VAX systems running the VMS operating system [Rama92]. *Air-Rsv* is from a transaction processing environment in which about

[1] This is the manufacturer's default configuration for the HP C2247 disk drive.

Trace	Length (hours)	Disks	Requests	Mean Req 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 4.2: HP and DEC Disk Request Trace Characteristics

500 travel agents made airline and hotel reservations. *Sci-TS* is from a scientific time-sharing environment running analytic modeling software and graphical and statistical packages. *Order* and *Report* are from a machine parts distribution company. *Order*, collected during daytime hours, represents an order entry and processing workload. *Report*, collected at night, represents a batch environment generating reports of the day's activities.

4.2.2 Full System Traces

To drive the process-flow host model, system activity was traced on an NCR 3433 workstation containing a 33 MHz Intel 80486 microprocessor and 48 MB of main memory. A single HP C2247 disk drive serviced all of the disk requests in each trace. Full system traces were captured using an instrumented version of the SVR4 MP UNIX operating system that writes timestamped event data into a dedicated 8 MB kernel memory buffer. The instrumentation is unobtrusive, increasing the dynamic instruction count by less than 0.1% in the worst case (assuming that the trace buffer could not be otherwise used).

Traces were captured for two different types of workloads. *Compress* represents a single 30.7 MB file being compressed to 10.7 MB. The *SynRGen* traces capture the activity of the SynRGen synthetic file reference generator [Ebli94], configured to emulate an edit/make/debug environment with a parameterized number of users. Each user performs a series of random "tasks" intended to simulate editing, compiling, and executing files.

The trace buffer size limits the length of the NCR traces. Very light workloads take several hours to fill the buffer, but are uninteresting for disk scheduling research. Heavy workloads quickly fill the buffer, preventing the capture of "complete" application activity (i.e., all phases of application execution). As a result, the traces cover light-to-medium workloads. Since medium-to-heavy workloads benefit the most from aggressive disk scheduling, the performance improvements reported for experiments using the NCR traces are therefore conservative.

4.3 Scheduling Algorithms

This section describes the various scheduling algorithms compared in the following chapters, categorized by the type of information used by the algorithms.

4.3.1 Scheduling with Information from "Below"

In this dissertation, scheduling algorithms that use hardware-specific knowledge are partitioned by the level of information detail: scheduling based on LBNs only, scheduling given a full LBN-to-PBN mapping, and scheduling with full knowledge (including current read/write head position, command overheads, cache contents, etc.). Only schedulers with full knowledge have sufficient information to enable algorithms that reduce overall positioning times (i.e., combined seek and rotational latencies).

Experiments with **LBN-based** and **full-map** schedulers compare SSTF, C-LOOK, LOOK, VSCAN(0.2), and FCFS (for reference purposes). Experiments with full-map schedulers use a heuristic to select between requests to the same physical cylinder. The heuristic is based on the C-LOOK algorithm to take advantage of the HP C2247's prefetching cache and to reduce the number of head switches. A full-map scheduler satisfies all requests on the current track using C-LOOK, then all requests on the current cylinder using C-LOOK, and then the appropriate seek-reducing algorithm to select the next cylinder to service.

Experiments with **full-knowledge** schedulers use SPTF and a cache-sensitive version of SPTF termed **Shortest Positioning (w/Cache) Time First** [Wort94]. The SPCTF algorithm assumes a positioning time of zero for any read request that will hit in the on-board cache.[2]

4.3.2 Scheduling with Information from "Above"

The experiments in this dissertation explore overall system performance effects of various algorithms with and without explicit system-level information. Without such information, a scheduler's only recourse is to reduce the possibility of starvation for high priority requests by minimizing response time variance (i.e., reducing starvation in general). For this purpose, an age-weighted SPTF algorithm is also studied. **ASPTF(W)** is equivalent to the ASATF algorithm proposed in [Jaco91].[3] ASPTF(W) adjusts each positioning time prediction (T_{pos}) by subtracting the weighted amount of time the request has been waiting for service ($W * T_{wait}$). A scheduler uses the resulting effective positioning time (T_{eff}) to select the next request to issue:

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

The age-sensitive algorithm recommended in [Jaco91] is equivalent to ASPTF(6.3). The **Aged Shortest Positioning (w/Cache) Time First** algorithm, which combines age-

[2]SPCTF classifies a read request as a hit if the first requested sector is resident in the cache or is currently being fetched from the media.

[3]The ASATF algorithm was chosen instead of the WSTF algorithm suggested in [Selt90] because WSTF is less sensitive to differences in predicted positioning times when comparing requests whose waiting times are near the aging limit.

sensitivity and cache-sensitivity, is also included in the experiments. For a sufficiently large W, both ASPTF(W) and ASPCTF(W) degenerate into FCFS.

Request Criticality Information

The classification scheme introduced in [Gang93] is used for system-level partitioning of requests. **Time-critical** requests cause processes to immediately wait (or **block**) for disk service. **Time-noncritical** requests do not cause processes to block, but must still be serviced in order to update the on-disk copy of the data and free up shared resources (e.g., main memory). **Time-limited** requests become time-critical unless serviced within the corresponding time limit. The appropriate classification for a request is known by the system software that generates it.

To exploit request criticality information, a scheduler can maintain separate queues for requests with different priorities. The **2Q** schedulers used in this dissertation place time-limited and time-critical requests in a high priority queue and time-noncritical requests in a low priority queue. 2Q schedulers always empty the high priority queue before servicing requests from the low priority queue. For simplicity, both queues are scheduled using the same scheduling algorithm (e.g., 2Q C-LOOK).

4.3.3 Sequential Stream Optimizations

A scheduling algorithm can be optimized to improve performance for sequential request streams. Request **concatenation** combines one or more sequential requests into a single larger request. One combined request incurs less request processing and communication overhead than the total overhead experienced by several separate requests. A combined request may also incur less rotational latency, since there is no delay between the media accesses for the component requests. On the other hand, none of the individual requests can be reported as complete until the entire combined request has been serviced. This may inflate the response times for some of the contributing requests. Also, request concatenation requires the ability to perform scatter/gather DMA.[4] Otherwise, an additional memory-to-memory copy may be necessary to move the requested data to or from a single physically contiguous region of memory. The suffix **.CR** denotes concatenation of sequential read requests, and the suffix **.CW** denotes concatenation of sequential write requests (e.g., SSTF.CR and LOOK.CW.CR).

Sequential scheduling of sequential request streams imitates C-LOOK scheduling on a smaller scale. Using this optimization, a scheduler only services the "heads" of sequential streams (including streams consisting of a single request). A stream "head" is the request with the smallest starting LBN. For example, an SSTF scheduler with the sequential scheduling optimization will select the stream "head" that will incur the shortest seek time. The suffix **.SSR** denotes sequential scheduling of read requests, and the suffix **.SSW** denotes sequen-

[4]Large data transfers between memory and lower levels of the storage hierarchy are typically handled by a **Direct Memory Access** engine. To handle request concatenation, the DMA hardware must be able to dynamically redirect a stream of data to or from the physically discontiguous memory regions containing the individual components of a combined request.

tial scheduling of write requests (e.g., SSTF.SSR and LOOK.SSW.SSR). A scheduler may handle sequential read and write streams with different optimizations (e.g., SSTF.CW.SSR).

4.4 Metrics

For experiments using the simple host model, the mean disk request response time (across all simulated disks) is the primary metric used for comparing various scheduling algorithms and scheduler implementations. The squared coefficient of variation (σ^2/μ^2) of request response times is also reported, as in [Teor72]. Given a constant mean response time, a decrease in the coefficient of variation implies reduced response time variance (i.e., improved starvation resistance). Reducing starvation can indirectly improve system performance by reducing the possibility of high priority requests incurring excessive queueing time.

For experiments using the process-flow host model, several application-specific metrics are also utilized. For *Compress*, the simulator reports the application run time (i.e., the elapsed time of the compression process). For *SynRGen*, it reports the mean elapsed time for the individual user "tasks" (i.e., the mean **task completion time**). It also reports the mean **non-compute time**, which gives some indication of the amount of time each task spends waiting on disk activity. The latter metric is conservative, however, as it is obtained by subtracting the large computation delays (e.g., compile time and execution time) from the task completion times. Delays associated with waiting on shared processor resources or user activity are still part of the mean non-compute time.

CHAPTER 5

Centralized Scheduling at the Host

Centralized scheduling implies that a single entity along the I/O path is actively reordering pending requests. Other entities participate in disk scheduling only by extracting and passing useful information to the active scheduler. The host and the disk are examined as possible locations for a centralized scheduler. Both possess a significant quantity of useful information. Each location has certain advantages and disadvantages.

A host-based scheduler has easy and immediate access to system-level information. It has the freedom to reorder the entire set of outstanding requests, and the pending request queue length is limited only by the amount of main memory. A host-based scheduler can perform certain scheduling optimizations, such as request concatenation, that are not possible (or have little benefit) for disk-based schedulers. On the other hand, hardware-specific information must be extracted and transmitted to a host-based scheduler. For the most aggressive scheduling algorithms (e.g., SPCTF), the required accuracy and quantity of hardware-specific information make this a difficult task. Also, systems with centralized host-based schedulers cannot take full advantage of the inter-request concurrency offered by command-queued disks.

In this chapter, the centralized scheduler resides at the host (e.g., within the O/S device drivers). The experiments are partitioned based on the type of information used by the scheduler.

5.1 Scheduling with Information From "Below"

5.1.1 Synthetic Workloads

A small number of experiments were performed using synthetic workloads in order to replicate previous work and allow comparison with real-world workloads. The synthetic request streams 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 and heavier workloads. The ratio of reads to writes was set to 2:1. Each data point is the average of at least three separate runs of 250,000 disk requests, corresponding to simulated workloads of 50–400 hours of activity.

Scheduling by Logical Block Number

Even if a scheduler has little or no knowledge of the LBN-to-PBN mapping for a given disk, it can approximate seek delays using the "distance" between logical block numbers for individual requests. For example, an LBN-based C-LOOK scheduler will select a pending request for LBN 200 over one for LBN 300 if the last request to complete accessed LBN 100. The accuracy of this approximation depends on the choice of scheduling algorithm, the variance in sectors per track between zones, the defect slipping/reallocation scheme(s), and any cache prefetching activity.

Figure 5.1a presents the mean response times for FCFS, LOOK, C-LOOK, VSCAN(0.2), and SSTF for a range of mean arrival rates. FCFS quickly saturates as the workload increases. Since C-LOOK reduces the response time variance (as well as the mean), its mean response time runs 5–10% higher than those of LOOK, VSCAN(0.2), and SSTF for medium and heavy workloads. The 95% confidence intervals shown in figure 5.1b support these observations. However, the intervals for LOOK (which are typical for VSCAN(0.2) and SSTF as well) make it difficult to predict the relative positions of LOOK, VSCAN(0.2), and SSTF for a given arrival rate.

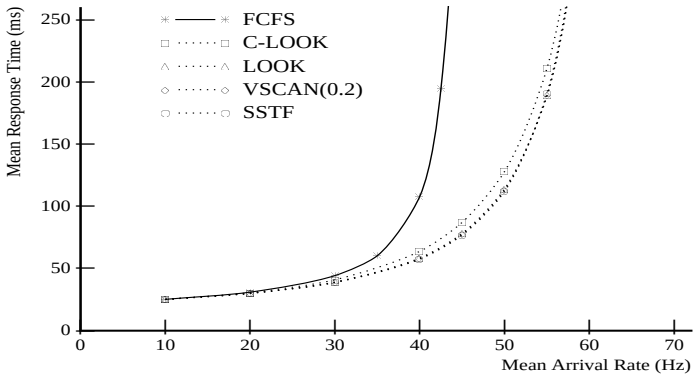
Figure 5.1c shows the squared coefficients of variation (σ^2/μ^2) for the same set of experiments. FCFS has the lowest coefficient for the lightest workloads. As FCFS begins to saturate, C-LOOK emerges as the algorithm with the best starvation resistance. SSTF, on the other hand, is highly susceptible to starvation. At an arrival rate of 50 requests per second, the coefficient of variation for SSTF is 64% greater than that of C-LOOK.

Scheduling with a Known Mapping

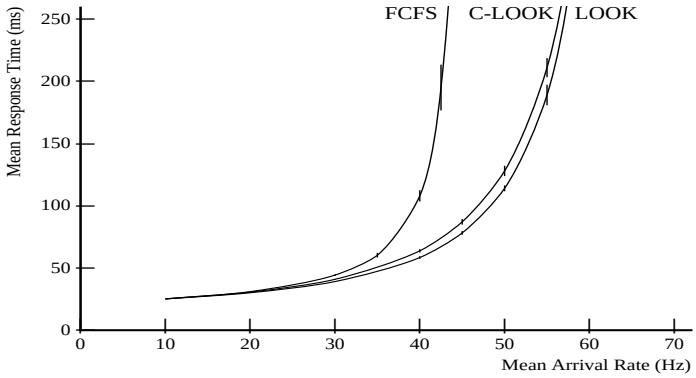
If a scheduler has knowledge of LBN-to-PBN mappings, it can more accurately predict seek delays and thereby produce better schedules. For the following set of experiments, the scheduling algorithms have full mappings for each cylinder and a heuristic for choosing between multiple pending requests to the same cylinder (see section 4.3.1).

Figure 5.2 displays the mean response times and squared coefficients of variation for the LBN-based algorithms and the algorithms utilizing a full LBN-to-PBN mapping. The LBN-based and full-map versions of each algorithm produce almost identical performance (as measured by either metric). Since workloads with uniformly distributed request starting locations contain few sequential requests, prefetching provides little or no performance improvement. The cylinder heuristic also has a minimal effect, as individual cylinders rarely contain multiple pending requests. The only visible deviation between algorithm versions is a slight increase in SSTF's coefficient of variation for the full-map algorithm. This is reasonable, as SSTF utilizes mapping knowledge strictly to reduce response times (at the expense of increased request starvation).

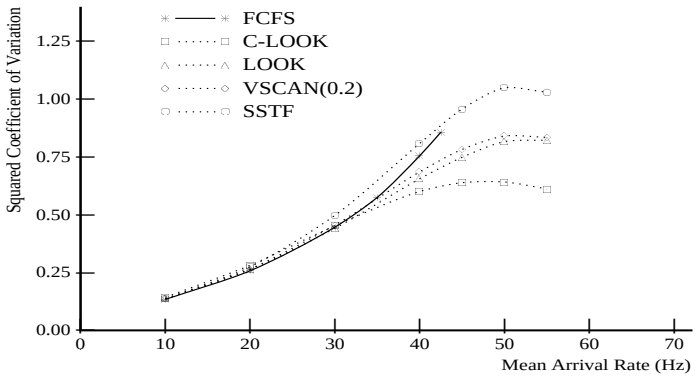
To determine the influence of excessive defects on scheduler accuracy, the simulator was reconfigured to model an HP C2247 with half of its spare tracks (450 out of 900) occupied due to randomly grown defects. To maximize the perturbation in the LBN-to-PBN mapping, all of the defects resulted from dynamic reallocation (i.e., no slipped tracks). Even with over 2% of a disk's tracks remapped, full mapping information provides little improvement in performance. SSTF, the algorithm most sensitive to the LBN-to-PBN mapping, improves



(a) Mean Response Time

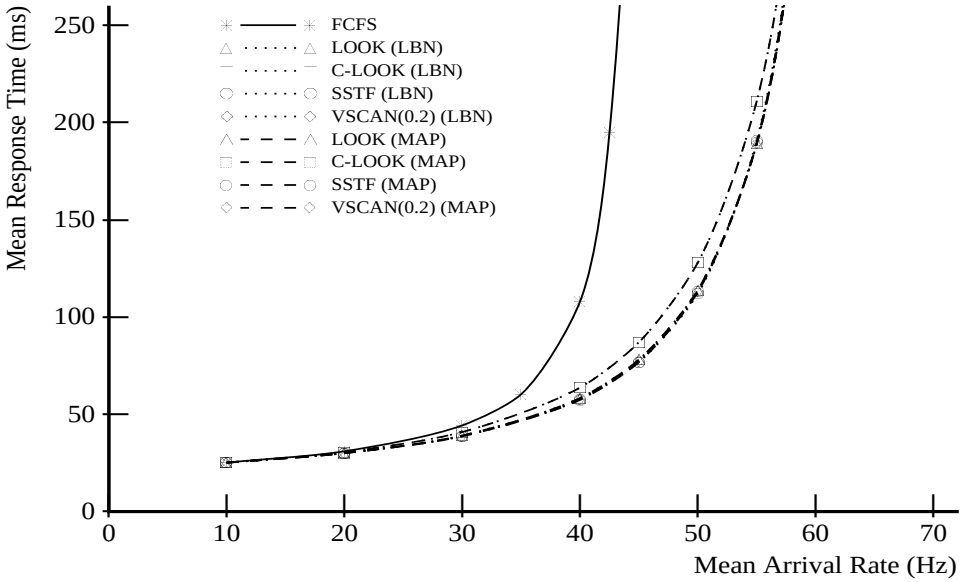


(b) 95% Confidence Intervals (Mean Response Time)

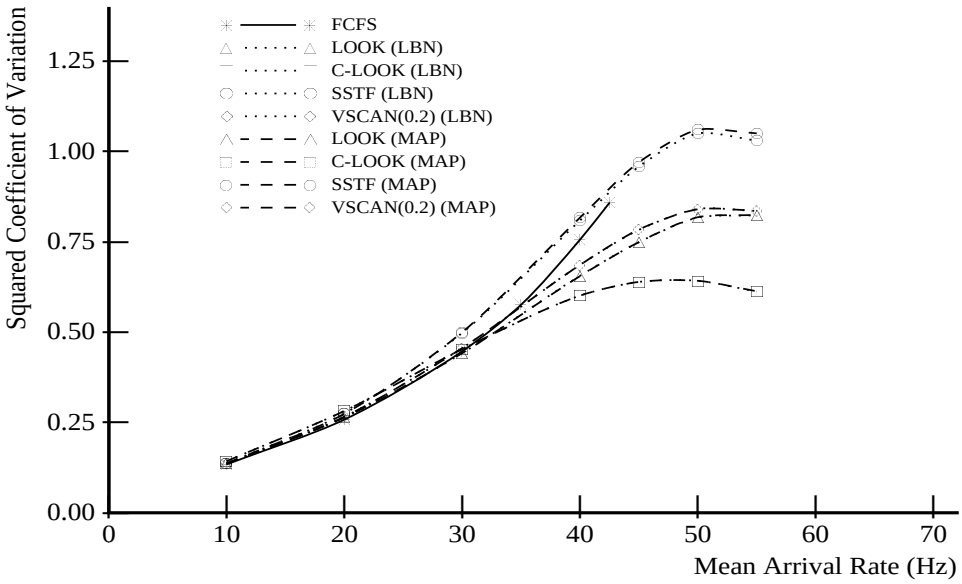


(c) Squared Coefficient of Variation

Figure 5.1: LBN-Based Algorithm Performance using a Synthetic Workload with Uniformly Distributed Request Starting Locations



(a) Mean Response Time



(b) Squared Coefficient of Variation

Figure 5.2: Full-Map and LBN-Based Algorithm Performance using a Synthetic Workload with Uniformly Distributed Starting Locations

by less than 1%. Thus, seek-reducing algorithms obtain only a marginal benefit from access to full LBN-to-PBN mappings when scheduling workloads with uniformly distributed request starting locations.

Scheduling with Full Knowledge

Given sufficient computation resources, a full LBN-to-PBN mapping, accurate mechanical and overhead specifications, and some indication of the current rotational position of the actuator, a scheduler can select the pending request that will incur the smallest total positioning time (i.e., seek and rotational latency) for a given disk. As with SSTF, the basic SPTF algorithm is highly susceptible to request starvation.

Figure 5.3 presents performance data for FCFS, C-LOOK, SSTF, SPTF, and ASPTF(W) (an age-sensitive version of SPTF described in section 4.3.2). As W increases from 2 to 30, the mean response time of ASPTF slowly grows, while its response time variance drops significantly. The SPTF and ASPTF algorithms have consistently lower mean response times than the seek-reducing algorithms or FCFS. For higher values of W, ASPTF suffers a sharp increase in mean response time as a disk begins to saturate. For a sufficiently large W, ASPTF degenerates into FCFS.

Modifying SPTF with an appropriate aging factor results in an algorithm with the mean response time of SPTF and the starvation resistance of C-LOOK. For example, ASPTF(6) and ASPTF(12) provide equivalent mean response times to SPTF for all but the heaviest workloads, yet have much better starvation resistance. In fact, ASPTF(12) has a lower response time variance than C-LOOK, even though its coefficient of variation is slightly higher.

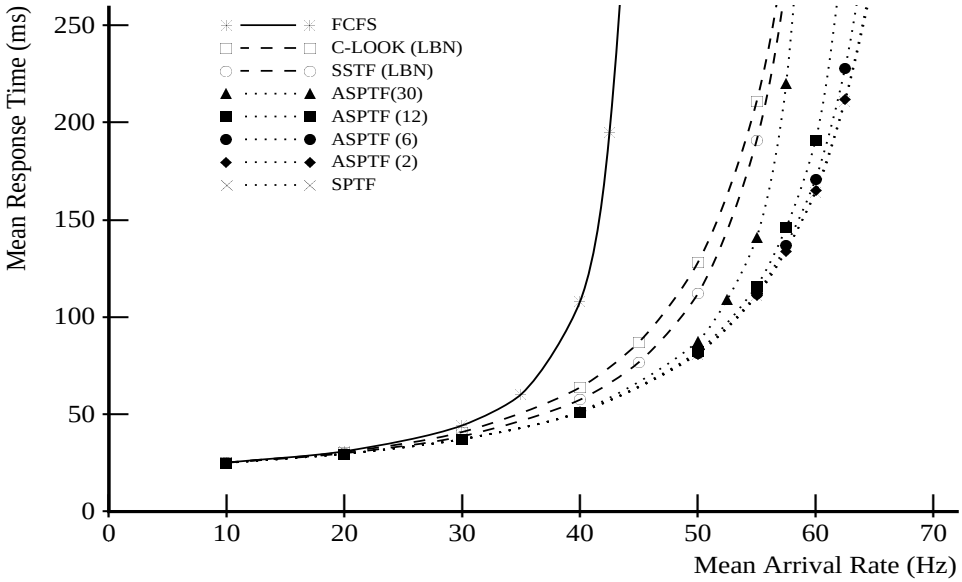
Selecting the request with the smallest positioning delay entails significant computation (especially for large queues), cf. [Jaco91]. Unless the pending queue size is bounded, it may be unacceptable to simply compute and compare the positioning times for all of the pending requests. One area for future research is the design and implementation of clever data structures to organize pending queues in such a manner as to streamline the search for the "closest" pending request. The addition of an aging factor further complicates this problem.

5.1.2 Disk Request Traces

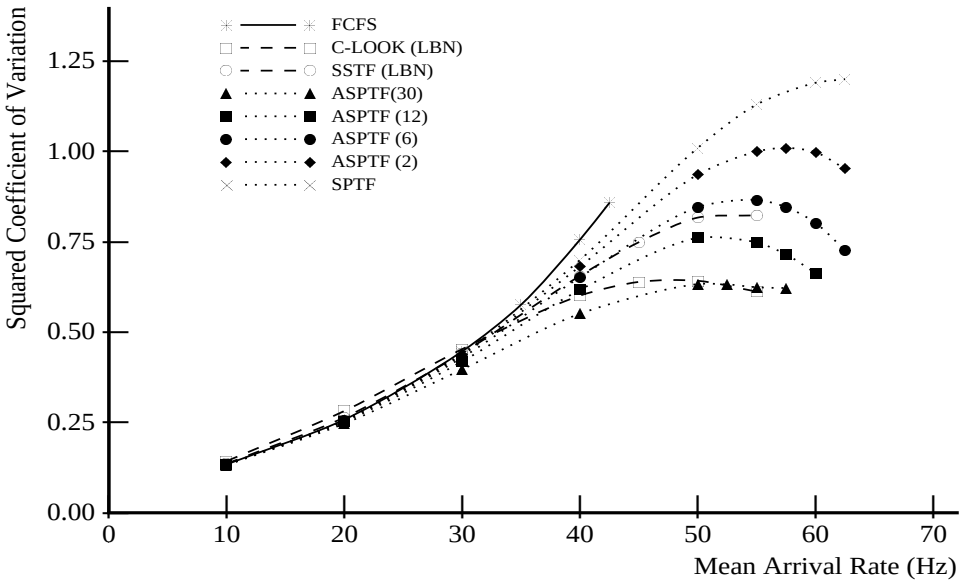
This section contains experiments comparing host-based schedulers for the HP and DEC traces. The performance graphs in this section use a range of trace scaling factors. Note that the scaling factor (the X-axis) is shown in $\log_2$ scale. The workloads with an identity scaling factor correspond to the traced request streams. The initial experiments compare the performance of the baseline scheduling algorithms. Subsequent experiments study algorithm performance with sequential scheduling optimizations and FCFS command-queued disks.

Scheduling by Logical Block Number

Figures 5.4–5.9 present the mean response times and the corresponding squared coefficients of variation for LBN-based scheduling of the six HP and DEC traces. Except in

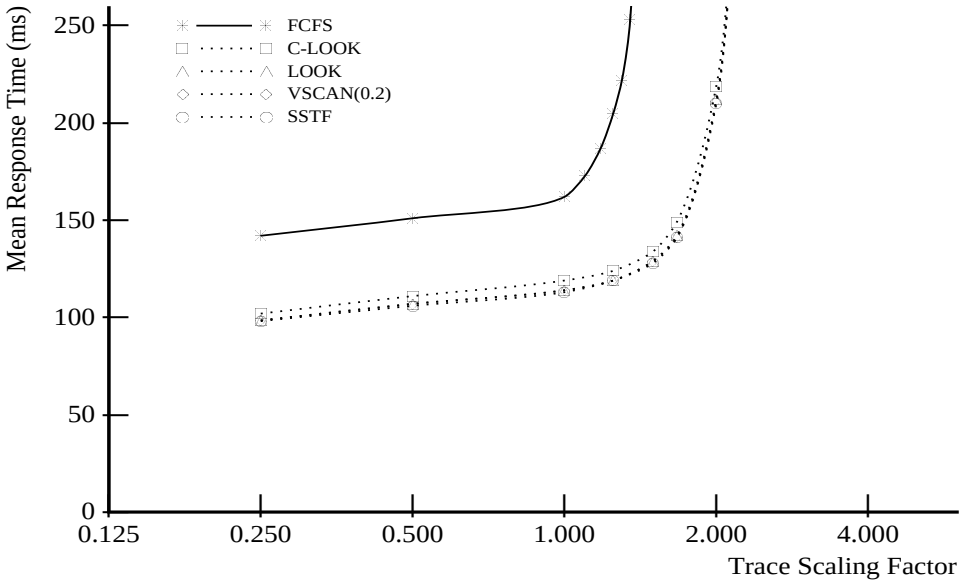


(a) Mean Response Time

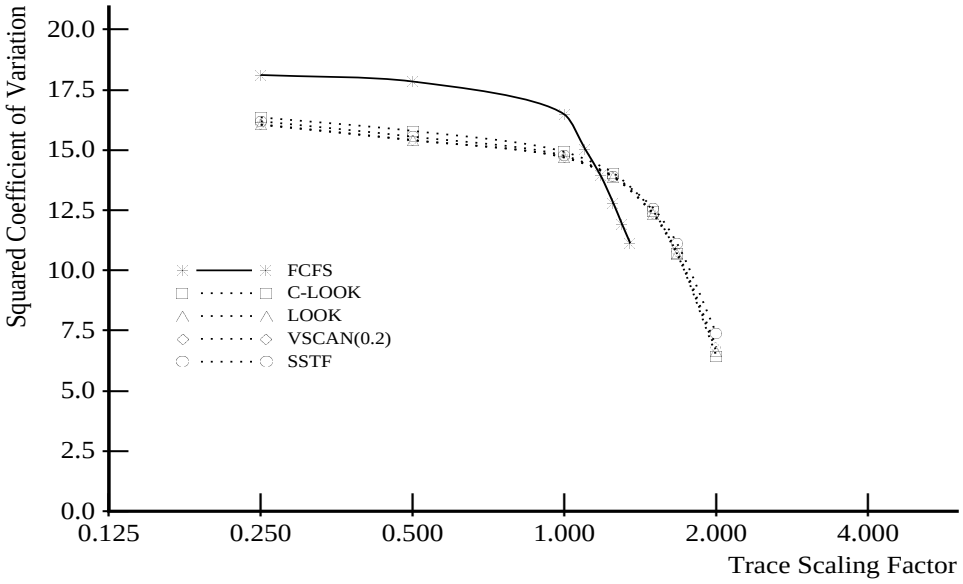


(b) Squared Coefficient of Variation

Figure 5.3: Full-Knowledge Algorithm Performance using a Synthetic Workload with Uniformly Distributed Request Starting Locations

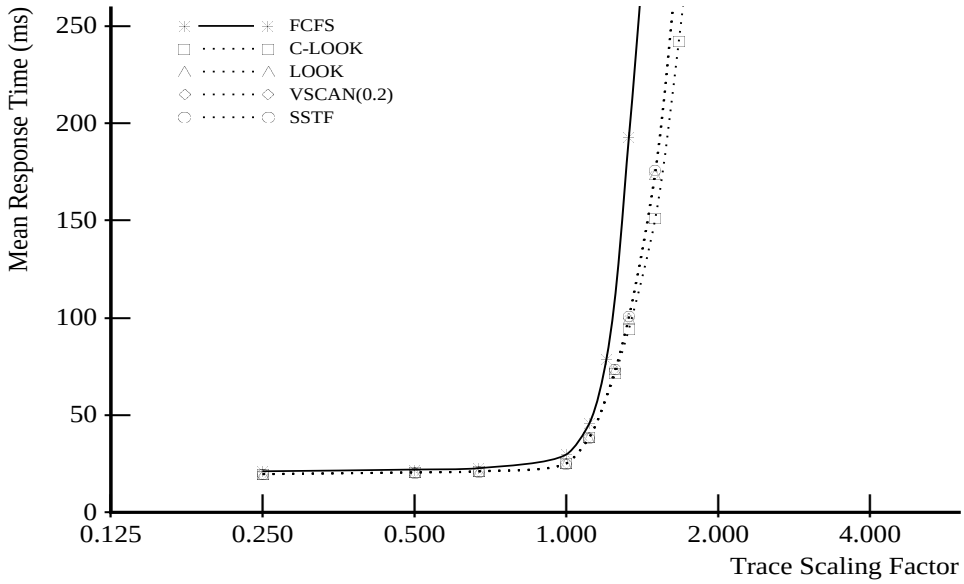


(a) Mean Response Time

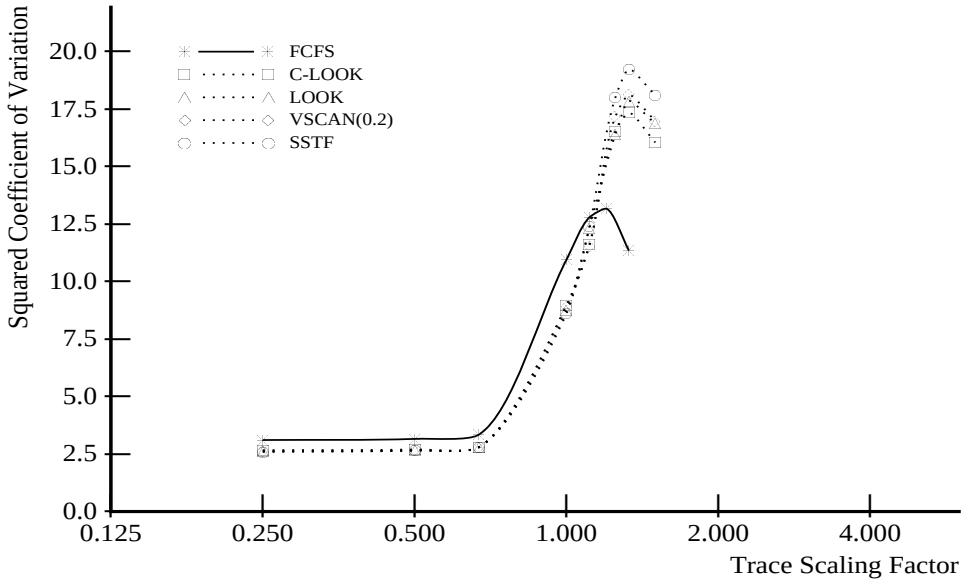


(b) Squared Coefficient of Variation

Figure 5.4: *Cello*: LBN-Based Algorithm Performance

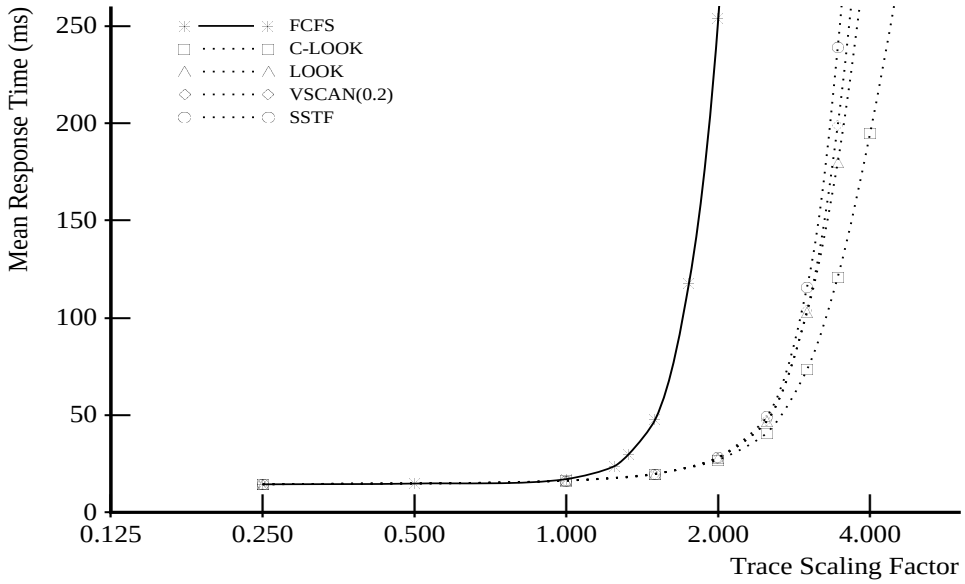


(a) Mean Response Time

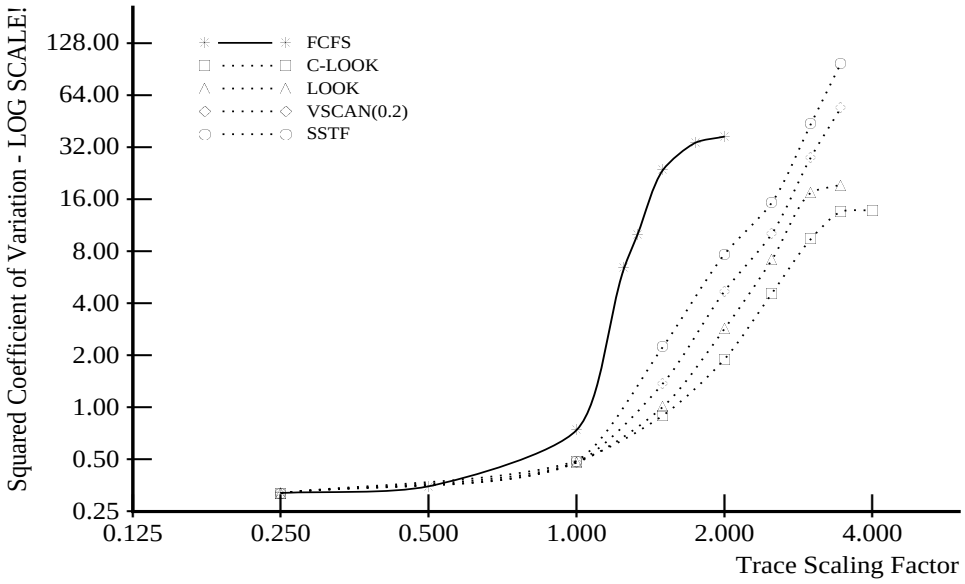


(b) Squared Coefficient of Variation

Figure 5.5: *Snake*: LBN-Based Algorithm Performance

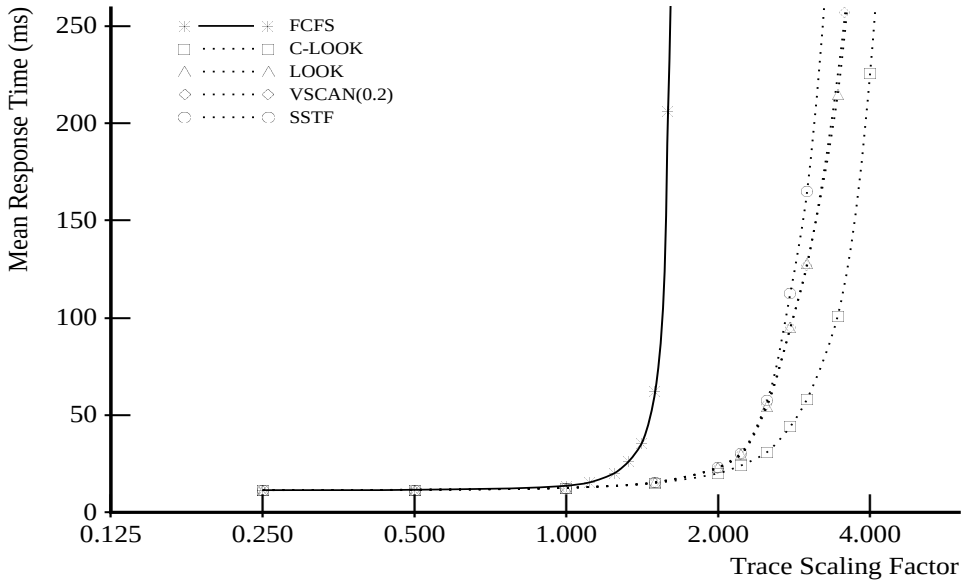


(a) Mean Response Time

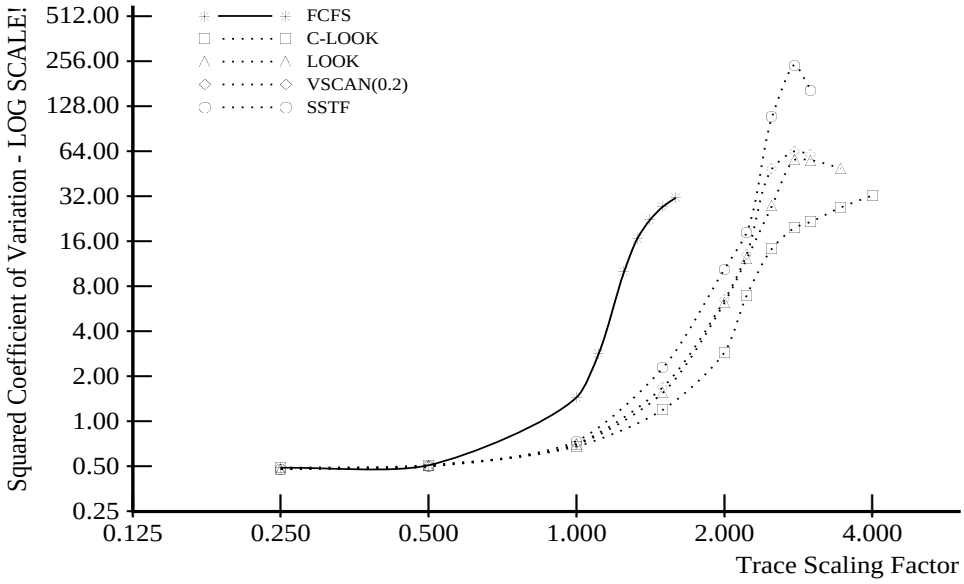


(b) Squared Coefficient of Variation

Figure 5.6: *Air-Rsv*: LBN-Based Algorithm Performance

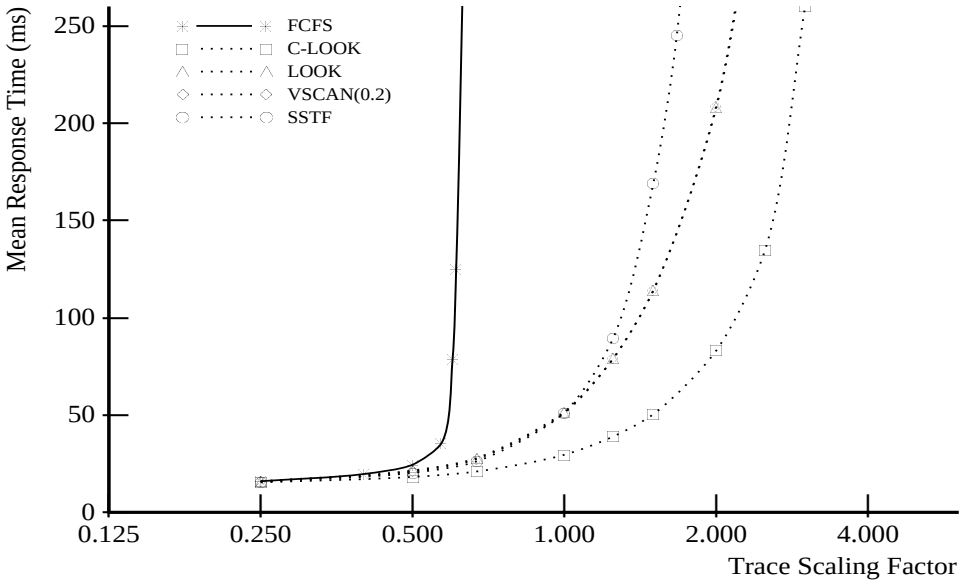


(a) Mean Response Time

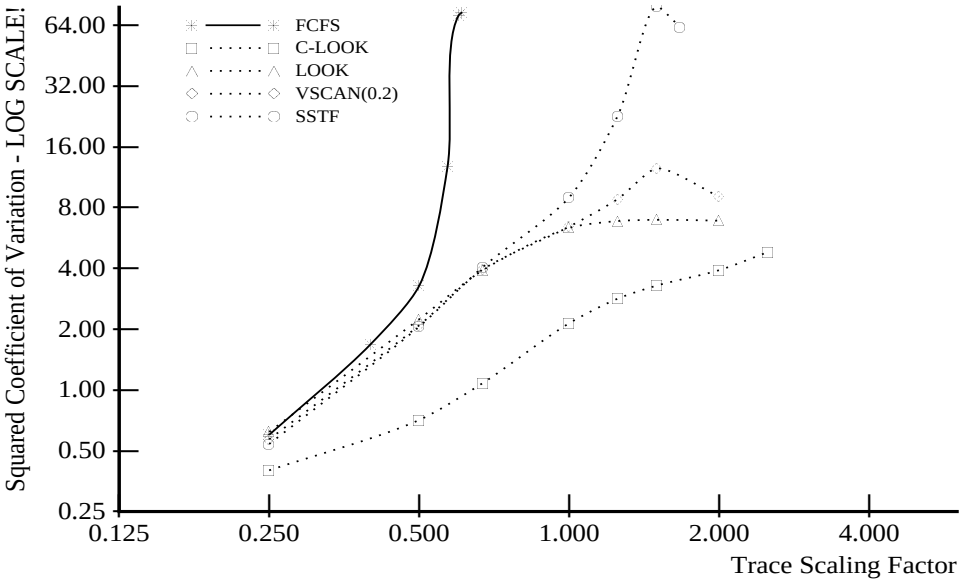


(b) Squared Coefficient of Variation

Figure 5.7: *Sci-TS*: LBN-Based Algorithm Performance

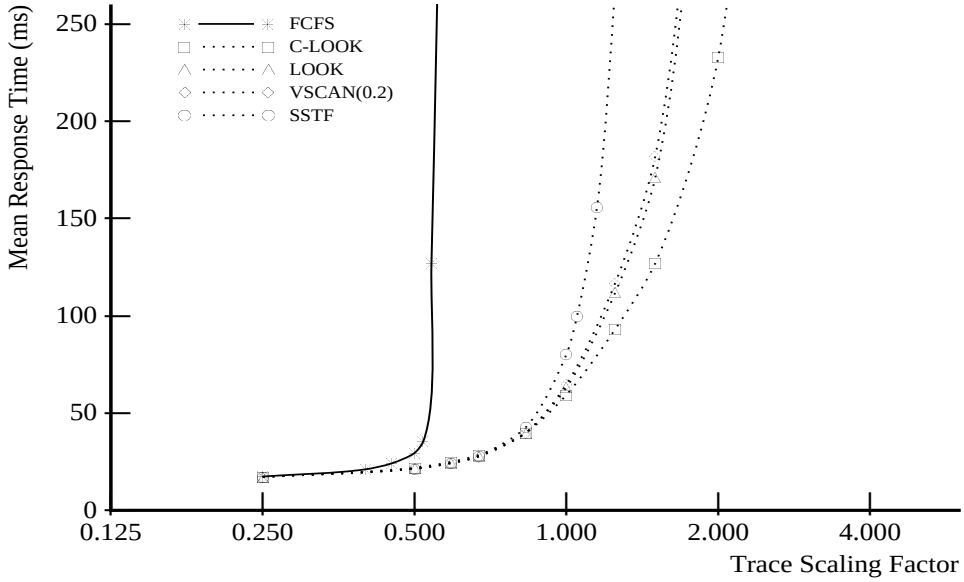


(a) Mean Response Time

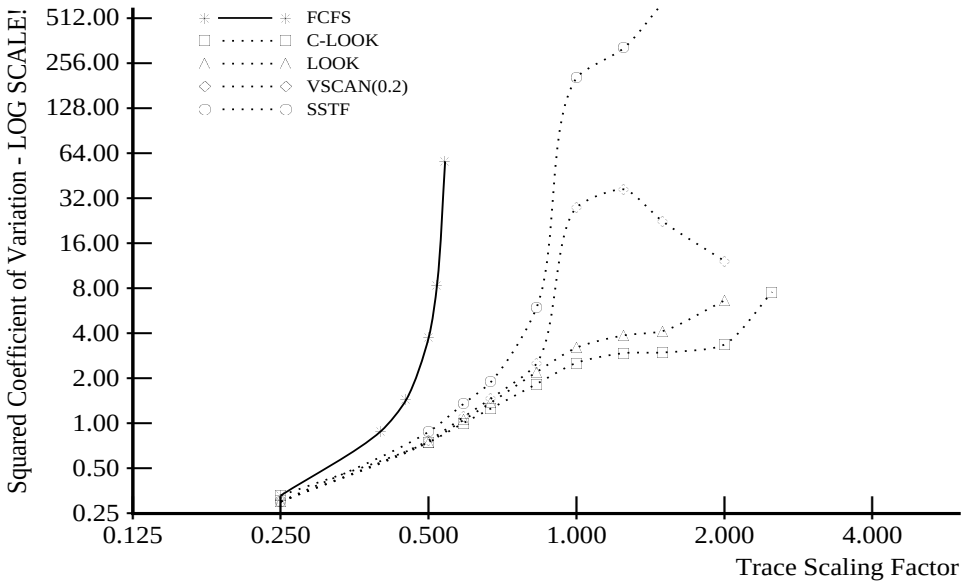


(b) Squared Coefficient of Variation

Figure 5.8: *Order*: LBN-Based Algorithm Performance



(a) Mean Response Time



(b) Squared Coefficient of Variation

Figure 5.9: *Report*: LBN-Based Algorithm Performance

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) Mean Response Times (ms) using a Prefetching On-Board Data 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) Mean Response Times (ms) using a Speed-Matching On-Board Data Buffer

Table 5.1: Sample Mean Response Times for LBN-Based Algorithms (taken from the "knees" of the Mean Response Time curves for the six traces)

the case of *Cello*, all algorithms perform equally for the lowest trace scaling factors studied. FCFS performs poorly for all but the lightest workloads, as was true for the synthetic workloads. The relative performance of the other algorithms, however, differs from that observed for the synthetic workloads. The results indicate that the HP C2247's prefetching cache plays a large role in determining algorithm effectiveness.

A disk with an on-board cache services read requests for cached data much faster than those requiring media access. If read requests arrive at a disk in logically ascending order, some fraction (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 superior performance for workloads containing a significant fraction of sequential or highly local read requests.

Table 5.1a lists a sample point from the "knee" of each trace's set of mean response time curves. 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 all sequential requests in logically ascending order during the "ascending" phase of the scan cycle. However, LOOK, VSCAN(0.2), and SSTF all exhibit similar performance when the

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 5.2: On-board Data Cache Read Hit Percentages for LBN-Based Algorithms

last of a sequential sequence of requests is scheduled first. When this occurs, the disk services the requests in logically descending order, completely negating the performance advantage of the prefetching cache.

For *Cello*, C-LOOK produces a higher mean response time than the other seek-reducing algorithms. Large bursts of write requests to a single disk dominate the *Cello* environment [Ruem93]. The */usr/spool/news* disk services almost half of the requests, 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 5.2 lists the on-board cache hit rates for each of the sample points given in table 5.1a. For all workloads except *Cello*, the C-LOOK algorithm exhibits a higher hit rate in the on-board data cache. Note that 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 lower queue times for numerous pending requests.

Additional evidence of the benefits of scheduling to exploit a prefetching cache are found by comparing tables 5.1a and 5.1b. The latter table lists sample points from experiments where the on-board "cache" functions only as a speed-matching buffer. Under this restriction, the performance of C-LOOK drops dramatically in relation to the other seek-reducing algorithms. For example, C-LOOK scheduling of the *Sci-TS* trace using a fully functioning on-board cache decreases the mean response time by more than 40% over LOOK, VSCAN(0.2), and SSTF. On the other hand, C-LOOK increases the mean response time by over 40% when the on-board memory acts as a simple buffer.

For some traces, such as *Report*, the performance advantage 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). This prefetching activity continues until satisfying a configured maximum prefetch size or

the disk receives a new request for service.[1] So, requests that 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 six traces, C-LOOK provides starvation resistance equivalent to 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.

Scheduling with a Known Mapping

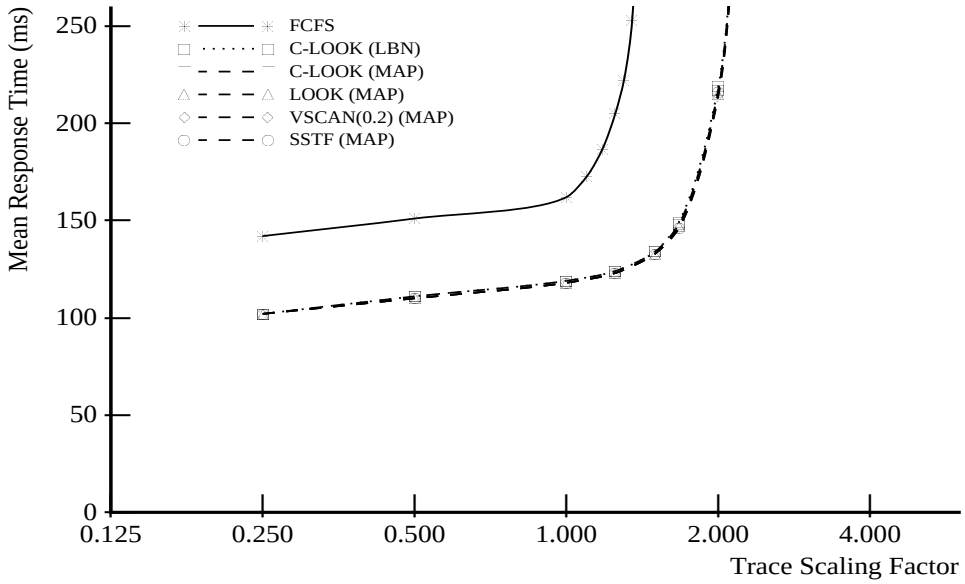
Figures 5.10–5.15 display the mean response times and squared coefficients of variation for full-map scheduling of the six traces. FCFS and the LBN-based C-LOOK algorithm are also shown for comparison purposes. Full-map scheduling improves the performance of LOOK, VSCAN(0.2), and SSTF for all traces except *Cello*. This improvement is not the result of reduced seek times via better mapping information, but rather is caused by the heuristic used to schedule requests within a single cylinder. As described in section 4.3.1, a full-map scheduler uses C-LOOK, LOOK, VSCAN(0.2), or SSTF when moving between cylinders and uses C-LOOK "within" a cylinder. Therefore, the full-map versions of LOOK, VSCAN(0.2) and SSTF use the prefetching cache much more effectively than their LBN-based counterparts.

To verify the relative importance of the C-LOOK heuristic compared to the full LBN-to-PBN mapping, the scheduling algorithms were temporarily modified to use a fixed number of sectors per cylinder (1024) rather than the actual zone-dependent values. The resulting performance for all traces and algorithms is within 1% of the performance obtained using accurate mappings. Thus, the seek-reducing algorithms obtain only a marginal benefit from accurate LBN-to-PBN mappings when scheduling the real-world traces.

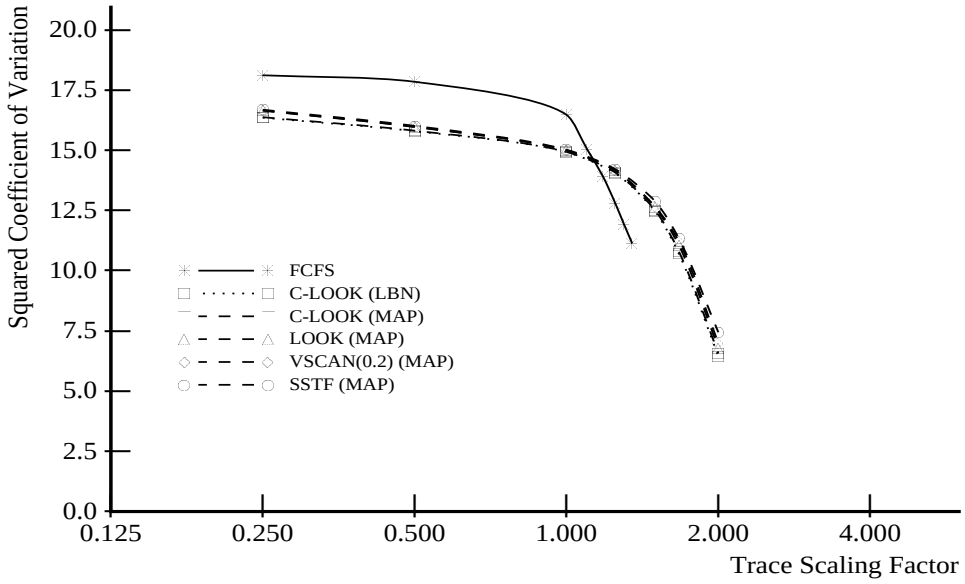
Because of the heuristic, the mean response times of full-map LOOK, VSCAN(0.2), and SSTF more closely match that of C-LOOK. For the five workloads where C-LOOK provides the best mean response time, this implies improved performance for LOOK, VSCAN(0.2), and SSTF. However, C-LOOK itself obtains only a marginal reduction in mean response time using accurate LBN-to-PBN mappings; adding the C-LOOK cylinder heuristic does little to modify the behavior of the LBN-based C-LOOK algorithm.

Full-map SSTF improves less than full-map LOOK or VSCAN(0.2) when compared to the LBN-based algorithms. In the case of *Report*, the LBN-based SSTF scheduler is actually superior to the full-map version. This is because of a slight difference in the exact implementation of the SSTF scheduler. 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 keeps the three scanning algorithms from starving overlapping requests. Full-map SSTF achieves a performance improvement equivalent to full-map LOOK and VSCAN(0.2) if it also uses

[1] When command queueing is disabled, lengthy delays often occur between reading the last requested sector from the media and receiving a new request to service. First, the disk transfers one or more requested sectors to the host (perhaps after waiting for permission to use the peripheral bus) and performs any necessary cleanup/completion processing. After acknowledging the completion message, the host obtains permission to use the bus and transfers a new request to the disk.

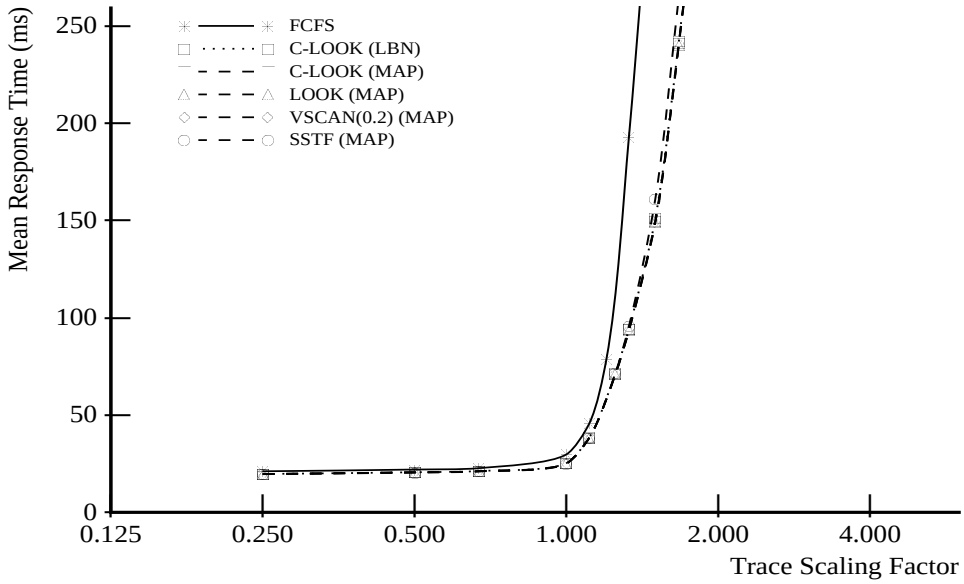


(a) Mean Response Time

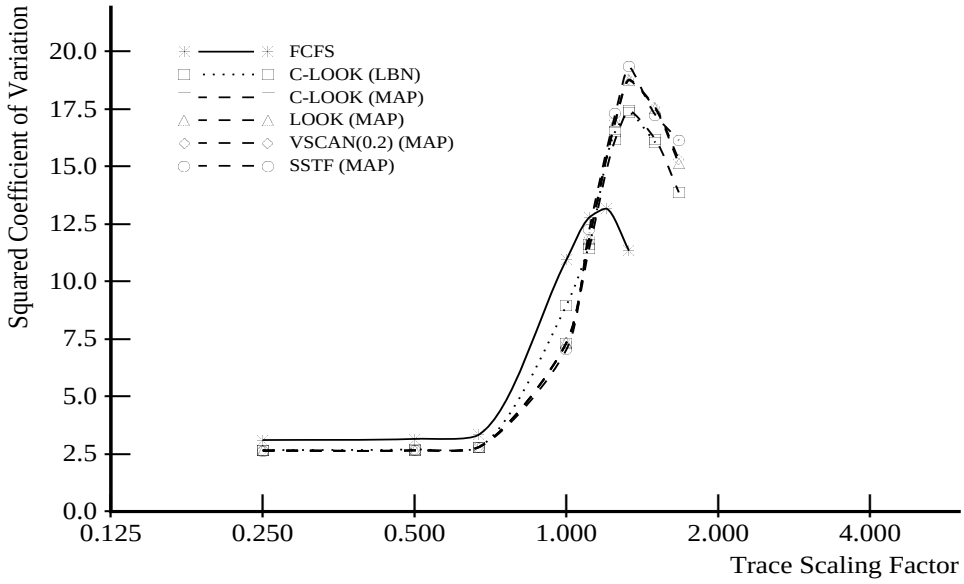


(b) Squared Coefficient of Variation

Figure 5.10: *Cello*: Full-Map and LBN-Based Algorithm Performance

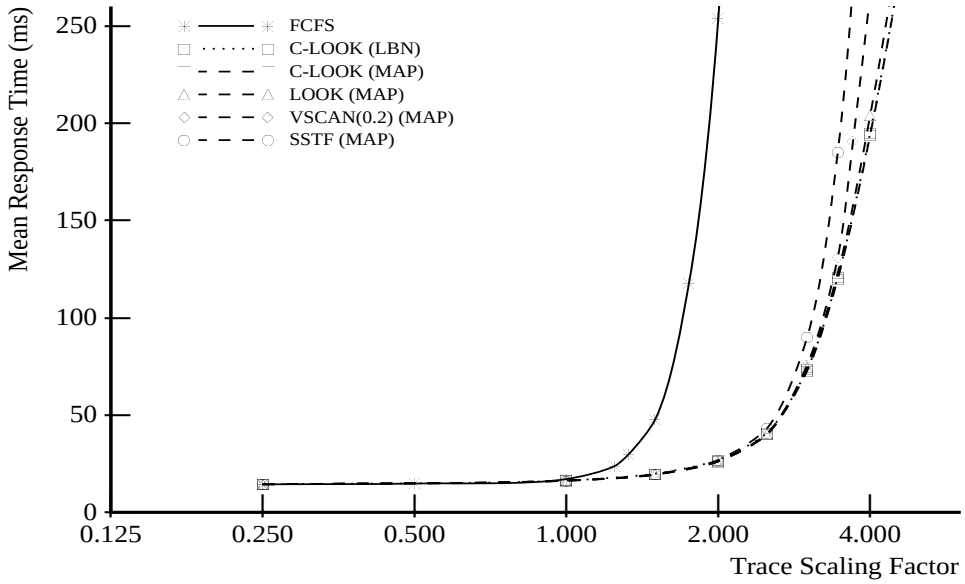


(a) Mean Response Time

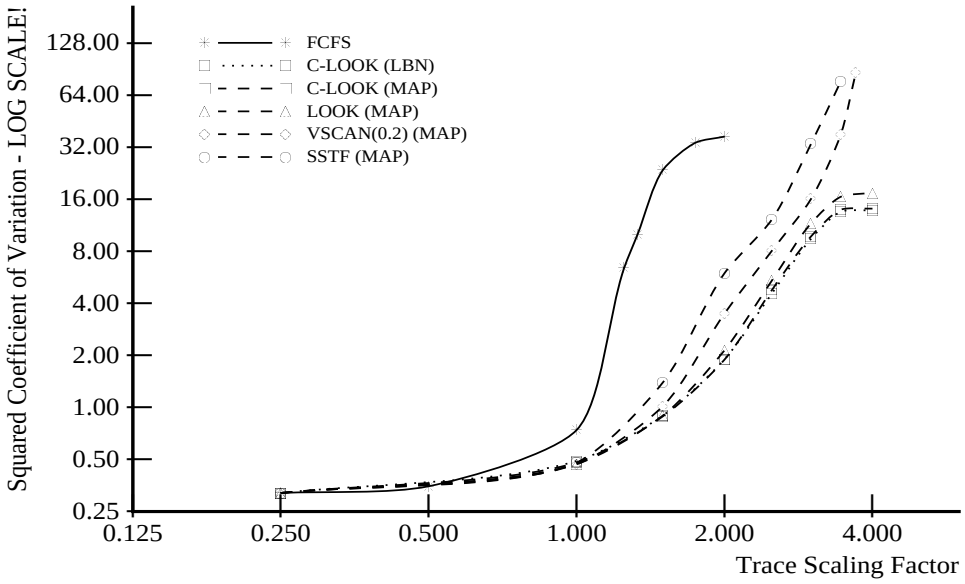


(b) Squared Coefficient of Variation

Figure 5.11: *Snake*: Full-Map and LBN-Based Algorithm Performance

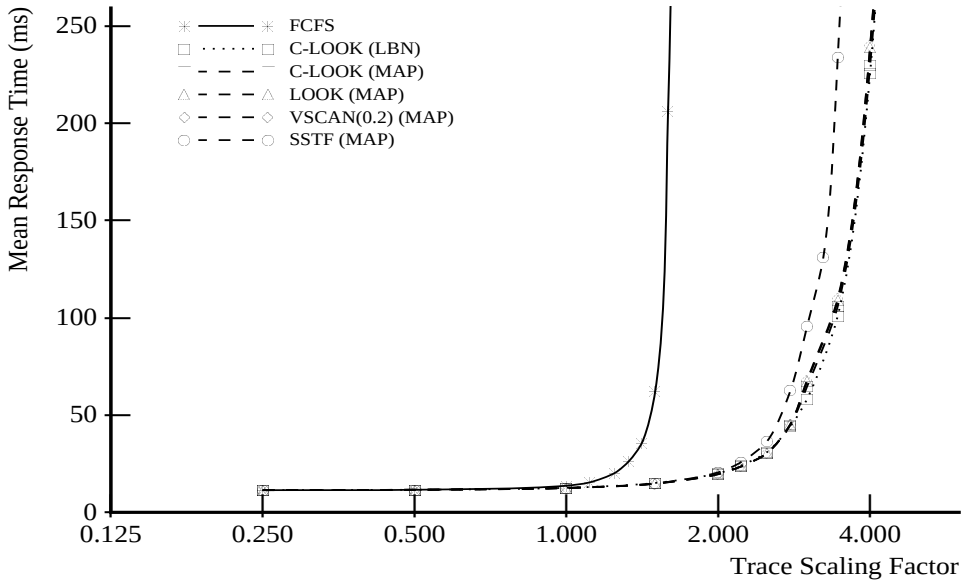


(a) Mean Response Time

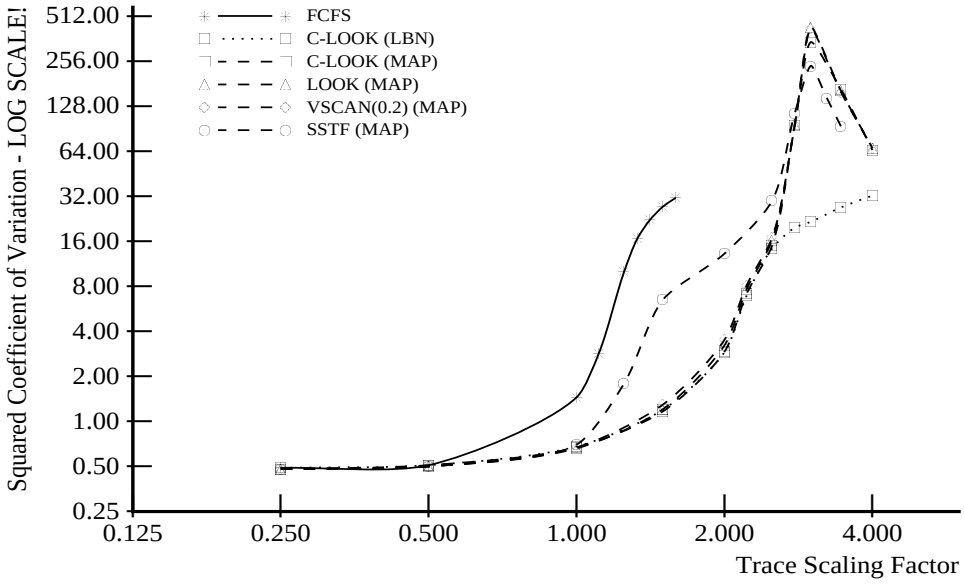


(b) Squared Coefficient of Variation

Figure 5.12: *Air-Rsv*: Full-Map and LBN-Based Algorithm Performance

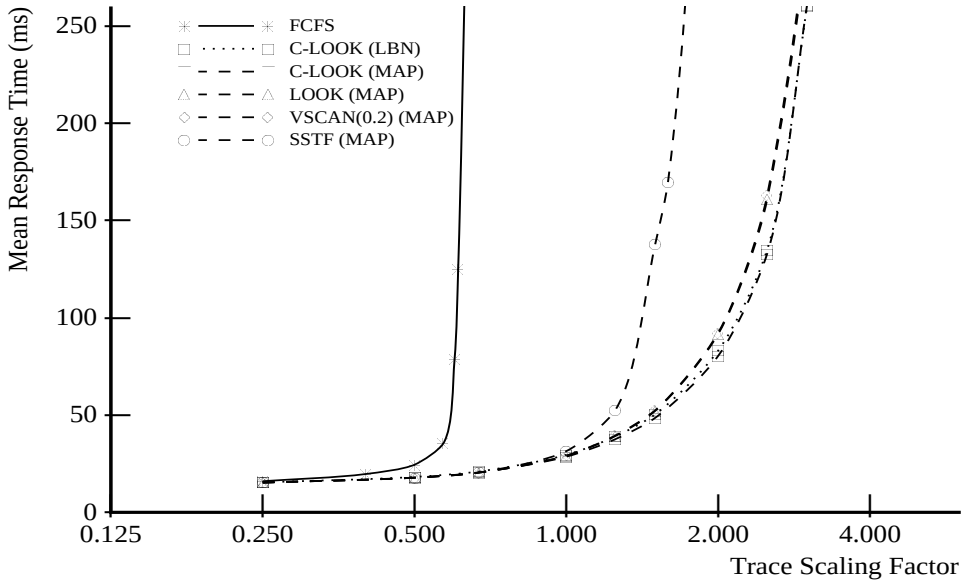


(a) Mean Response Time

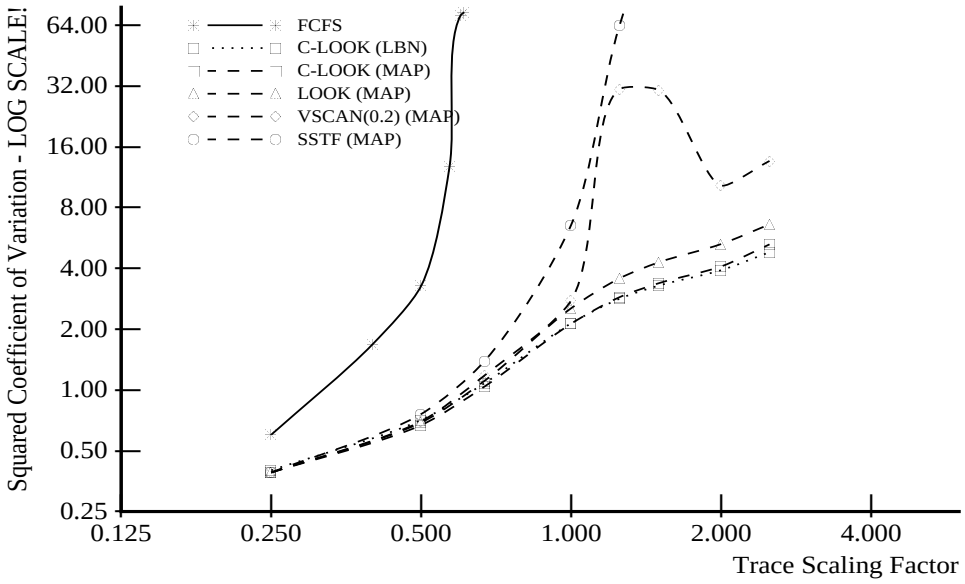


(b) Squared Coefficient of Variation

Figure 5.13: *Sci-TS*: Full-Map and LBN-Based Algorithm Performance

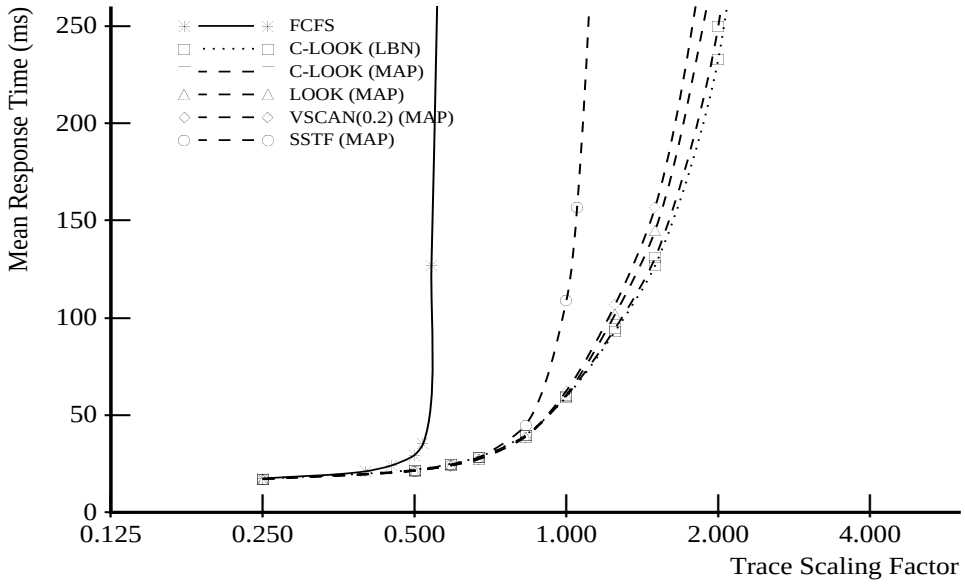


(a) Mean Response Time

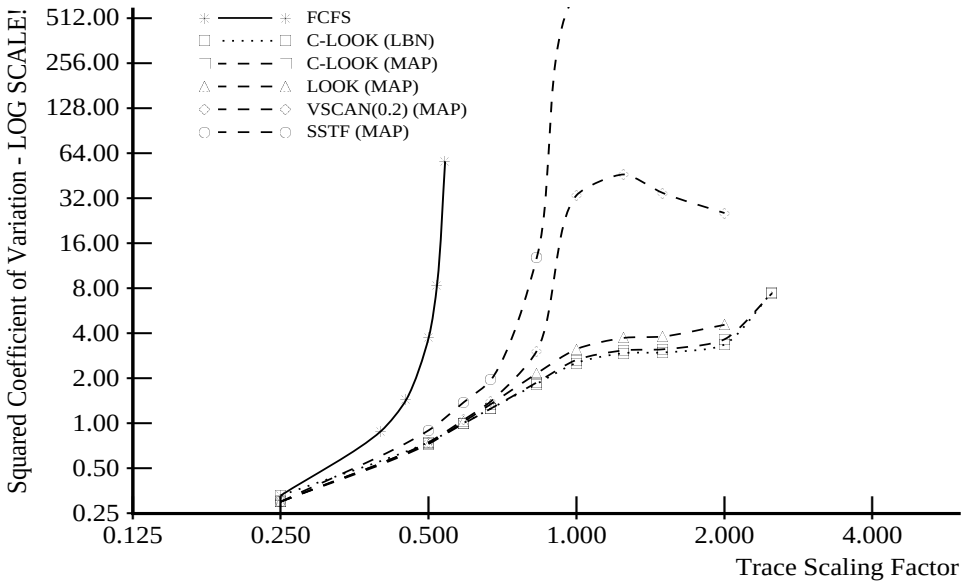


(b) Squared Coefficient of Variation

Figure 5.14: *Order*: Full-Map and LBN-Based Algorithm Performance



(a) Mean Response Time



(b) Squared Coefficient of Variation

Figure 5.15: *Report*: Full-Map and LBN-Based Algorithm Performance

the first block of the previous request. This indicates that overlapping requests exist in the experimental workloads. Some fraction of these requests are directly attributable to the chosen methodology (i.e., scaling of traced interarrival times).

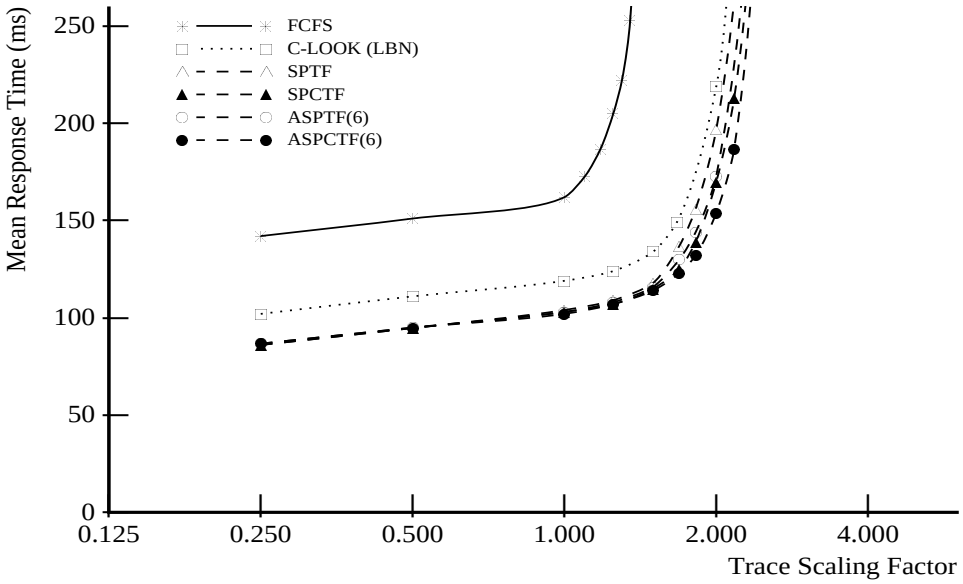
For *Cello*, LOOK, VSCAN(0.2), and SSTF decrease in performance when using a full-map scheduler. Since large bursts of write activity dominate this trace, it is write request performance that determines algorithm effectiveness (as measured by subsystem metrics). For sequential writes to a disk without write prebuffering (described in section 4.1.2), issuing requests in logically ascending order can actually degrade performance. When a write request completes, a disk sends a completion message back to the host, waits for the next request, and then performs the necessary command initiation processing. At the same time, the host processes the completion message, issues the next (sequential) write request, and begins transferring data blocks to the on-board data cache for subsequent transfer to the disk media. During these steps, the active read/write head rotates beyond the first sector of the next request. Assuming that the completion and initiation processing takes only a fraction of a rotation, the new request incurs almost a full rotation of latency.

The relative starvation characteristics of the full-map scheduling algorithms generally match those of the LBN-based algorithms. 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 squared coefficient of variation than C-LOOK for the heaviest workloads (although its response time variance is still greater). By satisfying all requests within a cylinder before moving to another cylinder, the full-map algorithms are slightly less resistant to starvation.

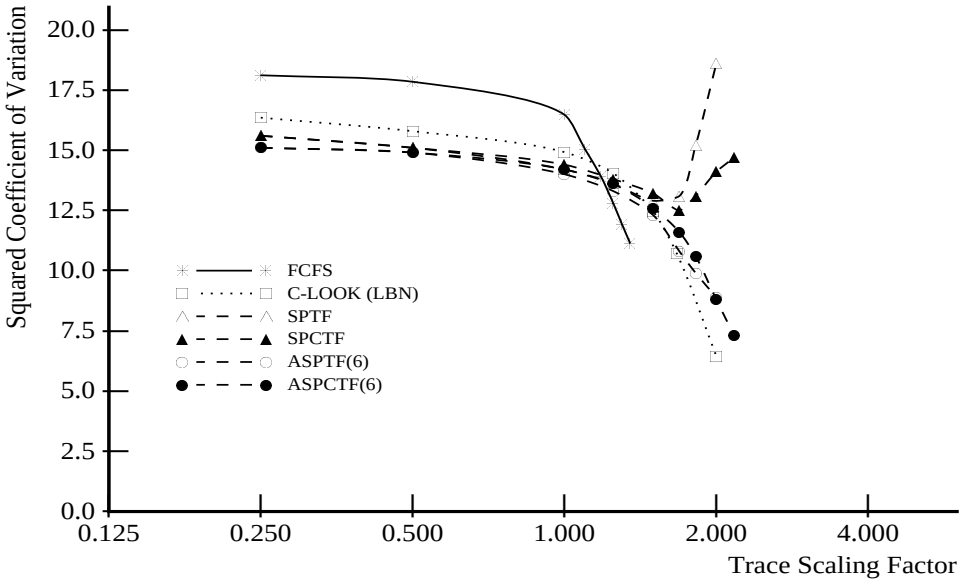
Scheduling with Full Knowledge

Figures 5.16–5.21 show results for experiments using the SPTF-based algorithms described in section 4.3.1. These algorithms have both an advantage and a disadvantage over C-LOOK. The advantage is the obvious improvement in positioning time prediction accuracy due to the inclusion of rotational latency information. A full rotation for an HP C2240 series disk takes approximately 11 ms, equivalent to a seek of over 550 cylinders (or 27% of the maximum seek distance). For workloads with a skewed distribution of request starting locations (i.e., with shorter average seek distances), rotational latency represents an even larger fraction of the total positioning time. The disadvantage is that SPTF-based algorithms have no inherent bias towards servicing sequential reads in logically ascending order. An SPTF scheduler selects requests based on their predicted positioning time, not their LBN or cylinder. Even SPCTF only services subsets of a sequential stream in logically ascending order. As a result, the SPTF-based algorithms utilize prefetching caches less effectively. Table 5.3 lists the cache hit percentages for some sample trace scaling factors. Note that C-LOOK always exhibits a higher cache hit rate.

This tradeoff affects overall performance to different degrees for each of the six traces. For *Cello*, the SPTF-based algorithms are clearly superior to the seek-reducing algorithms. Cache knowledge increases performance an additional 5–15% for the largest scaling factors. In experiments using the other five traces, the comparison is not as straightforward. SPTF saturates more quickly than C-LOOK, but SPTF provides superior performance for some sub-saturation workloads. For example, figure 5.21a shows that SPTF is up to 18% better

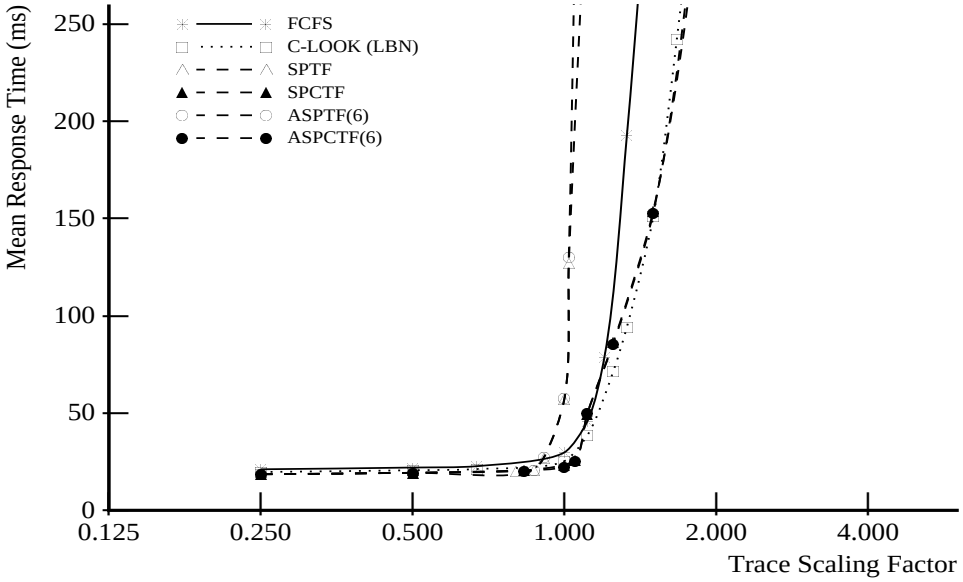


(a) Mean Response Time

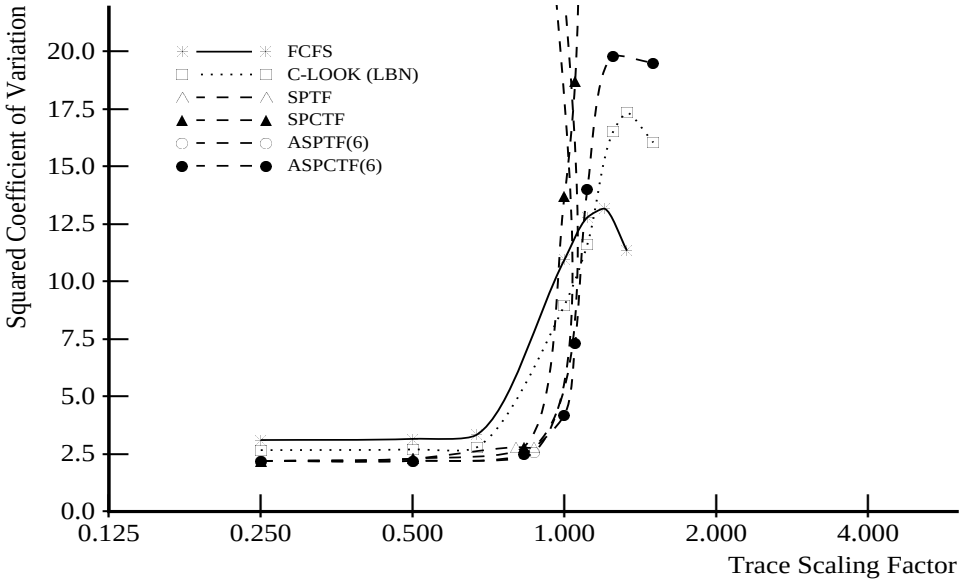


(b) Squared Coefficient of Variation

Figure 5.16: *Cello*: Full-Knowledge Algorithm Performance

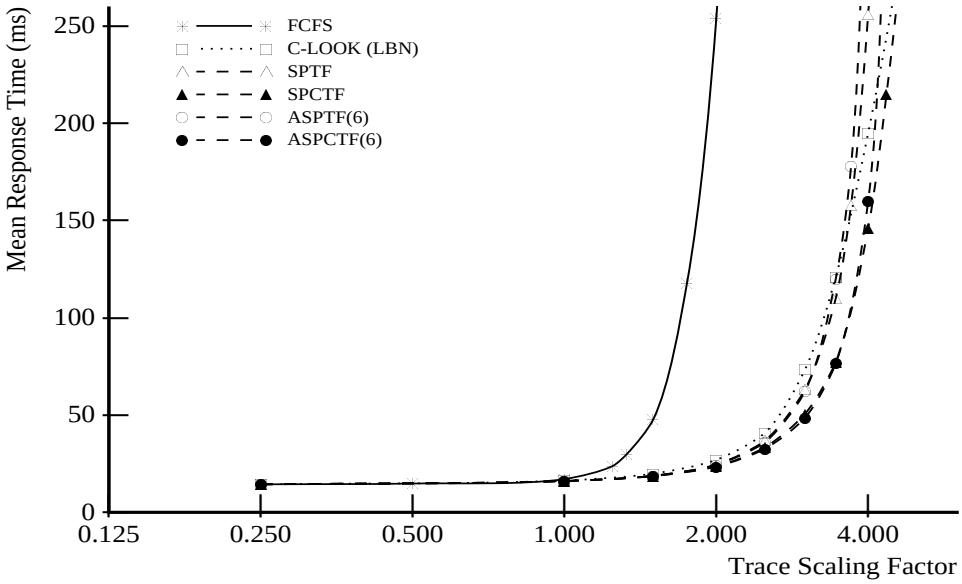


(a) Mean Response Time

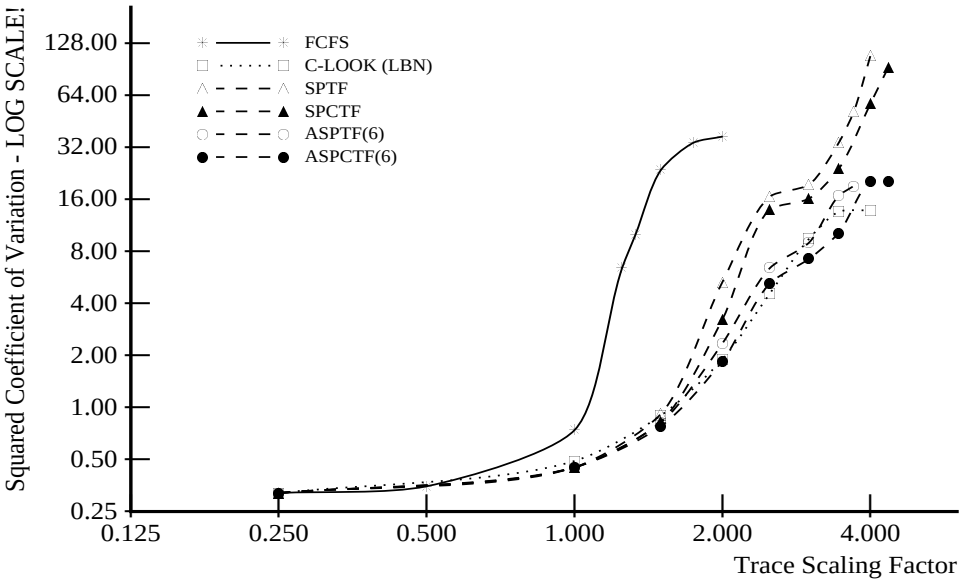


(b) Squared Coefficient of Variation

Figure 5.17: *Snake*: Full-Knowledge Algorithm Performance

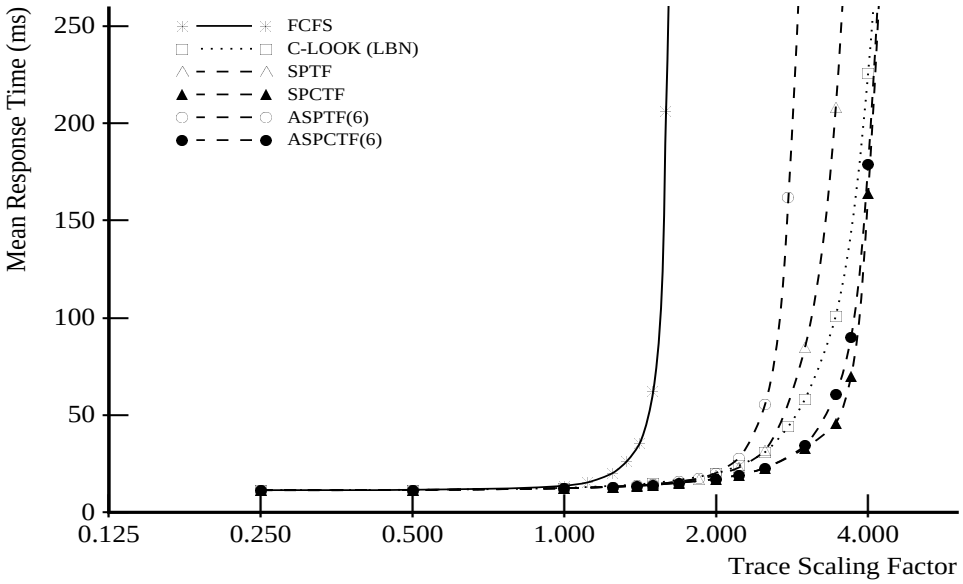


(a) Mean Response Time

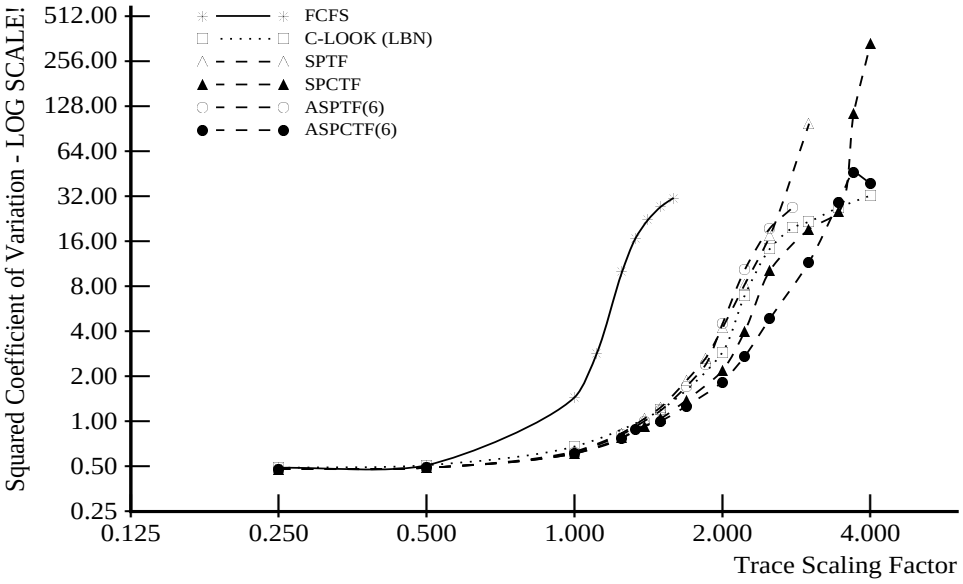


(b) Squared Coefficient of Variation

Figure 5.18: *Air-Rsv*: Full-Knowledge Algorithm Performance

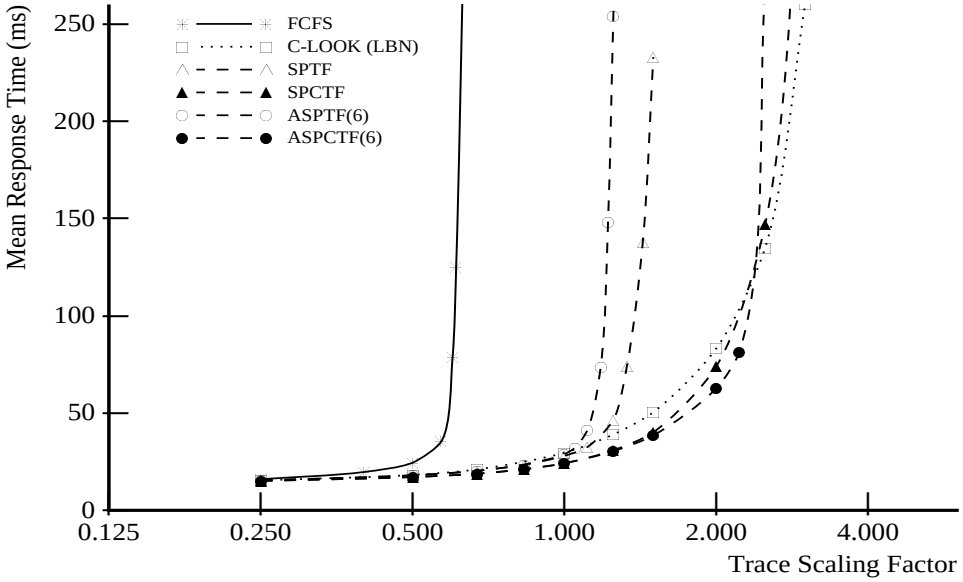


(a) Mean Response Time

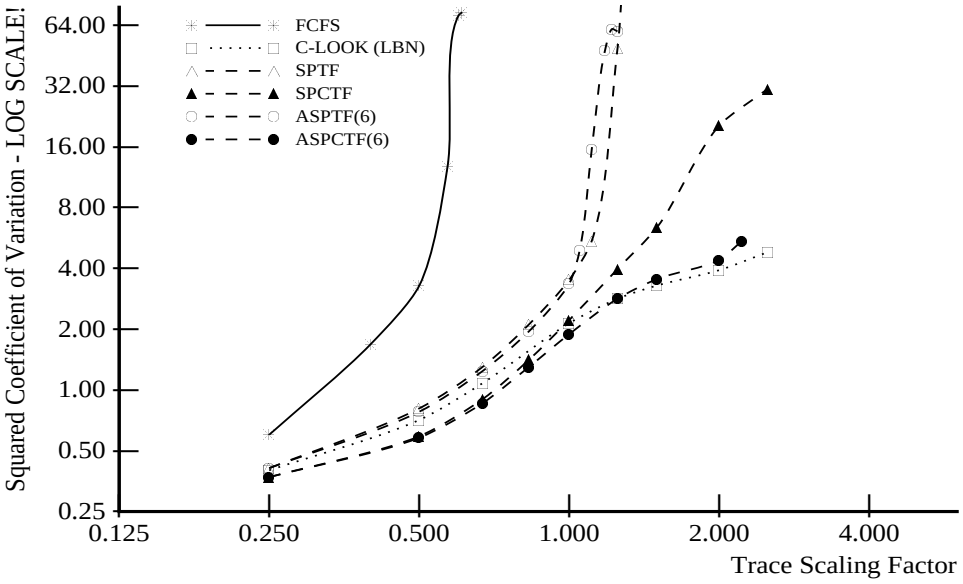


(b) Squared Coefficient of Variation

Figure 5.19: *Sci-TS*: Full-Knowledge Algorithm Performance

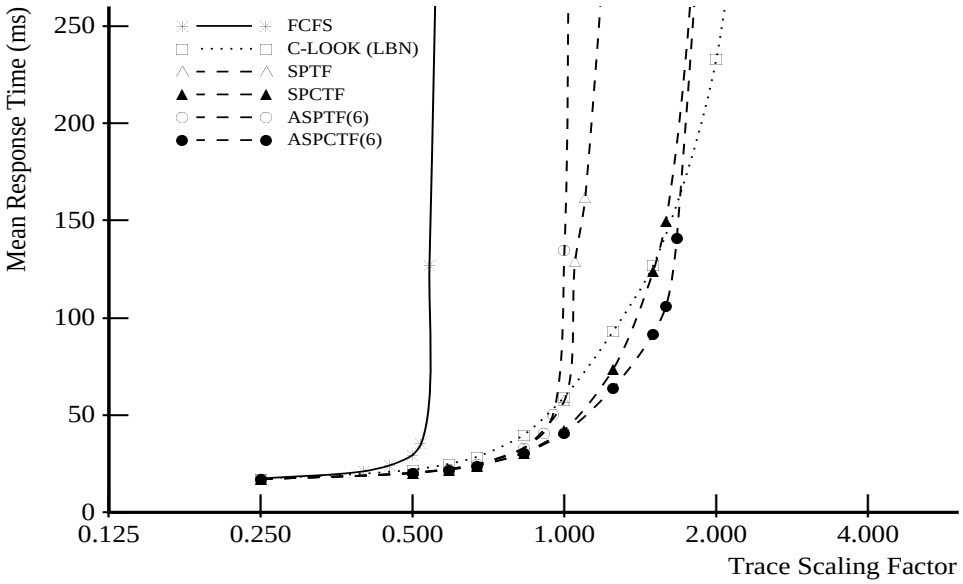


(a) Mean Response Time

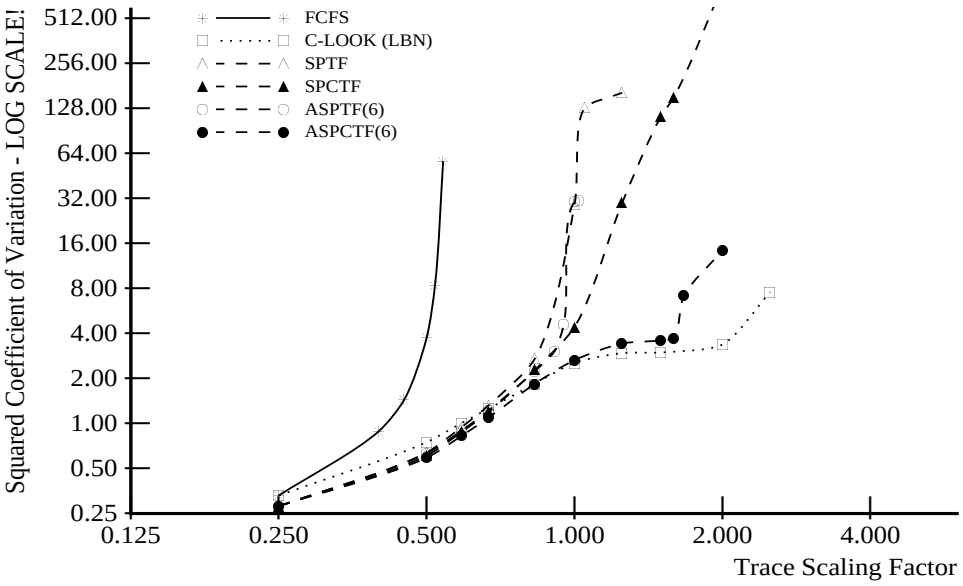


(b) Squared Coefficient of Variation

Figure 5.20: *Order*: Full-Knowledge Algorithm Performance



(a) Mean Response Time



(b) Squared Coefficient of Variation

Figure 5.21: *Report*: Full-Knowledge Algorithm Performance

Trace	Scale Factor	LBN-Based C-LOOK	SPCTF	ASPCTF(6)	SPTF	ASPTF(6)
Cello	1.0	11.5	11.5	11.5	11.2	11.2
Snake	1.25	21.8	21.2	21.2	17.4	17.4
Air-Rsv	2.5	15.0	14.2	14.1	13.2	13.1
Sci-TS	2.5	26.9	25.4	25.3	23.0	22.5
Order	1.0	17.6	16.2	16.0	11.0	10.9
Report	1.0	14.5	11.7	11.6	10.2	9.6

Table 5.3: On-board Data Cache Read Hit Percentages for Full-Knowledge Algorithms

than C-LOOK in the 0.4–0.9 range of scaling factors for *Report*. For SPCTF, saturation does not occur as quickly. Incorporating cache knowledge widens the window where SPTF-based algorithms are superior to C-LOOK. In the case of *Sci-TS*, SPCTF provides increased performance for all scaling factors studied.

SPTF scheduling of the *Snake* trace exhibits interesting behavior. Without cache knowledge, SPTF-based algorithms do a very poor job of scheduling the occasional bursts of sequential reads that dominate the performance metrics for this trace. During a sequential read burst, SPTF rarely chooses to service requests in logically ascending order; by the time the data transfer and cleanup overheads complete for one read request, the read/write head has rotated past the first sector of the next sequential read request. Thus, the scheduler will predict almost a full rotation of positioning delay for that request. As the queue of pending requests grows, the scheduler will almost always be able to locate another request whose data can be positioned over more quickly. Figure 5.17a shows that SPTF and ASPTF(6) saturate before all other algorithms, including FCFS. Since the bursts of sequential read requests in *Snake* arrive in logically ascending order, the seek-reducing algorithms and the cache-sensitive SPTF algorithms service requests in FCFS order (for the duration of the burst).

The performance of the age-sensitive SPTF algorithms (i.e., ASPTF(6) and ASPCTF(6)) are analyzed in section 5.2.1, but it is worth noting that the aging factor can improve performance as well as starvation resistance in some cases. It does this by maintaining the ordering of sequential requests (if the scheduler receives them in logically ascending order).

Sequential Stream Optimizations

The above experiments were repeated to study the effects of concatenation and sequential scheduling of sequential request streams. Since scheduling individual sequential writes to a disk without write prebuffering degrades performance (see page 51), this particular set of experiments does not consider sequential scheduling of write requests.

Figures 5.22–5.27 show results for LBN-based and full-knowledge scheduling algorithms using the sequential stream optimizations. Each bargraph displays mean response times for four versions of each algorithm at a trace scaling factor along the "knee" of the set of response time curves. C-LOOK maintains the lowest mean response time among the seek-reducing

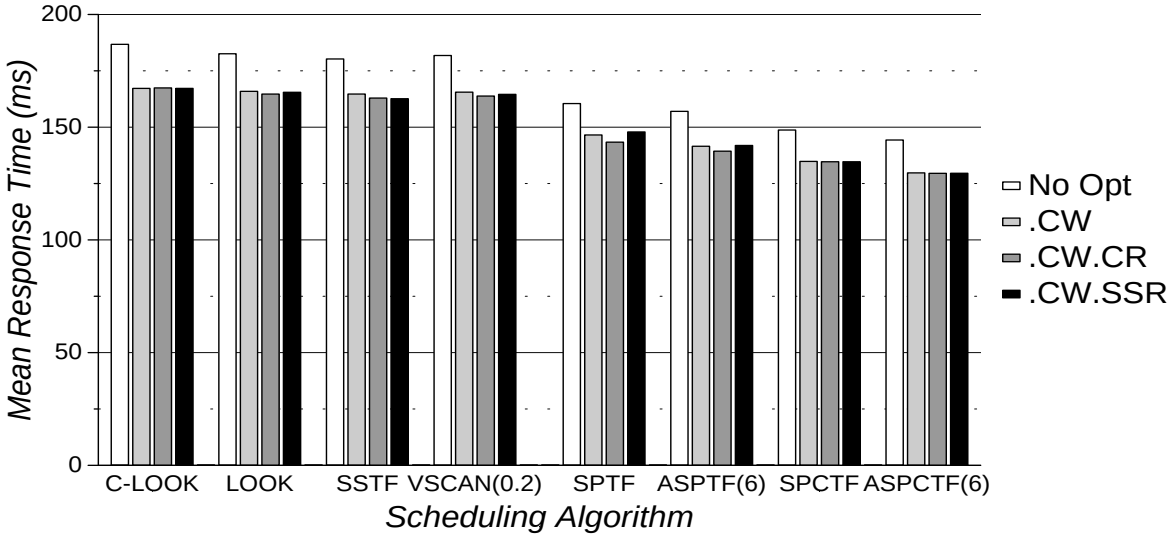


Figure 5.22: *Cello*, 1.75X: Sequential Stream Algorithm Performance

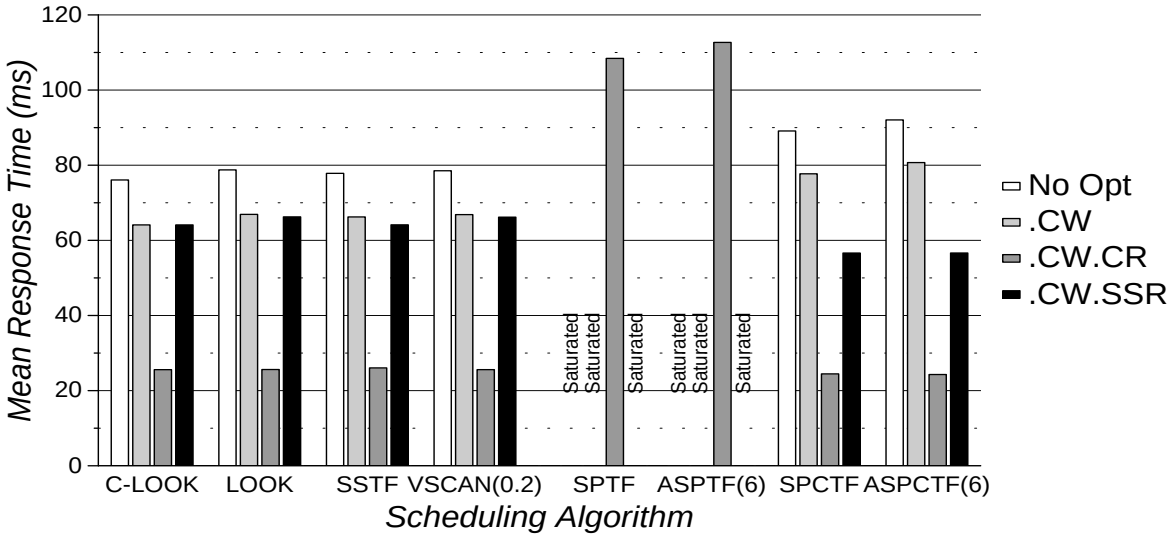


Figure 5.23: *Snake*, 1.25X: Sequential Stream Algorithm Performance

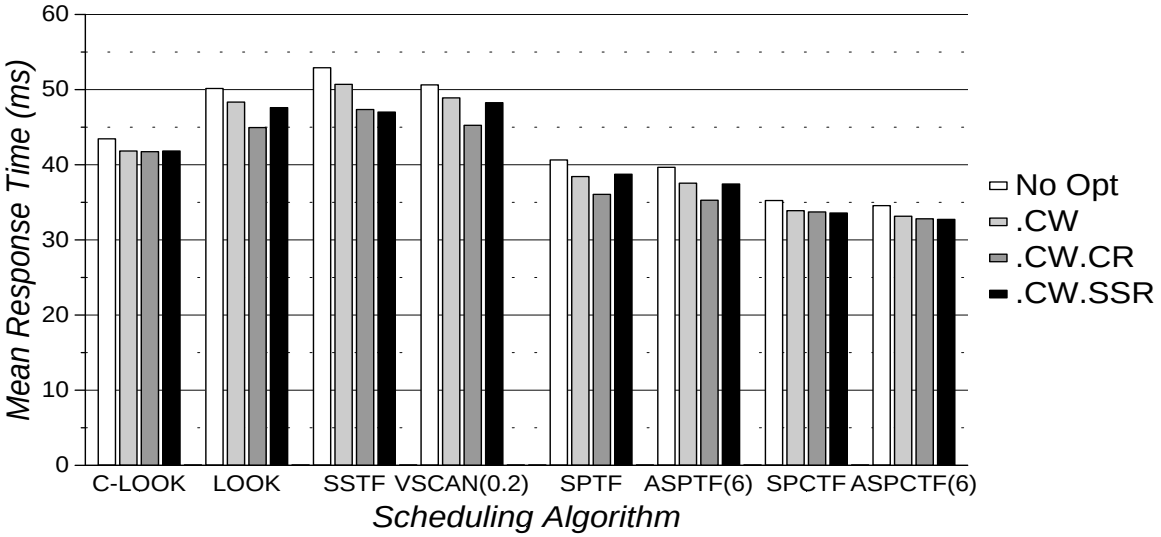


Figure 5.24: *Air-Rsv*, 2.5X: Sequential Stream Algorithm Performance

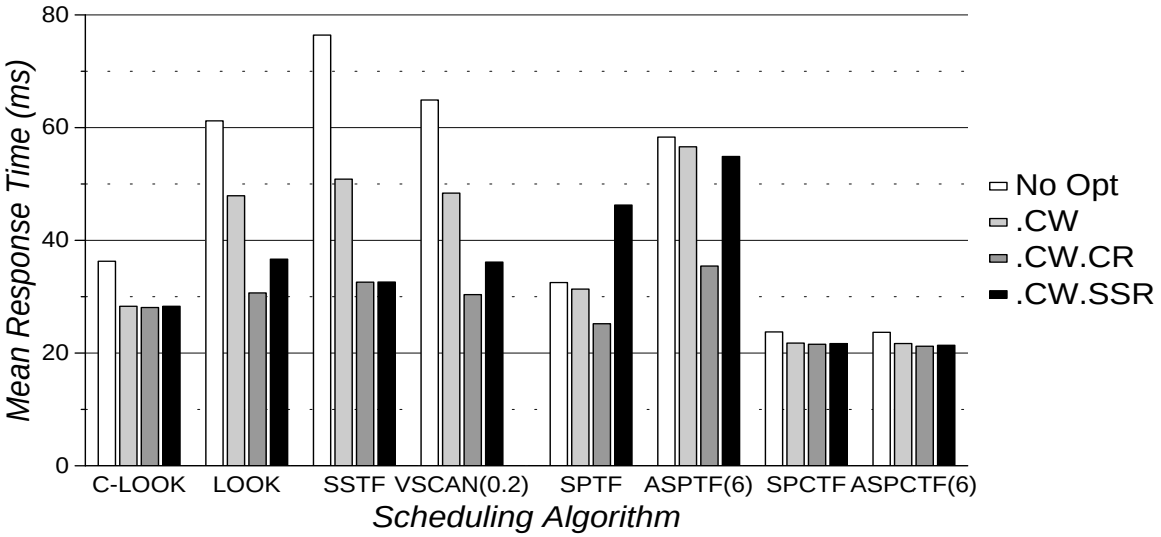


Figure 5.25: *Sci-TS*, 2.5X: Sequential Stream Algorithm Performance

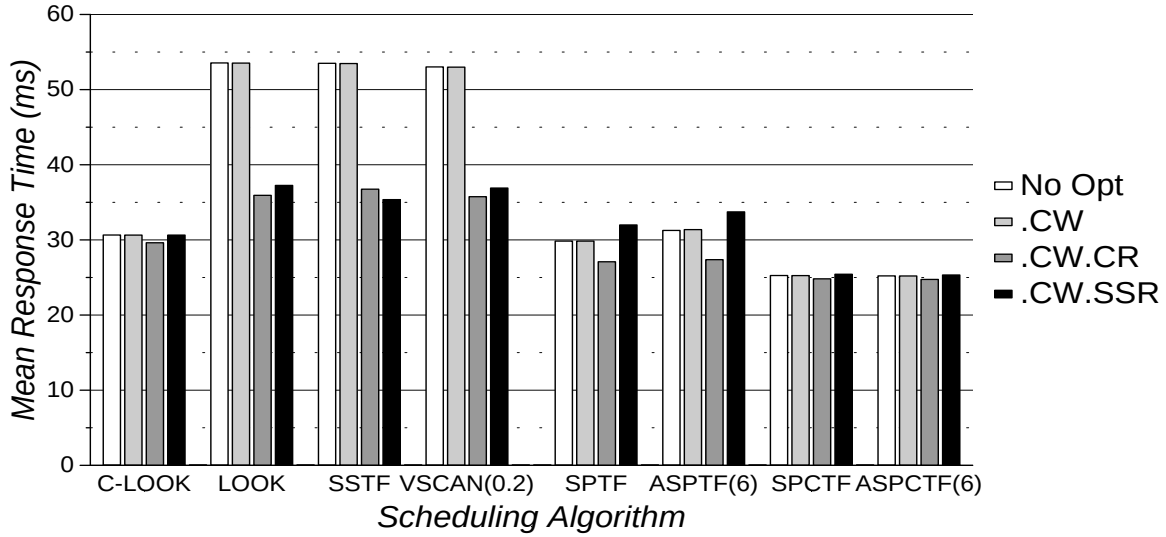


Figure 5.26: *Order*, 1.0X: Sequential Stream Algorithm Performance

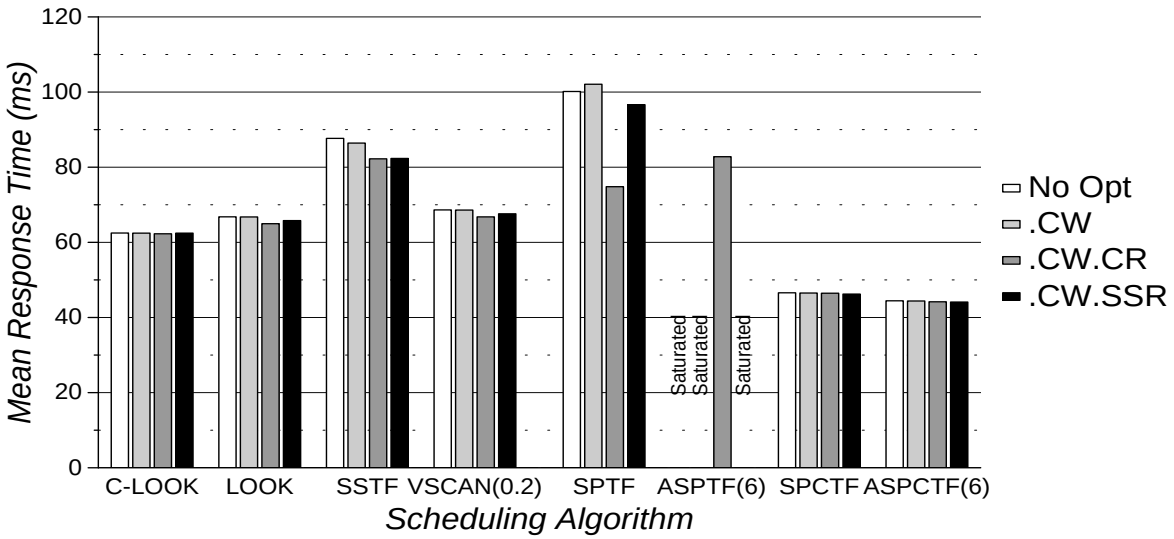


Figure 5.27: *Report*, 1.0X: Sequential Stream Algorithm Performance

algorithms for all traces except *Cello*. As in previous experiments, all other seek-reducing algorithms outperform C-LOOK for this trace. For example, SSTF has a 1–2% lower mean response time than C-LOOK after enabling concatenation of sequential write requests.

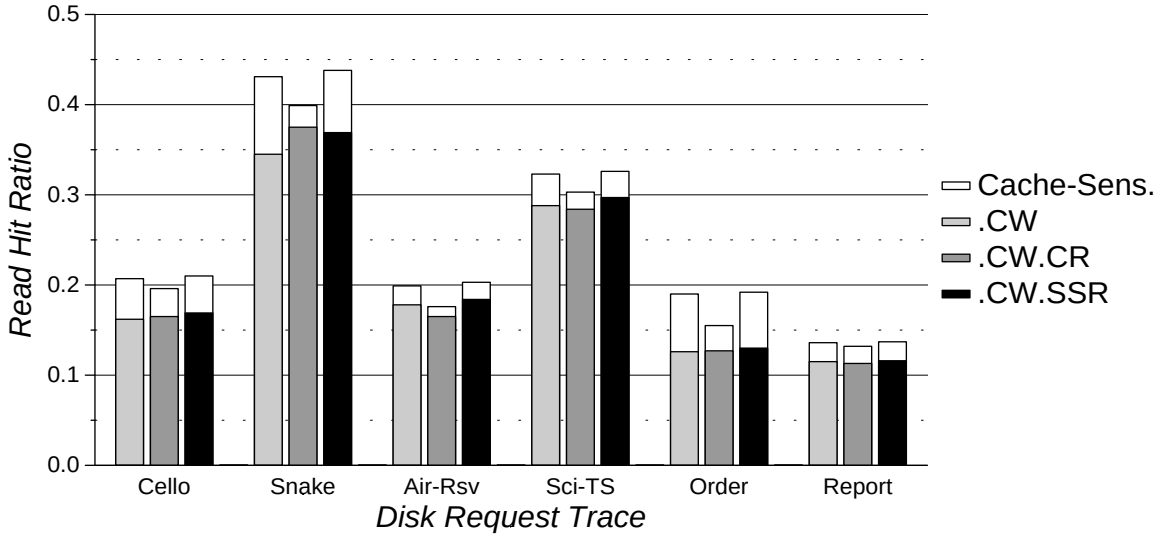
For *Air-Rsv*, *Sci-TS*, *Order*, and *Report*, the sequential scheduling optimizations affect the performance of LOOK, SSTF, and VSCAN(0.2) in a different fashion than C-LOOK. A discussion of the implementation details for sequential scheduling of these algorithms should help explain this behavior. Without the sequential scheduling optimization, these three algorithms frequently schedule streams of sequential requests in logically descending order — a highly undesirable behavior for disks with prefetching caches. LOOK always services requests in logically descending order during the "descending" phase of its scan cycle. During the "descending" phase of VSCAN(0.2), it treats a sequential stream the same as LOOK as long as no additional stream requests arrive once it has begun servicing the stream. SSTF behaves the same as VSCAN(0.2) if it approaches from logically "above" the stream.

The three algorithms differ in how they handle stream requests that arrive after the service of the stream has begun. LOOK ignores the requests until its next "ascending" phase. VSCAN(0.2) handles the new requests (in logically ascending order) after it completes the previous stream requests as long as there aren't any requests in the logical space just "below" the stream (i.e., within a 0.2 fraction of the logical space). SSTF almost always handles the new requests immediately after servicing the previous stream requests.

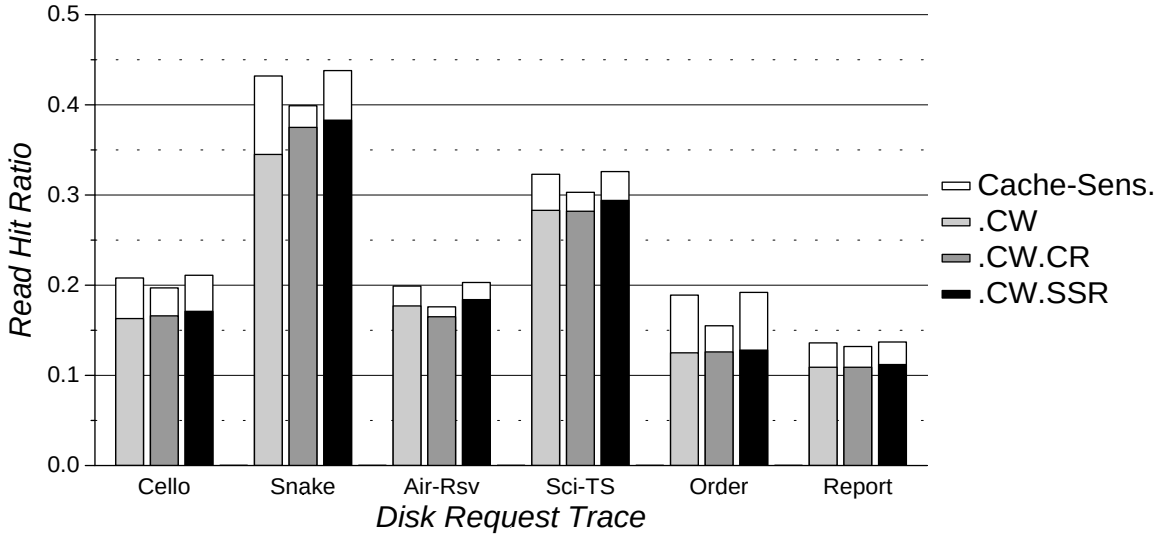
Therefore, SSTF with sequential scheduling services most sequential streams in logically ascending order. VSCAN(0.2) with sequential scheduling does so (during its "descending" phase) as long as there aren't any requests in the logical space just "below" the stream. Otherwise, it services those requests instead of continuing forward along the stream. LOOK with sequential scheduling, on the other hand, services just the first request in a sequential write (or read) stream during its "descending" phase, returning to service the rest of the stream during a subsequent "ascending" phase. Thus, all three algorithms benefit from sequential scheduling optimizations, but LOOK and VSCAN(0.2) still suffer from non-optimal servicing of sequential streams. Alternate sequential stream optimizations for LOOK and VSCAN(0.2) may avoid this particular pitfall, but the potential benefits of augmenting such scheduling algorithms are unclear when C-LOOK — an extremely simple algorithm to implement — already exhibits the desired behavior.

Looking at the full-knowledge algorithms, SPCTF and ASPCTF(6) outperform SPTF and ASPTF(6), respectively, across all six workloads. For the selected trace scaling factors, cache knowledge alleviates saturation of the disk subsystem for ASPTF(6) scheduling of *Report* and for both SPTF and ASPTF(6) scheduling of *Snake* (see figures 5.23 and 5.27). Cache-sensitive schedulers achieve lower mean response times by increasing the on-board cache hit rates. Figure 5.28 presents the improvement in read hit rates due to cache knowledge. Adding cache knowledge to an SPTF-based algorithm improves the read hit ratio by up to 50% for the selected trace scaling factors.

The remaining analysis in this section focuses on C-LOOK, SSTF, SPCTF, and ASPCTF(6), as representatives of the seek-reducing and SPTF-based scheduling algorithms. For four of the six traces, concatenation of sequential write requests improves the performance of host-based scheduling. The mean response times for *Cello*, *Snake*, *Air-Rsv*, and



(a) SPTF and SPCTF Read Hit Ratios



(b) ASPTF(6) and ASPCTF(6) Read Hit Ratios

Figure 5.28: Improvement in Read Hit Ratios of Full-Knowledge Scheduling Algorithms due to Cache Knowledge

Sci-TS drop by 9–10%, 12–16%, 4%, and 8–33%, respectively. Concatenation of sequential write requests only marginally affects the mean response times for *Order* and *Report*.

For all traces except *Snake*, sequential stream optimizations for read requests only have a significant impact on the SSTF algorithm (if at all). Adding either concatenation or sequential scheduling of sequential reads to SSTF.CW decreases mean response times by 7%, 36%, 31–34%, and 5% for *Air-Rsv*, *Sci-TS*, *Order*, and *Report*, respectively. The performance of C-LOOK was not expected to change with the addition of sequential scheduling optimizations (since it already schedules requests in logically ascending order), but the minimal improvement in SPTF-based algorithm performance was surprising. For workloads "farther up" the response time curves (i.e., workloads on the very edge of saturation), the sequential stream optimizations enhance performance more significantly (e.g., 10–15% for SPCTF scheduling of *Sci-TS* at a trace scaling factor of 3.45).

For the *Snake* trace, sequential stream optimizations for read requests provide significant performance improvements. Assuming the scheduler already concatenates sequential write requests, adding concatenation of sequential reads decreases mean response times by 60–70%. Alternately, sequential scheduling of read requests decreases mean response times by 27–30% for the two SPTF-based algorithms. Since C-LOOK already schedules sequential requests in logically ascending order, the latter optimization does not alter its performance. SSTF.CW experiences a 3% drop in mean response time with sequential scheduling of reads.

The *Snake* trace was examined to determine the reason behind the major performance advantage of the sequential stream optimizations for this workload. The trace contains occasional bursts of numerous sequential read requests. For example, the first "day" of the trace (containing over 192,000 requests) has a burst of 3848 read requests of size 8 KB serviced by a single disk. The requests are grouped into 256 KB sequential streams (i.e., 32 requests per stream). The traced interarrival times show the entire burst being received in just over 17 seconds. The sequential stream optimizations obtain most of their performance advantage from scheduling such bursts. For this particular burst, a C-LOOK scheduler with concatenation of sequential read requests drops the number of requests serviced by the disk to 1345 (i.e., 65% fewer requests).

Table 5.4 lists the reduction in the total number of requests (serviced by the disks) due to request concatenation for C-LOOK, SSTF, SPCTF, and ASPCTF(6). Although some of the percentages indicate where concatenation provides a performance advantage, others are deceptive. For example, C-LOOK.CW scheduling of *Sci-TS* decreases mean response times by 8–22% by concatenating only 0.35% of all requests. C-LOOK.CW.CR concatenates an additional 2.7–3.4% requests, but only decreases mean response times by 1–2%.

In these experiments, concatenation of sequential reads is superior to sequential scheduling because of the decrease in total overhead delay incurred at the disk. If a disk has zero (or close to zero) command initiation and completion overheads, sequential scheduling of read requests provides slightly better performance than concatenation of sequential read requests.

Scheduling with FCFS Command Queued Disks

Even host-based centralized scheduling can take advantage of the inter-request concurrency available from disks with command queueing. Figures 5.29–5.34 present mean response times for C-LOOK at points along the "knee" of each set of response time curves. The

Trace	Scale Factor	C-LOOK	SSTF	SPCTF	ASPCTF(6)
Cello	1.75	1.53	1.52	1.48	1.49
Snake	1.25	2.07	2.04	2.09	2.10
Air-Rsv	2.5	0.21	0.20	0.19	0.19
Sci-TS	2.5	0.35	0.36	0.29	0.30
Order	1.0	0.01	0.02	0.01	0.01
Report	1.0	0.01	0.01	0.01	0.01

(a) Concatenation of Sequential Writes

Trace	Scale Factor	C-LOOK	SSTF	SPCTF	ASPCTF(6)
Cello	1.75	2.74	2.60	2.30	2.31
Snake	1.25	5.64	5.65	5.45	5.45
Air-Rsv	2.5	3.43	3.07	2.70	2.73
Sci-TS	2.5	3.74	3.48	2.95	2.97
Order	1.0	0.70	3.76	0.49	0.46
Report	1.0	3.87	0.74	3.48	3.52

(b) Concatenation of Sequential Reads and Writes

Table 5.4: Percentage Reduction in Total Requests Serviced due to Sequential Request Concatenation

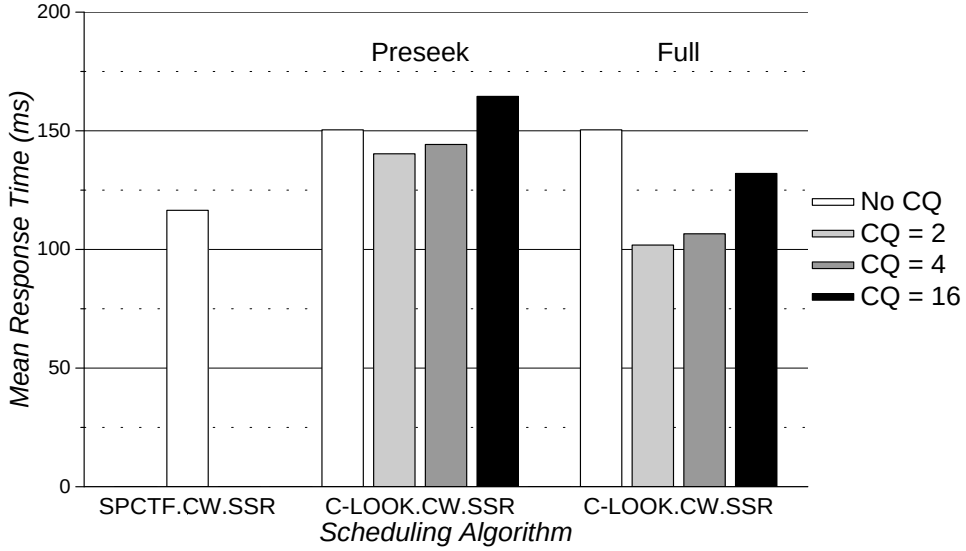


Figure 5.29: *Cello*, 1.75X: C-LOOK Performance with FCFS Command-Queued Disks

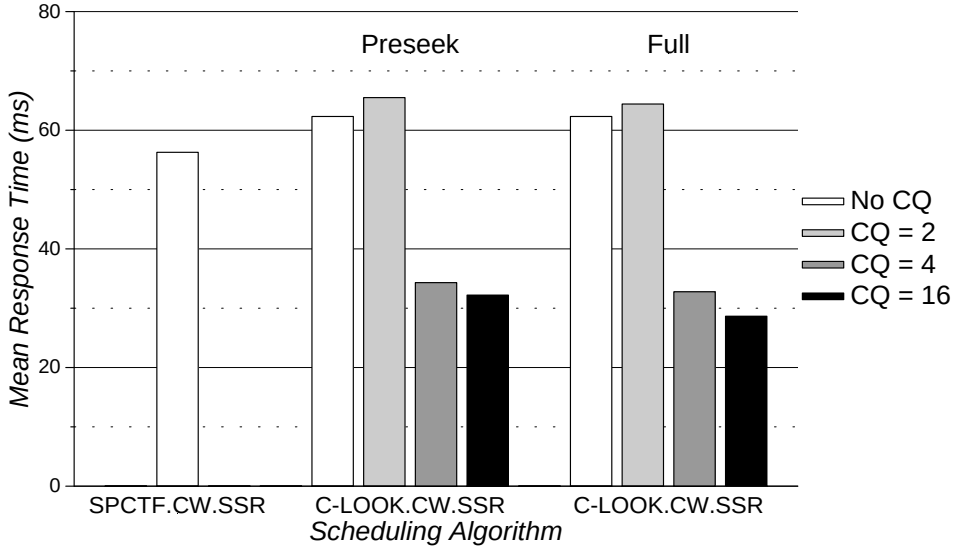


Figure 5.30: *Snake*, 1.25X: C-LOOK Performance with FCFS Command-Queued Disks

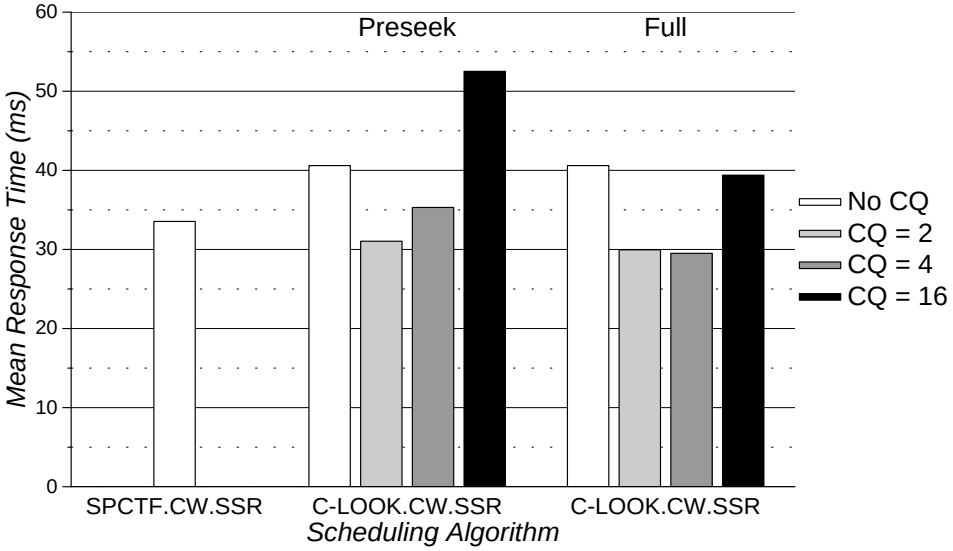


Figure 5.31: *Air-Rsv*, 2.5X: C-LOOK Performance with FCFS Command-Queued Disks

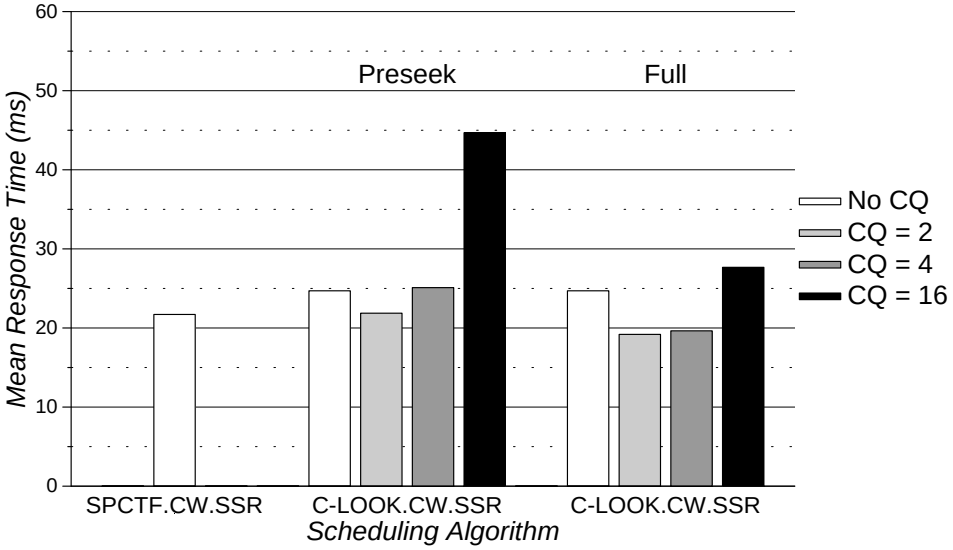


Figure 5.32: *Sci-TS*, 2.5X: C-LOOK Performance with FCFS Command-Queued Disks

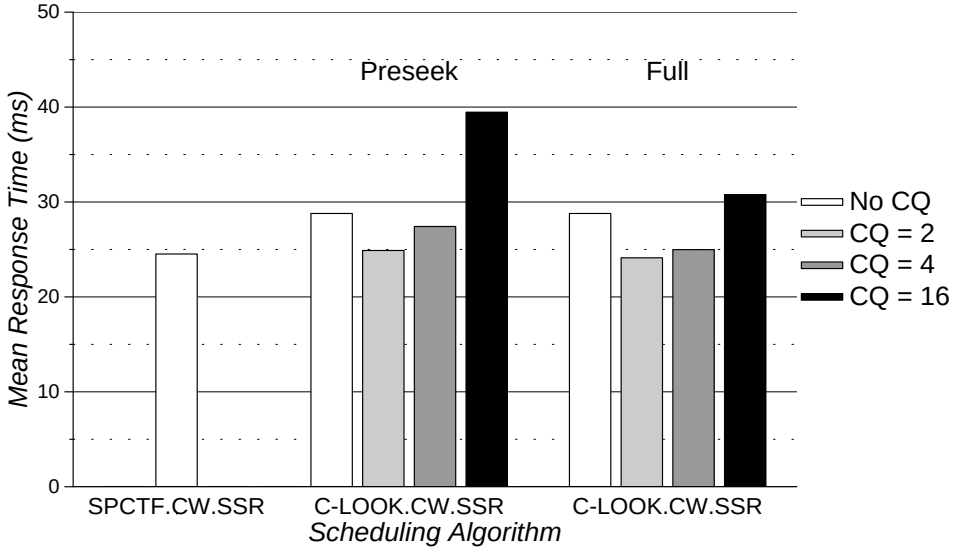


Figure 5.33: *Order*, 1.0X: C-LOOK Performance with FCFS Command-Queued Disks

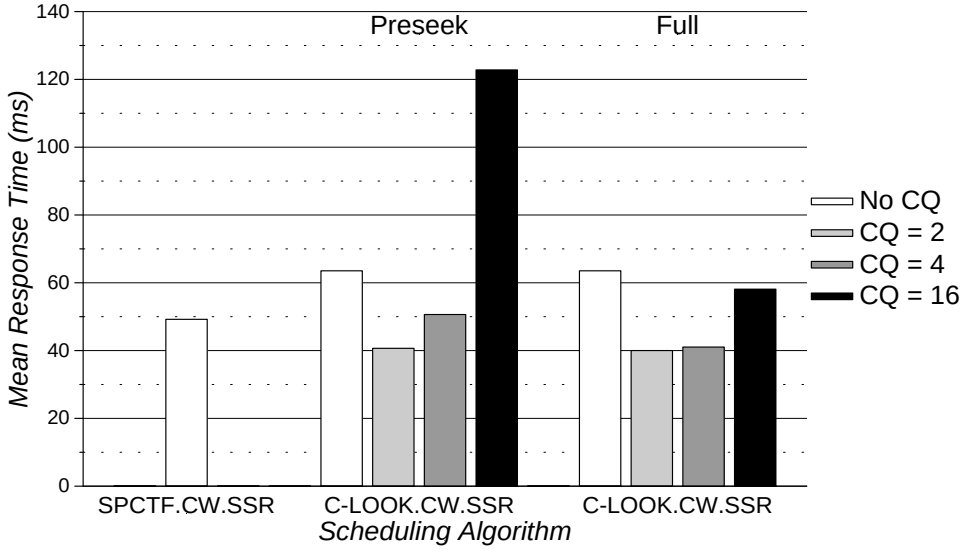


Figure 5.34: *Report*, 1.0X: C-LOOK Performance with FCFS Command-Queued Disks

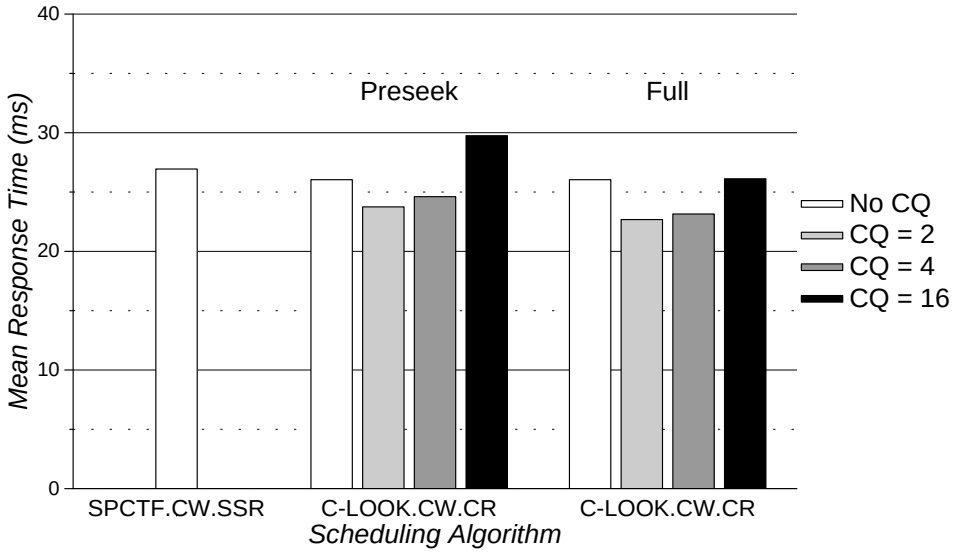


Figure 5.35: *Snake*, 1.25X: C-LOOK Performance with FCFS Command-Queued Disks

scheduler was configured to concatenate sequential write requests and sequentially schedule read requests. The first bar on each graph gives the performance of SPCTF.CW.SSR (for reference).[2] The remaining two sets of four bars give the mean response times for disks configured with Preseek and Full command queueing (described in section 4.1.2) over a range of maximum on-board queue depths (labeled CQ).

All of the traces exhibit relatively similar behavior except *Snake*. Section 5.1.2 demonstrates how concatenation of sequential read requests significantly outperforms sequential scheduling of read requests for the *Snake* trace. The simulations were repeated using concatenation of reads, and figure 5.35 shows that the mean response time behavior of C-LOOK.CW.CR scheduling for *Snake* matches the C-LOOK.CW.SSR behavior of the other traces.

The results for all six traces indicate that a host-based centralized scheduler generally performs best with a disk command queue length of two. By maintaining just two outstanding requests per disk, the scheduler takes advantage of concurrency between the "current" request and the "next" request at each disk, while preserving the ability to dynamically reorder (or concatenate) the remainder of the pending requests. That is, a host-based scheduler retains more flexibility to rearrange pending requests if the command queue length is kept short.

This also explains the difference in C-LOOK behavior between the two sequential stream optimizations for *Snake* read requests. A host-based C-LOOK.CW.SSR scheduler does not need to continually reorder the large bursts of sequential reads that dominate the performance metrics for this trace. Each sequential stream arrives at the scheduler in logically ascending

[2] As a host-based SPCTF algorithm relies on current knowledge of on-board data cache contents and actuator positions, it does not schedule effectively for disks with command queueing.

order. Therefore, the disk subsystem achieves the best performance by maximizing the available inter-request concurrency at the disk.

For disks using the Preseek command queueing configuration, the mean response times for the six traces drop 7–36% when the maximum command queue length is two. They drop an additional 1–25% after enabling Full command queueing.

5.1.3 Full System Traces

This section contains a similar set of experiments comparing host-based schedulers for the NCR traces. These full system traces enable the use of the process-flow host model, which simulates the bidirectional feedback between a host and a disk subsystem. Although the algorithms studied in this section optimize for subsystem performance criteria, the process-flow model allows the measurement of application-specific performance metrics as well. As before, the baseline scheduling algorithms are compared before adding sequential scheduling optimizations or FCFS command-queued disks.

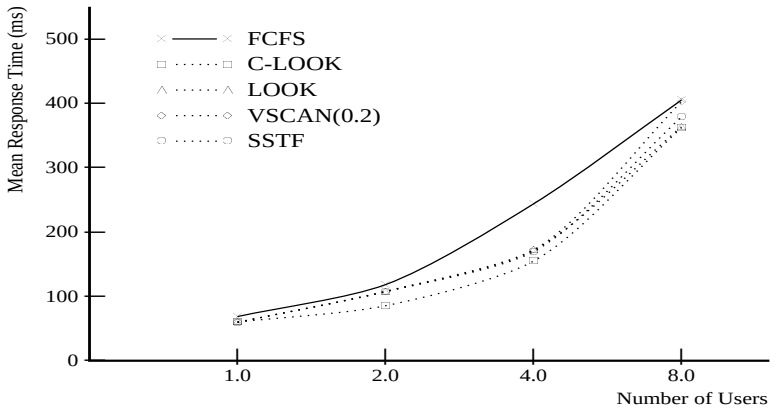
Scheduling by Logical Block Number

Figure 5.36 shows performance data for the *SynRGen* workload configured with 1, 2, 4, and 8 users. The number of users is plotted along the X-axis using a $\log_2$ scale. Each data point is the mean of multiple trace runs. The *SynRGen* benchmark uses random number generation to determine the sequence of events, the compute times, and the sleep times for each "user" generating activity. As a result, the performance metrics vary significantly between runs (using the same configuration).

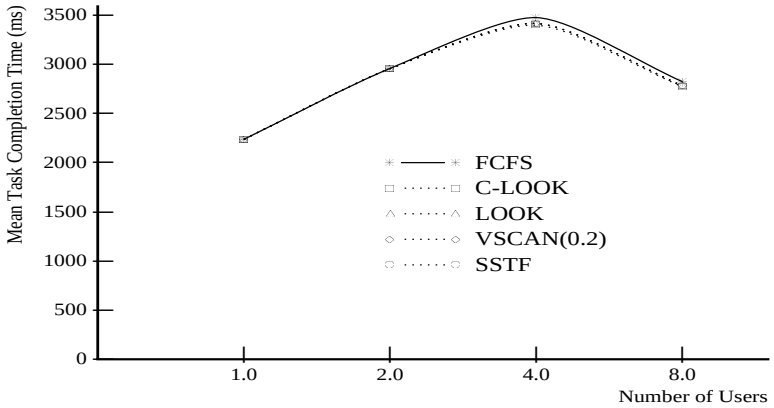
Figure 5.36a shows the mean response times for disk requests using FCFS and LBN-based C-LOOK, LOOK, VSCAN(0.2), and SSTF. The seek-reducing algorithms clearly outperform FCFS. For the 4-user workload, its mean response time is over 40% higher. C-LOOK provides the lowest mean response times for all but the lightest (1-user) workload, where SSTF outperforms C-LOOK by 1%. For the 2-user and 4-user workloads, C-LOOK decreases the mean response time by 20% and 7%, respectively, compared to the other seek-reducing algorithms.

Figure 5.36b shows the corresponding mean completion times for the *SynRGen* tasks. For this metric, the seek-reducing algorithms outperform FCFS by only 2%. However, the largest fraction of the task completion time is the time "executing" code from main memory (the **compute time**). Figure 5.36c shows the mean task non-compute times (i.e., the mean task completion times excluding the large application compute times). For the 4-user and 8-user workloads, the mean non-compute time drops by 12–13% and 7–10%, respectively, when a seek-reducing algorithm replaces FCFS. C-LOOK outperforms the other seek-reducing algorithms by 1–3% for these workloads.

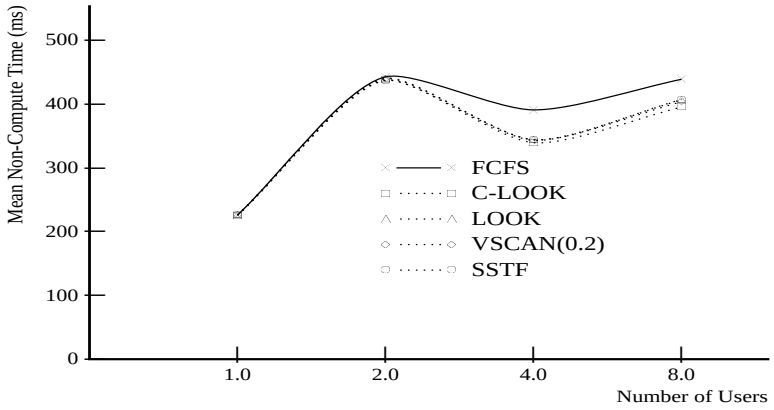
Figure 5.37 presents performance data for experiments using the *Compress* workload. Each data point is the mean of five separate trace runs. Figure 5.37a shows the mean response times for FCFS and the LBN-based algorithms. For this metric, the seek-reducing algorithms produce 15% better performance than FCFS. Figure 5.37b shows the total run time for the file compression process. The seek-reducing algorithms decrease compression times by approximately 2%. Although this may seem insignificant, the magnitude of the



(a) Mean Request Response Time

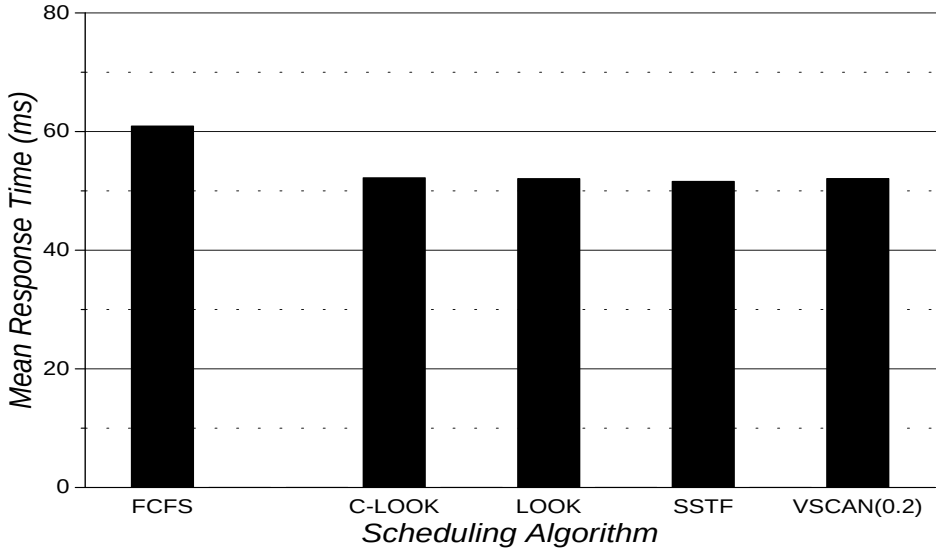


(b) Mean Task Completion Time

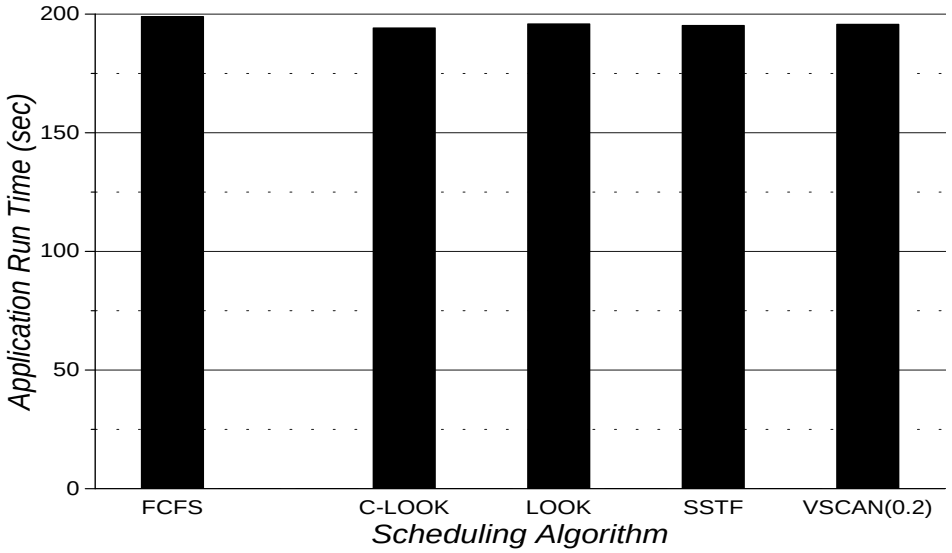


(c) Mean Non-Compute Time

Figure 5.36: *Synrgen*: LBN-Based Algorithm Performance



(a) Mean Response Time



(b) Application Run Time

Figure 5.37: *Compress*: LBN-Based Algorithm Performance

improvement should increase with the processing power of the host system. On the NCR workstation traced, the *Compress* workload is at least partially CPU-bound; the processor spends approximately 10% of its time in the idle loop. For a faster host system, *Compress* could become I/O-bound, resulting in a workload much more sensitive to disk subsystem performance.

Scheduling with Full Knowledge

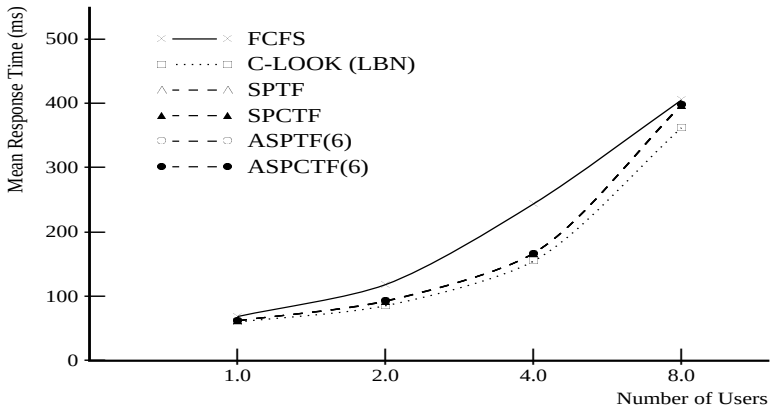
Figure 5.38a shows mean response times for SPTF-based scheduling of *SynRGen*. The full-knowledge algorithms have 7–9% longer mean response times than C-LOOK for all but the 1-user workload. However, this is not reflected in either the mean task completion time or the mean non-compute time (see figures 5.38b and 5.38c).

Figure 5.39a shows the mean response times for full-knowledge scheduling of the *Compress* workload. C-LOOK has a 2% lower mean response time than SPTF and ASPTF(6). The cache-sensitive algorithms provide the worst performance, with mean response times 19% higher than C-LOOK and only 2% lower than FCFS. Figure 5.39b shows corresponding data for the compression run times. C-LOOK outperforms SPTF and ASPTF(6) by 0.5% using this metric. Adding cache-sensitivity to the SPTF-based algorithms increases the run times by another 1%.

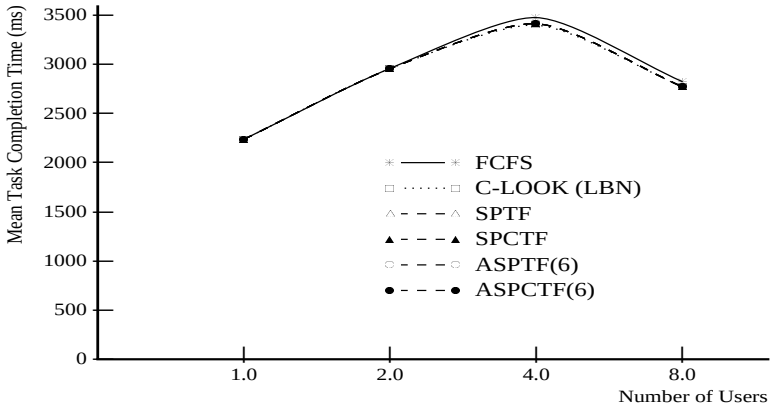
Intuitively, adding cache-sensitivity should enhance performance, not degrade it. In this case, the inferior performance of the cache-sensitive algorithms is due to an unfortunate interaction between the disk model's prefetching algorithms and the specific behavior of the *Compress* workload. The default disk model configures the segmented cache to prefetch data after read hits as well as read misses. The *Compress* workload consists of separate sequential streams of read requests and write requests (reading the uncompressed file and writing the compressed file). The write requests occur in bursts triggered by an O/S daemon that periodically flushes dirty blocks from the host disk block cache (see section 2.2.1).

Consider the situation when the disk is servicing a burst of write requests and a read request arrives whose data is completely contained in a read segment of the on-board cache. Cache-sensitive algorithms give immediate priority to the read request, and (with the default prefetching configuration) the disk initiates a seek to prefetch additional data into the read segment. But as soon as the read request is satisfied from the on-board cache, the host-based scheduler immediately issues the next pending write request. This preempts the prefetching activity — perhaps before the disk actuator even reaches the target cylinder containing the data blocks to prefetch. An additional seek returns the disk actuator to the cylinder containing the current stream of write requests. Without cache-sensitivity, the scheduler would finish the write burst before servicing the read request. Since most read requests for *Compress* are time-limited, this does not necessarily lengthen the application run time (if the write burst completes before the time limit).

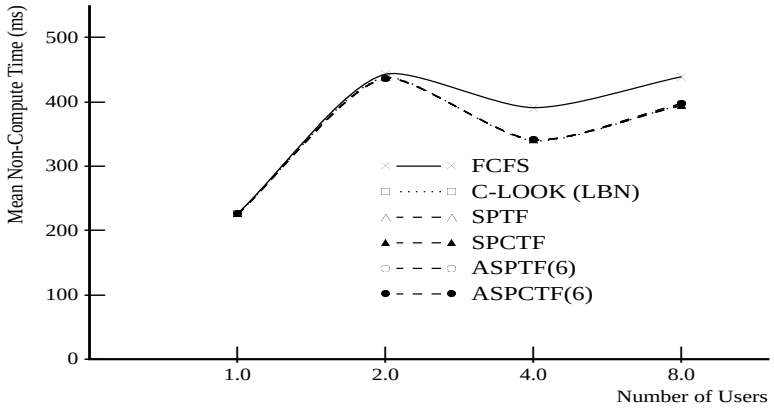
If the disk model prefetches only on read misses (i.e., not on full read hits), the cache-sensitive scheduling algorithms provide superior performance, as measured by both mean response times and application run times. Of all the full-knowledge configurations, SPCTF with read miss prefetching has the lowest run time (although it is still slightly higher than that of C-LOOK).



(a) Mean Request Response Time

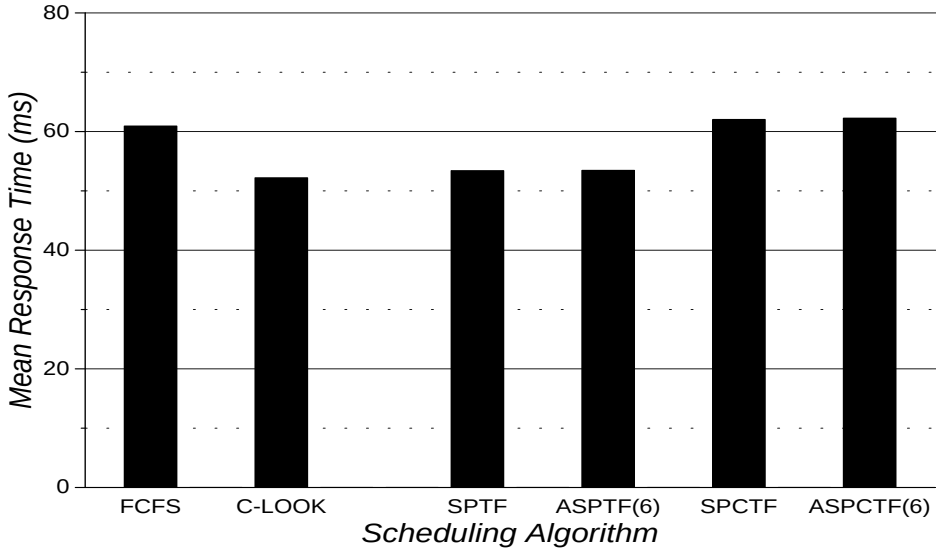


(b) Mean Task Completion Time

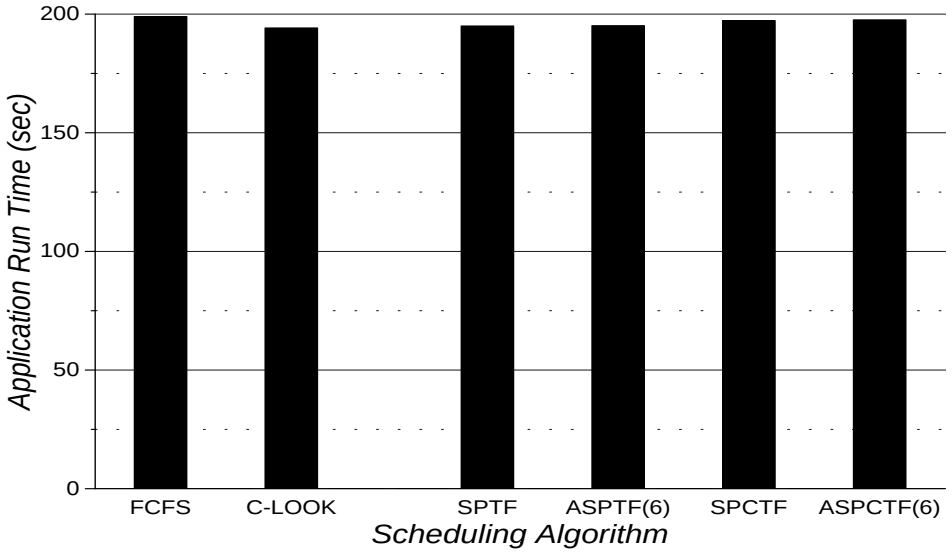


(c) Mean Non-Compute Time

Figure 5.38: *Synrgen*: Full-Knowledge Algorithm Performance



(a) Mean Response Time



(b) Application Run Time

Figure 5.39: *Compress*: Full-Knowledge Algorithm Performance

Sequential Stream Optimizations

Figures 5.40–5.43 show the effects of the sequential stream optimizations for scheduling *SynRGen*. Sequential stream optimizations for read requests have little or no effect on mean request response times. For the 1-user, 2-user, and 4-user workloads, concatenation of sequential write requests improves mean response times by 1% or less. For the 8-user workload, the mean response times decrease by 3–5%, with C-LOOK dropping 5.1% (the largest decrease among the eight algorithms). C-LOOK always schedules sequential writes in logically ascending order, a behavior that can degrade performance when servicing sequential write streams with disks lacking write prebuffering (see section 5.1.2, page 51). By concatenating streams of sequential writes at the host, the number of sequential writes issued to the disk in logically ascending order decreases.

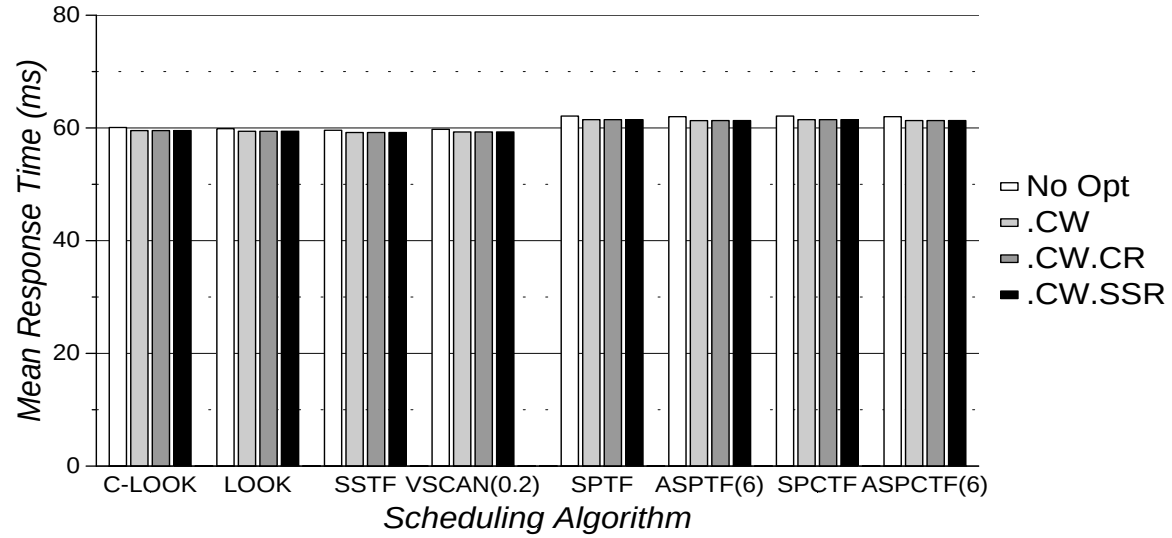
The mean non-compute times reflect the improvement in mean response times caused by concatenation of sequential writes. Figure 5.43b shows the mean non-compute times for an 8-user *SynRGen* workload decreasing by 0.4–2.3% as a result. SPTF and SPCTF improve the most (2.3%), followed by the age-sensitive SPTF-based algorithms (1.4%). Thus, the SPTF-based algorithms actually outperform C-LOOK when using a system-level (i.e., application-specific) performance metric. This indicates that C-LOOK's improvement in mean response time is probably due to better handling of the bursts of write requests issued by the O/S cache flush daemon. Although it is important to service such background requests efficiently, it is more important to service critical requests (e.g., read requests for which processes are blocking). That is, improved service for background write requests only improves application run times indirectly (if at all), whereas improved service for high priority requests typically allows a process to finish more quickly.

Figure 5.44 gives the mean response times and application run times for the *Compress* workload. Concatenation and sequential scheduling of read requests does not affect these performance metrics. Concatenation of sequential writes improves mean response times by 3% for all of the scheduling algorithms. The file compression completes 0.5–0.8% faster as a result. C-LOOK benefits the most (0.8%) from this particular optimization. This indicates the C-LOOK is not only servicing the background writes efficiently, it is also providing better service for the critical requests in the *Compress* trace.

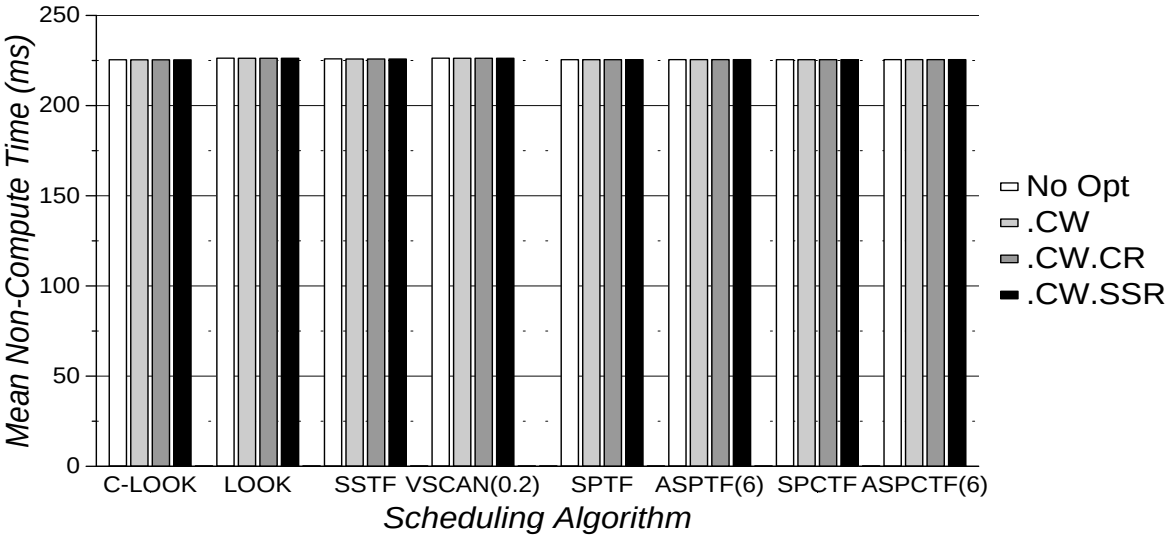
Scheduling with FCFS Command Queued Disks

Figures 5.45–5.48 give performance data for host-based C-LOOK.CW.SSR scheduling of *SynRGen* using disks with FCFS command queueing. The bargraphs also show host-based SPTF.CW.SSR and SPCTF.CW.SSR algorithm performance (using non-queued disks) for reference.[3] For Preseek and Full command queueing configurations, a command queue length of two provides the best mean response times regardless of the number of users. Allowing up to two commands queued at each disk lowers mean response times by 4–10% for Preseek command queueing and 14–20% for Full command queueing. The improvement in mean response time increases with the number of users.

[3] As host-based SPCTF algorithms rely on current knowledge of on-board data cache contents and actuator positions, they do not schedule effectively for disks with command queueing.

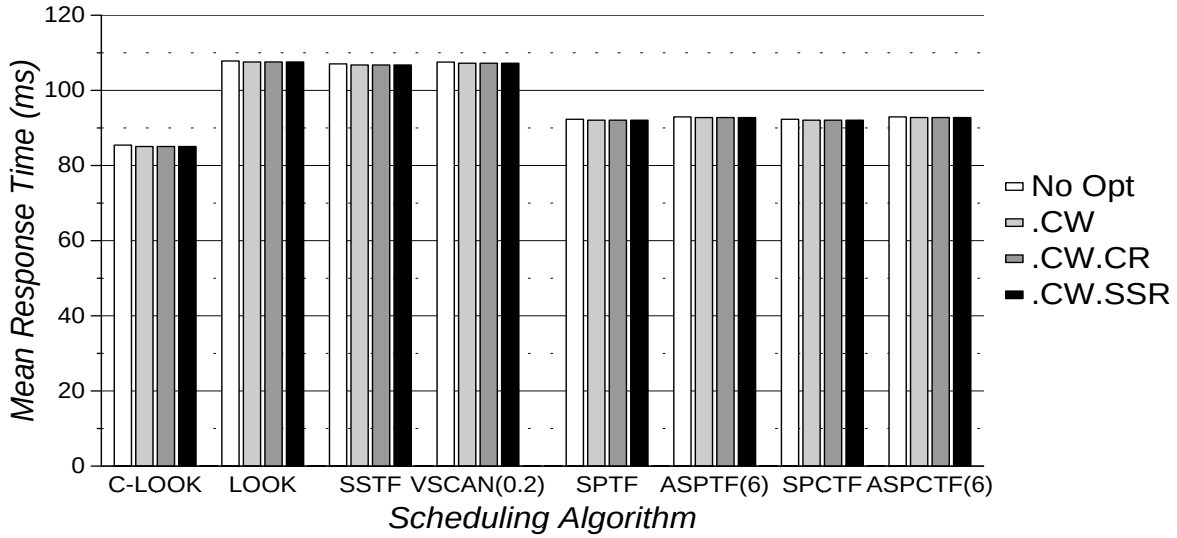


(a) Mean Response Time

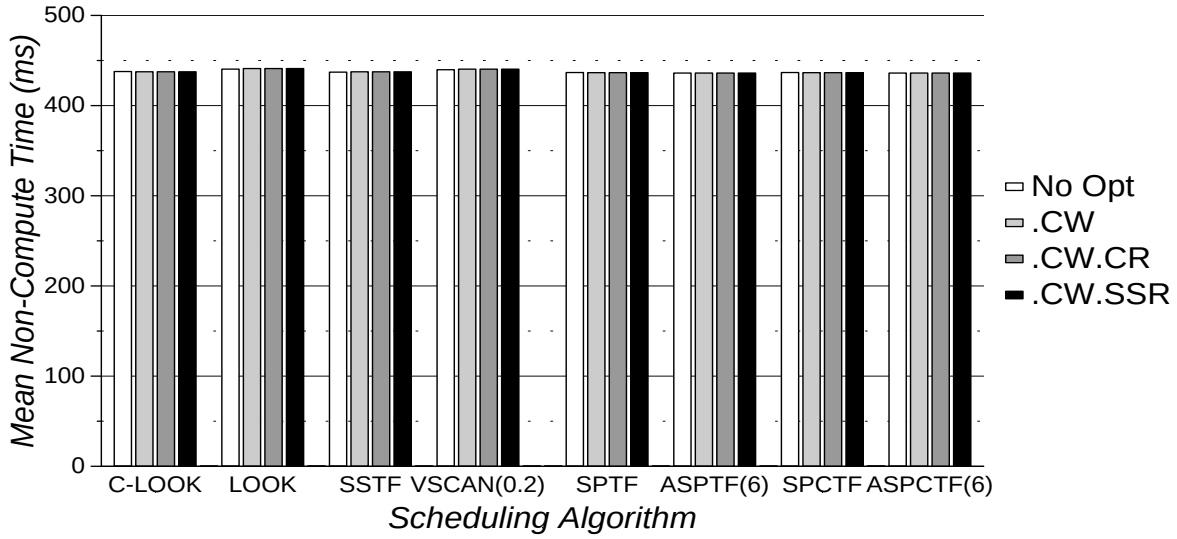


(b) Mean Non-Compute Time

Figure 5.40: *SynRGen*, 1 User: Sequential Scheduling Algorithm Performance

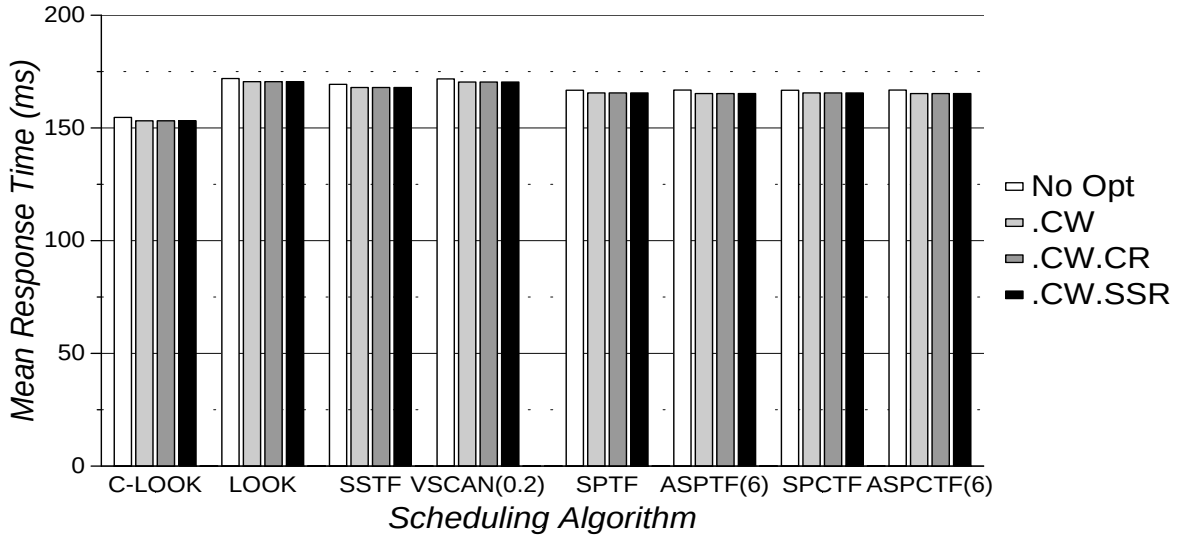


(a) Mean Response Time

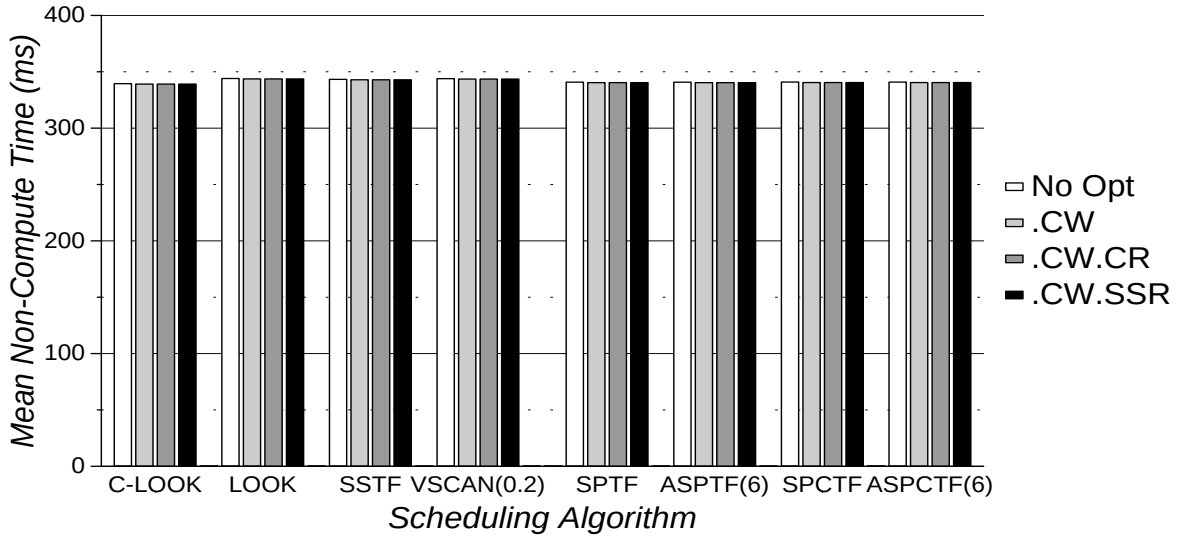


(b) Mean Non-Compute Time

Figure 5.41: *SynRGen*, 2 Users: Sequential Scheduling Algorithm Performance

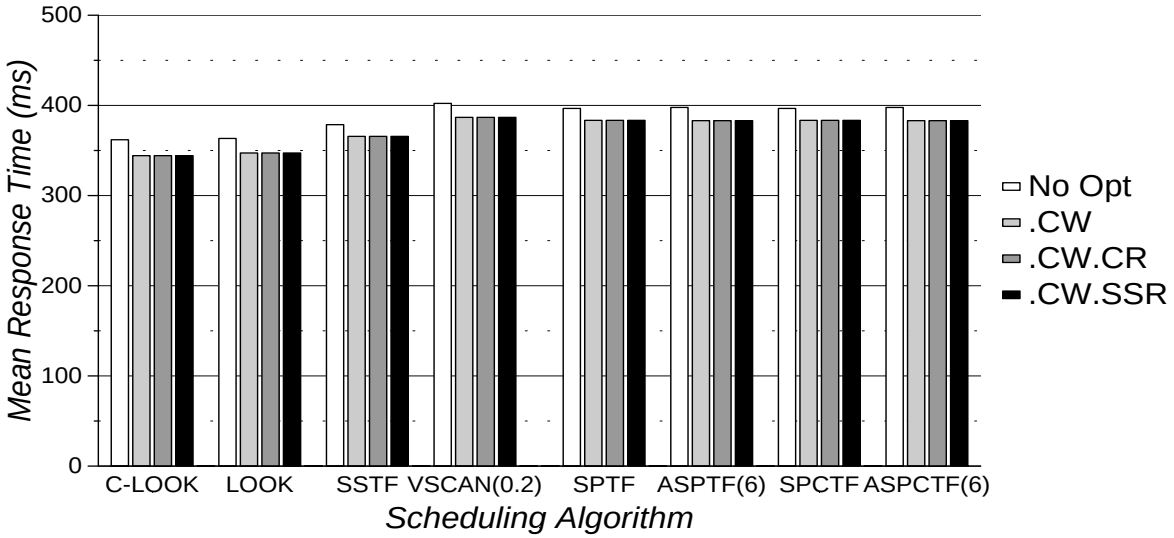


(a) Mean Response Time

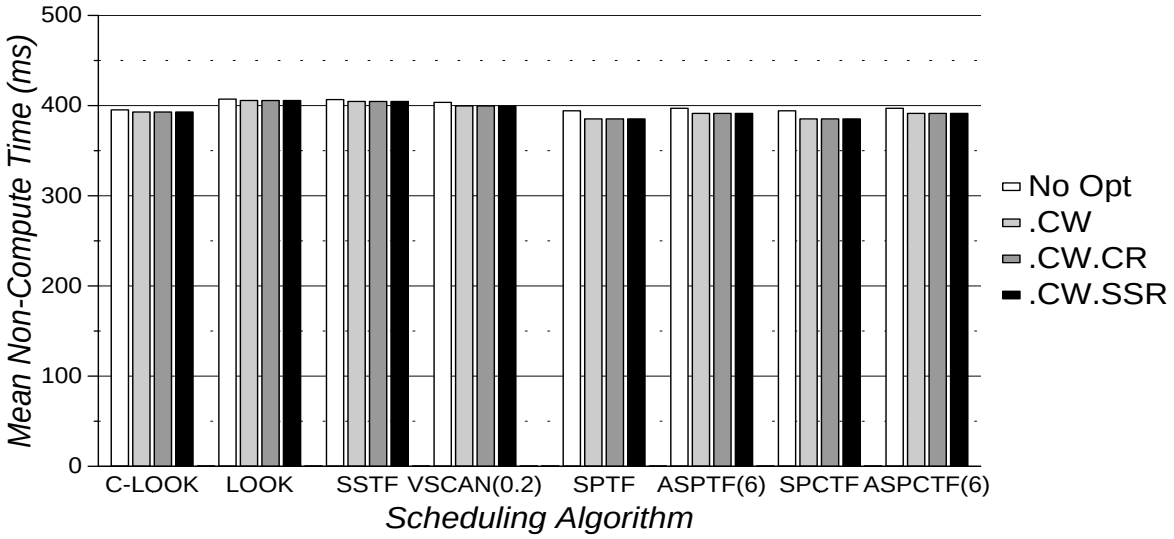


(b) Mean Non-Compute Time

Figure 5.42: *SynRGen*, 4 Users: Sequential Scheduling Algorithm Performance

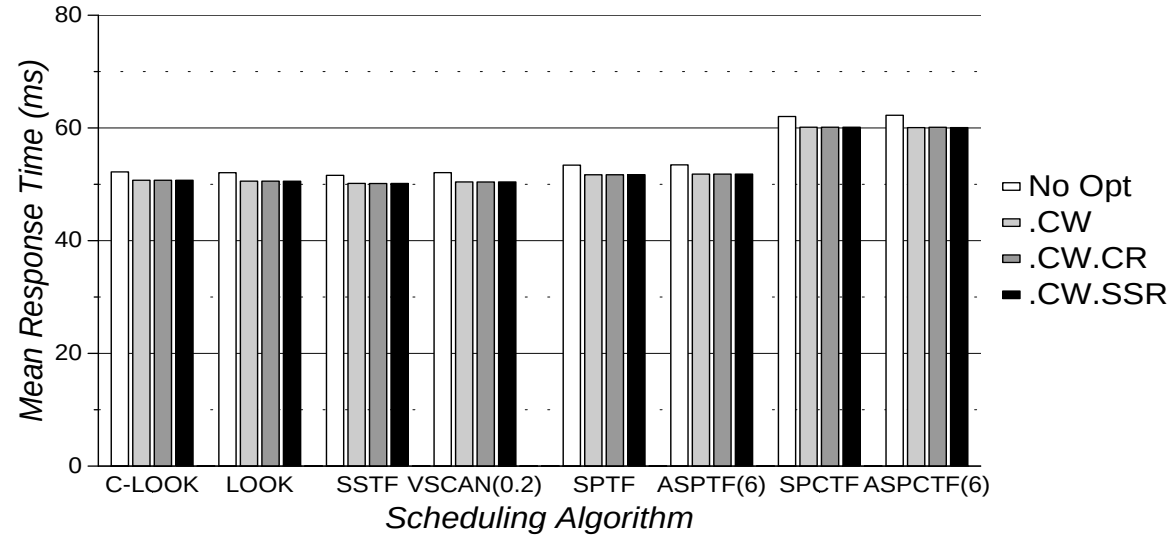


(a) Mean Response Time

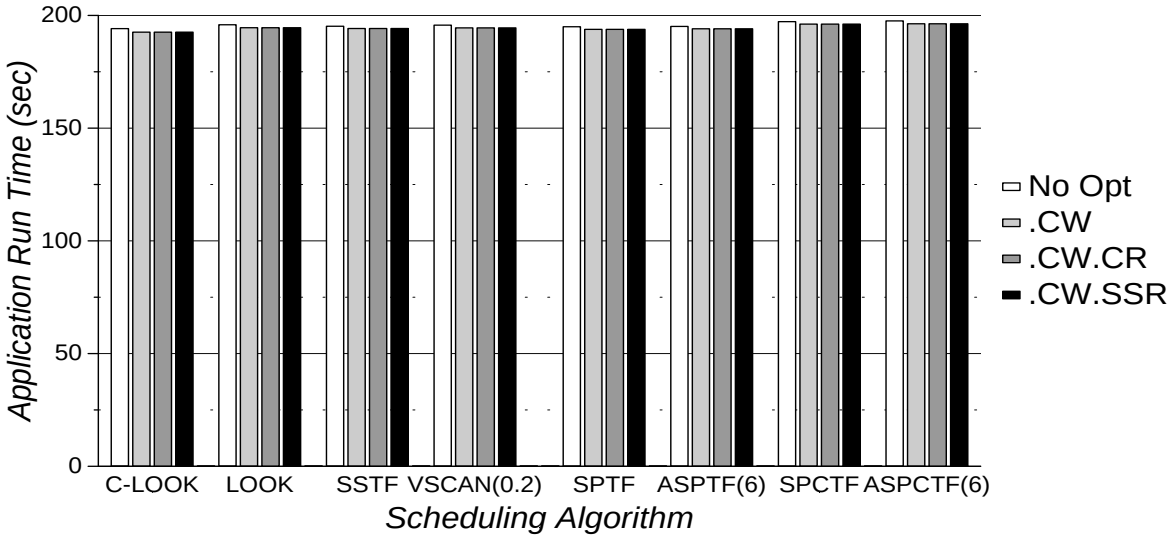


(b) Mean Non-Compute Time

Figure 5.43: *SynRGen*, 8 Users: Sequential Scheduling Algorithm Performance

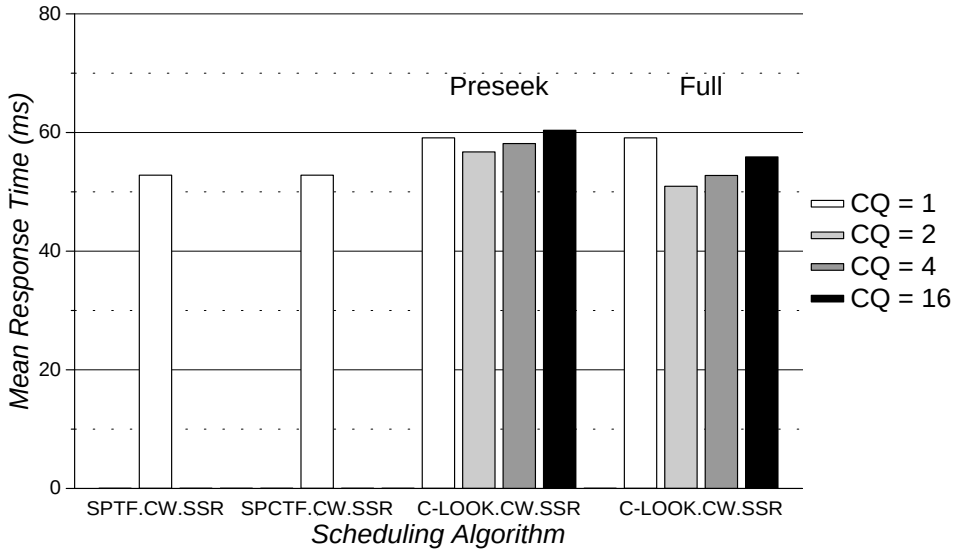


(a) Mean Response Time

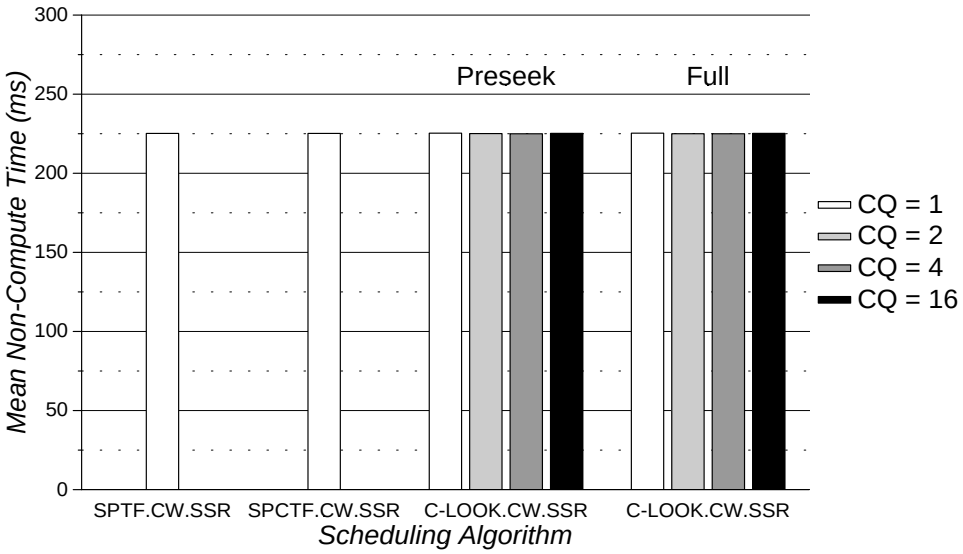


(b) Application Run Time

Figure 5.44: *Compress*: Sequential Scheduling Algorithm Performance

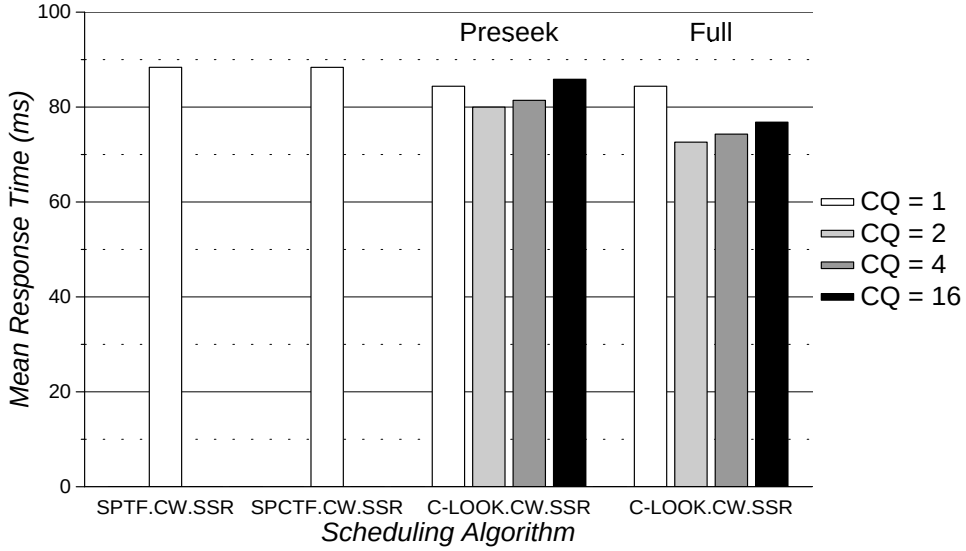


(a) Mean Response Time

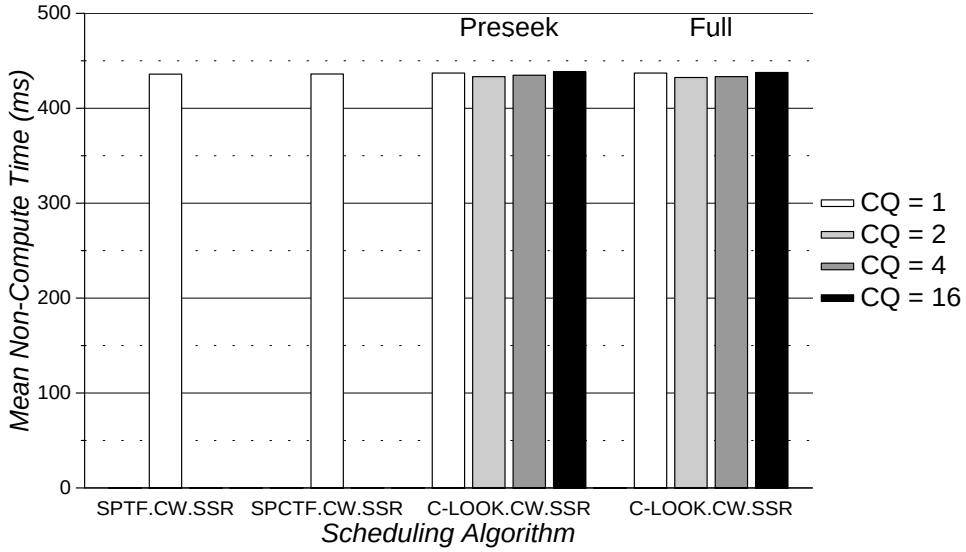


(b) Mean Non-Compute Time

Figure 5.45: *SynRGen*, 1 User: Scheduling Algorithm Performance for Different Command Queue Lengths

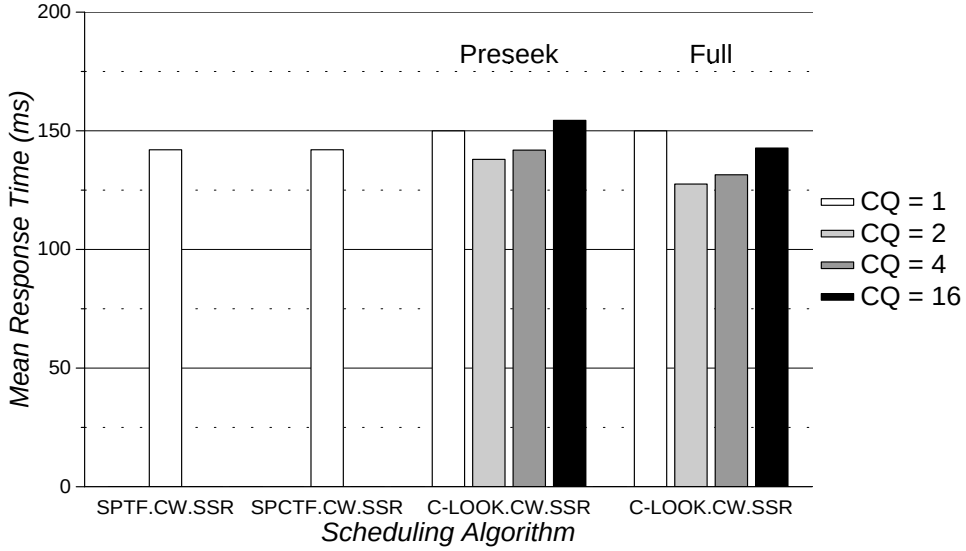


(a) Mean Response Time

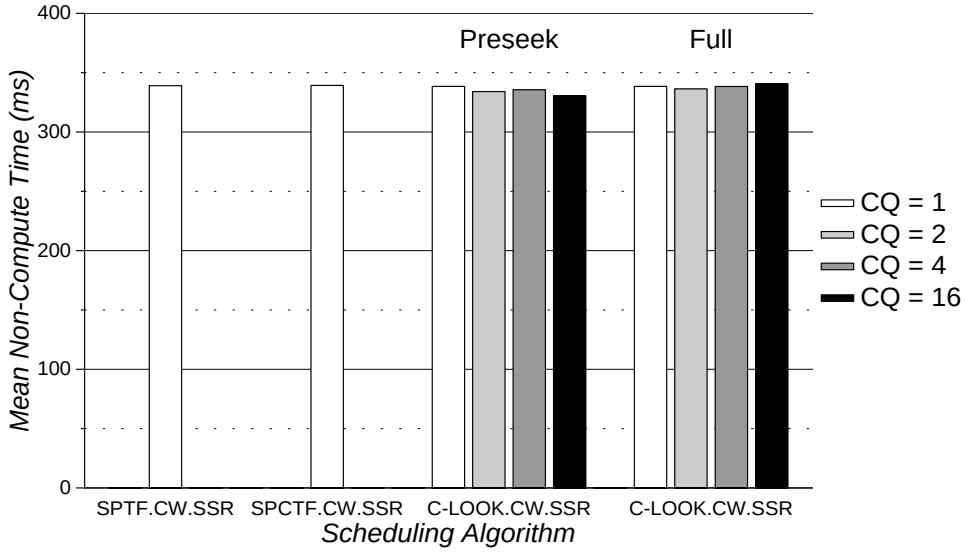


(b) Mean Non-Compute Time

Figure 5.46: *SynRGen*, 2 Users: Scheduling Algorithm Performance for Different Command Queue Lengths

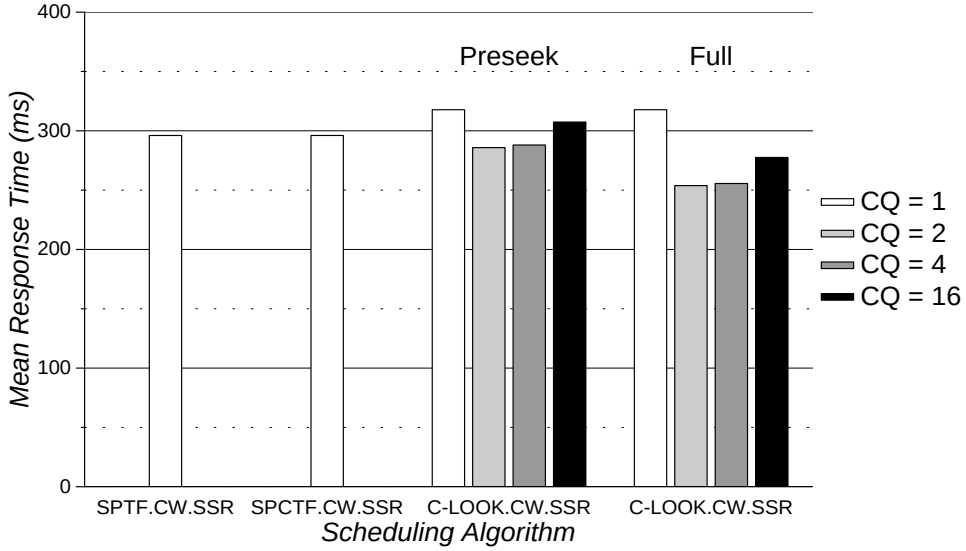


(a) Mean Response Time

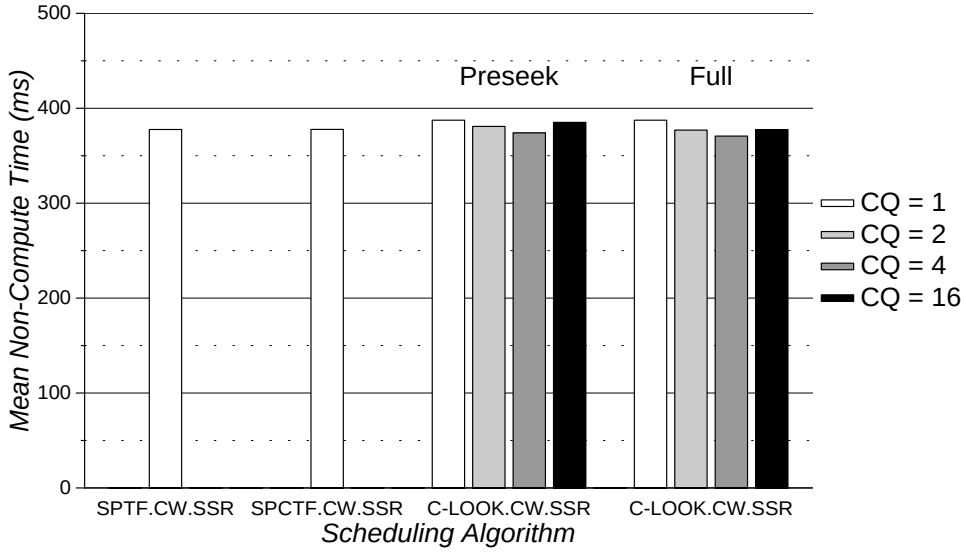


(b) Mean Non-Compute Time

Figure 5.47: *SynRGen*, 4 Users: Scheduling Algorithm Performance for Different Command Queue Lengths



(a) Mean Response Time



(b) Mean Non-Compute Time

Figure 5.48: *SynRGen*, 8 Users: Scheduling Algorithm Performance for Different Command Queue Lengths

For the 1-user, 2-user, and 4-user workloads, the mean non-compute time varies only slightly (up to 2.3%) for maximum command queue lengths of 1, 2, 4, and 16. The mean response time difference between maximum queue lengths of 1 and 2 is only 0.2–1.1%. For the 8-user workload, however, the maximum command queue length begins to visibly affect the mean non-compute time (by up to 4.3%). Allowing two commands queued per disk improves mean non-compute times by 2% for Preseek command queueing and 3% for Full command queueing. Allowing four commands improves performance by another 2%. Increasing the command queue length beyond four begins to degrade performance, as the host-based scheduler loses more and more of its ability to dynamically reorder the pending requests.

Figure 5.49 gives the mean response times and application run times for the *Compress* workload using the same range of maximum command queue lengths. A queue length of two provides the best performance, improving mean response times by 5% and 10% for Preseek and Full command queueing, respectively. For both configurations, the application runs 1.2% more quickly with a command queue length of two.

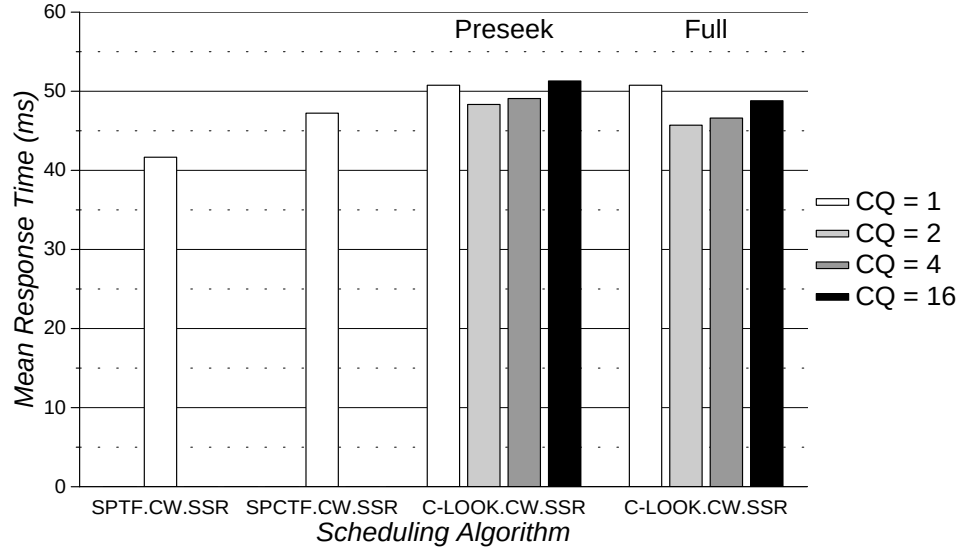
5.1.4 Summary of Conclusions

As section 5.1 presents a large number of experiments, it is useful to restate some of the major conclusions:

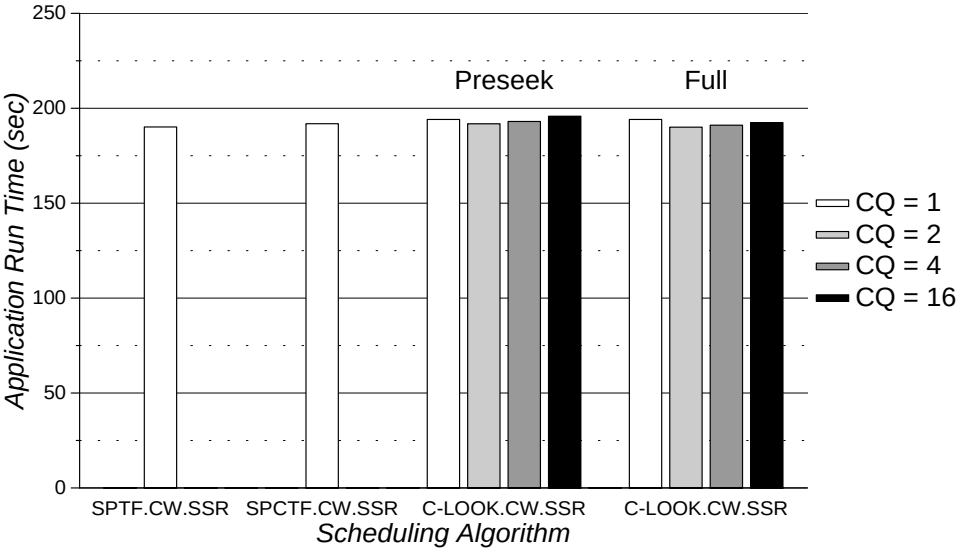
Seek-reducing algorithms do not require detailed mappings of logical blocks to physical media locations, since logical adjacency generally implies physical adjacency in modern disk drives. Having access to accurate data layout mappings only marginally improves scheduling performance for such algorithms. Among the seek-reducing algorithms, C-LOOK interacts best with on-board prefetching caches. This results in lower mean response times for most of the workloads studied and increased system-level performance for the full system traces.

Algorithms designed to reduce combined seek and rotational latency delays require accurate logical-to-physical mappings, seek curves, head switch times, command and completion overheads, rotation speeds, and knowledge of current actuator positions. This information allows a scheduler to accurately predict initiation delays for pending requests and thereby select better request schedules. Subsystem and system metrics indicate that such full-knowledge algorithms often provide the highest levels of performance. For experiments using the disk request traces, SPTF-based algorithms that incorporate knowledge of on-board cache state achieve the lowest mean response times for most sub-saturation workloads. For experiments using the full system traces, SPTF-based algorithms with cache-sensitivity provide higher performance if the disk model only prefetches on read misses (i.e., not on read hits). Otherwise, an unfortunate interaction occurs between the workload and the prefetching algorithm, making cache-sensitivity inadvisable.

If a workload contains a significant fraction of sequential requests, a host-based scheduler can improve performance by concatenating sequential requests into fewer, larger requests. This technique reduces the total command and completion overheads incurred at the disk and better utilizes the on-board data cache. Alternately, a scheduler can sometimes improve performance by explicitly scheduling sequential requests in logically ascending order. The latter optimization is contraindicated for sequential streams of write requests serviced by disks without the ability to prebuffer write request data, as it produces an almost pathologi-



(a) Mean Response Time



(b) Application Run Time

Figure 5.49: *Compress*: Scheduling Algorithm Performance for Different Command Queue Lengths

cally inferior request ordering. Also, the selected implementation of the sequential scheduling optimization is better suited to SSTF than LOOK or VSCAN(0.2).

In simulations using disks with FCFS command queueing, shorter command queue lengths (e.g., 2–4 pending requests per disk) result in better performance. Short on-disk queues allow a host-based scheduler to maintain the flexibility to dynamically reorder the majority of the pending requests, while exploiting some of the inter-request concurrency available at each disk. It is likely that the optimal queue length varies with the cache configuration. However, such a study is beyond the scope of this dissertation.

5.2 Scheduling with Information From "Above" and "Below"

This section examines the system performance effects of adding system-level knowledge to host-based centralized scheduling algorithms. Without explicit knowledge of request priorities or the impact of individual requests on performance and/or reliability goals, a scheduler can only attempt to prevent the starvation of critical requests by minimizing the response time variance. Such is the case for scheduling the disk request traces from HP and DEC. If applications or file systems pass system-level information to a scheduler, it can give priority to the most "important" requests. The full system traces (taken from the NCR workstation) identify which requests are time-critical and time-limited, allowing a scheduler to optimize for system performance. In either case, a scheduler should still use the subsystem hardware efficiently (by exploiting hardware-specific knowledge).

5.2.1 Disk Request Traces

Figures 5.16–5.21 in section 5.1.2 show the mean response times and coefficients of variation for ASPTF(6) and ASPCTF(6) scheduling of the six HP and DEC traces. The age-sensitive SPTF-based algorithms are visibly more resistant to starvation than SPTF and SPCTF. However, C-LOOK provides equivalent (or slightly superior) starvation resistance for workloads on the very edge of saturation. When the age-sensitive algorithms begin to saturate, their behavior starts to resemble FCFS (with a resulting sharp increase in mean response times). This is most visible with the *Sci-TS*, *Order*, and *Report* traces. The seek-reducing algorithms (especially C-LOOK) saturate much more gradually. For some systems, this type of graceful degradation is also an important goal.

Because ASPTF(6) does not effectively utilize the on-board cache, it saturates more quickly than either the seek-reducing algorithms or ASPCTF(6). However, it provides superior performance over the seek-reducing algorithms for some sub-saturation workloads. For example, figure 5.21a shows that ASPTF(6) has up to 18% lower mean response times in the 0.4-0.9 range of scaling factors for the *Report* trace.

Although ASPCTF(6) always has better starvation resistance than SPCTF, the relative performance of these two algorithms (as measured by mean response times) depends on the individual workload. In some cases, the aging factor improves both starvation resistance and performance by maintaining the ordering of sequential requests issued in logically ascending

order. ASPTF(6) and ASPCTF(6) provide better performance than SPTF and SPCTF, respectively, for most sub-saturation *Cello*, *Order*, and *Report* workloads.

5.2.2 Full System Traces

This section presents performance data for scheduling algorithms using per-request priority information. Section 5.1.3 demonstrates that concatenation of sequential write requests is the only effective sequential stream optimization for scheduling the full system traces. All of the schedulers in this section therefore use concatenation of sequential writes.

Scheduling by Logical Block Number

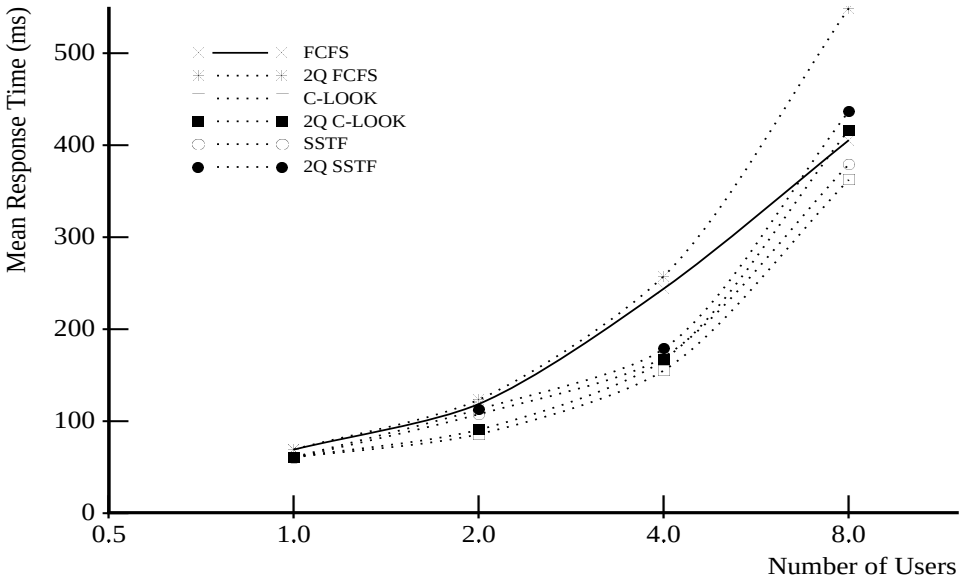
Figure 5.50 presents the mean response times and mean non-compute times for 2Q FCFS, 2Q C-LOOK, and 2Q SSTF scheduling of the *SynRGen* workloads. The 2Q algorithms always give priority to time-critical and time-limited requests (see section 4.3.2). These algorithms therefore have a tendency to starve time-noncritical requests for all but the lightest workloads. Figure 5.51 displays the mean response times measured for the three request classes when the scheduler uses a 2Q C-LOOK algorithm. The difference in mean response times between the request classes increases with the workload intensity.

By giving priority to requests on which the application and file system processes are waiting, the 2Q algorithms produce lower mean non-compute times. Figure 5.50b shows that the mean non-compute times for the 4-user and 8-user workloads are approximately 3% and 16% lower, respectively, for the seek-reducing 2Q algorithms. The difference between the performance of the FCFS, C-LOOK, and SSTF algorithms diminishes to less than 1% for 2Q scheduling.

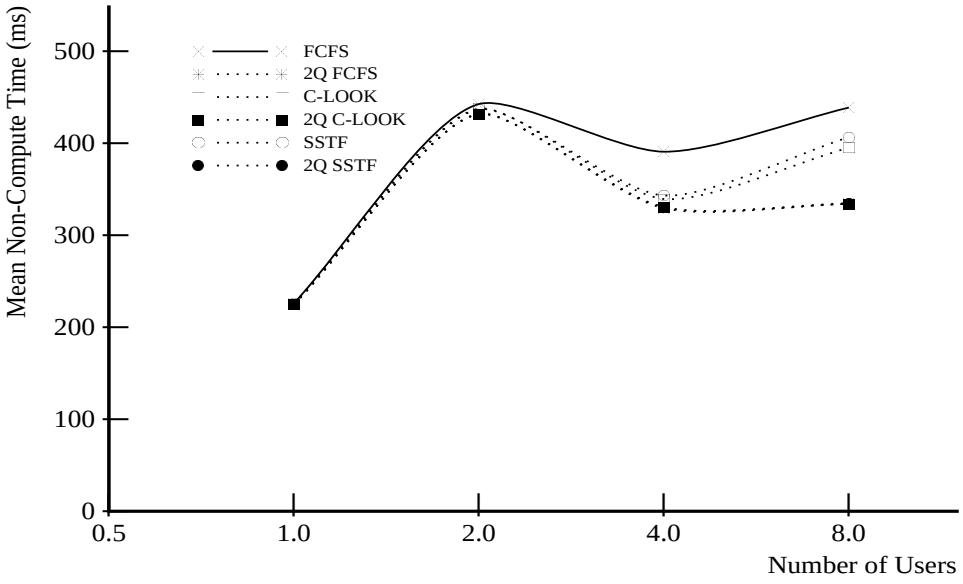
Figure 5.52 shows the mean response times and application run times for LBN-based 2Q scheduling of the *Compress* workload. While mean response times increase over an order of magnitude, run times drop slightly for 2Q scheduling of FCFS (2.9%), C-LOOK (0.2%), and SSTF (0.8%). This workload consists of separate sequential streams of read and write requests scattered across the disk. The read requests are mostly time-limited (with some time-critical requests), while the write requests are almost all time-noncritical. Giving priority to a fraction of the total set of requests defeats much of the seek-reducing behavior of C-LOOK and SSTF. For example, switching from C-LOOK.CW to 2Q C-LOOK.CW increases the mean seek distance from 126 cylinders to 563 cylinders. The mean seek time more than doubles as a result, increasing from 3.55 ms to 8.48 ms. FCFS, which does not attempt to minimize seek delays, benefits the most from 2Q scheduling.

Scheduling with Full Knowledge

Figure 5.53 displays performance data for 2Q SPTF and 2Q SPCTF scheduling of the *SynRGen* workloads. The behavior of these algorithms is similar to the behavior of the LBN-based 2Q algorithms. Mean response times grow rapidly as the workload increases, while mean non-compute times drop by up to 13.3%, matching the performance of 2Q FCFS. Therefore, it is far more important to exploit system-level information than minimize mechanical latencies for a *SynRGen* workload of up to 8 users.



(a) Mean Request Response Time



(b) Mean Non-Compute Time

Figure 5.50: *Synrgen*: LBN-Based 2Q.CW Algorithm Performance

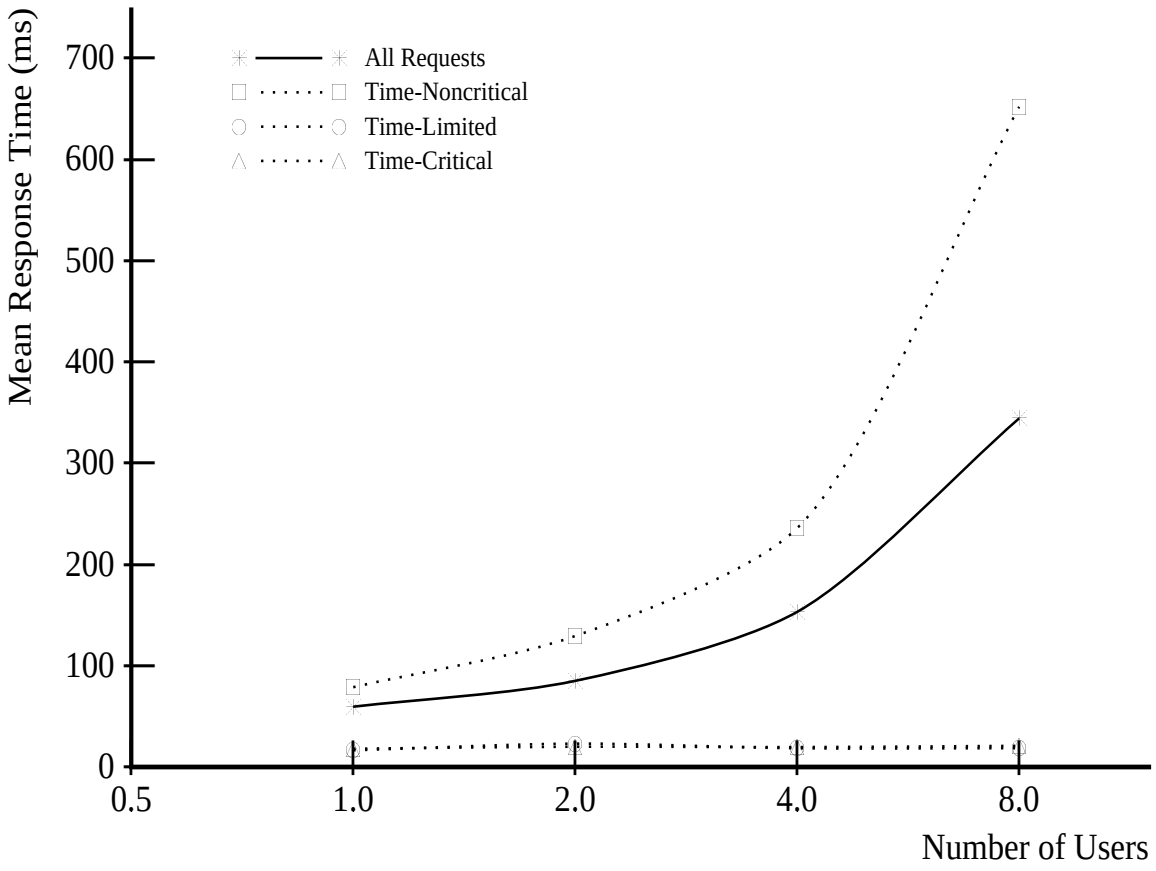
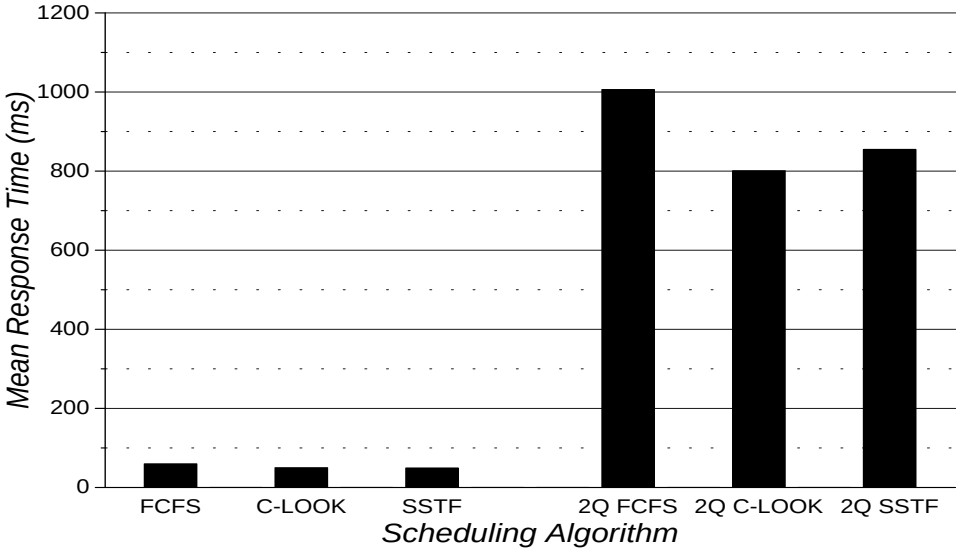
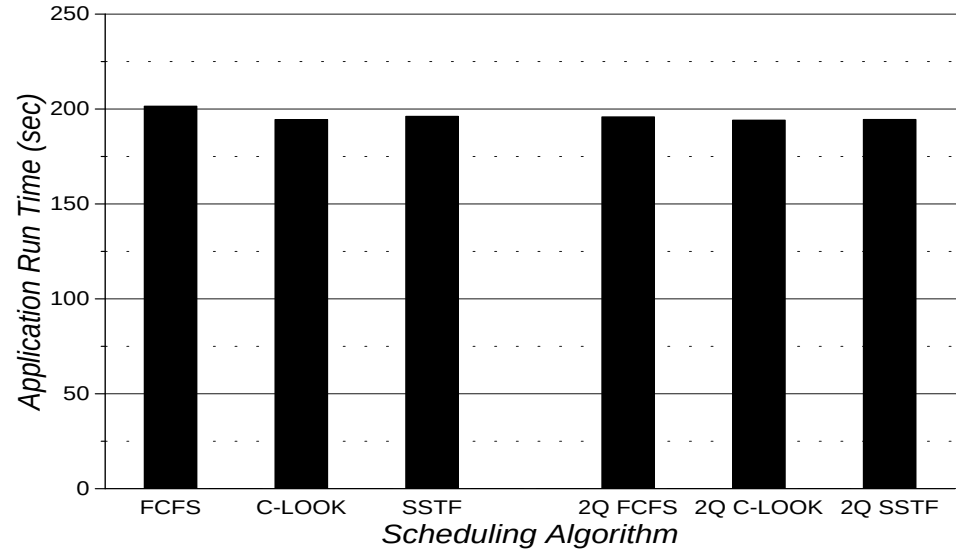


Figure 5.51: *Synrgen*, 2Q CLOOK.CW: Mean Response Times by Request Class

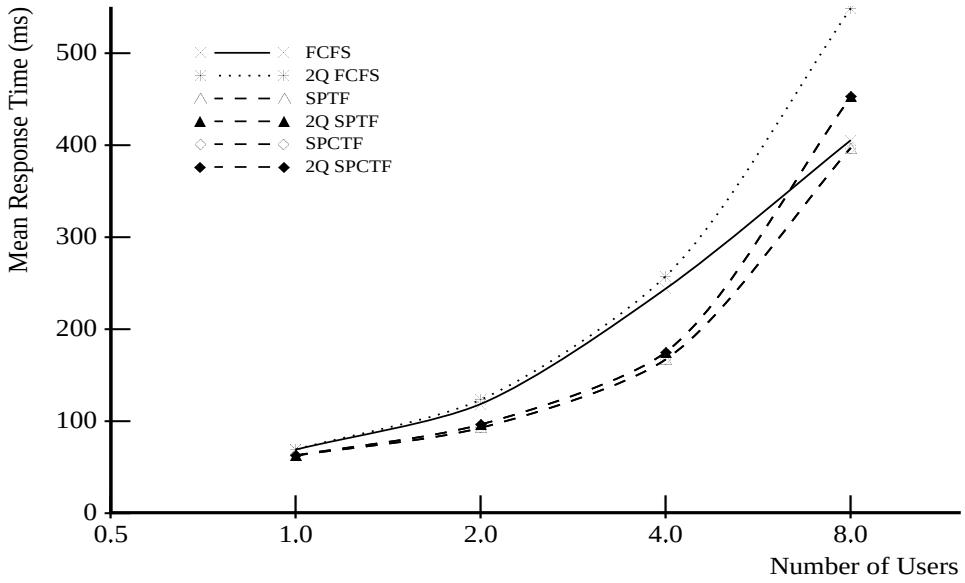


(a) Mean Response Time

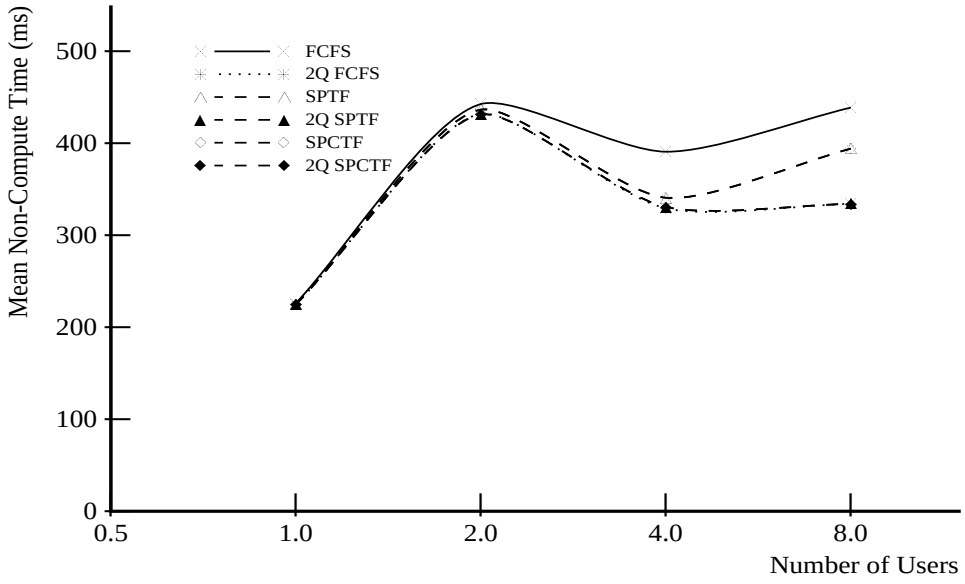


(b) Application Run Time

Figure 5.52: *Compress*: LBN-Based 2Q.CW Algorithm Performance



(a) Mean Request Response Time



(b) Mean Non-Compute Time

Figure 5.53: *Synrgen*: Full-Knowledge 2Q.CW Algorithm Performance

Section 5.1.3 describes how the performance of SPTF-based algorithms is below that of LBN-based algorithms for single-queue scheduling of the *Compress* workload. Figure 5.54 displays performance data for two 2Q full-knowledge schedulers, showing that the SPTF-based 2Q algorithms are slightly superior to both FCFS and LBN-based 2Q scheduling. 2Q SPTF and 2Q SPCTF have application run times that are over 1% lower than C-LOOK's run time. Also in contrast to the single-queue schedulers, 2Q SPCTF is slightly faster than 2Q SPTF. Since the scheduler is already giving priority to the time-critical and time-limited read requests, adding cache sensitivity just helps the scheduler reorder multiple outstanding read requests (a rare occurrence). As the SPTF-based 2Q algorithms outperform FCFS by 2%, a high-performance scheduler for *Compress* can therefore exploit both system-level and hardware-specific information.

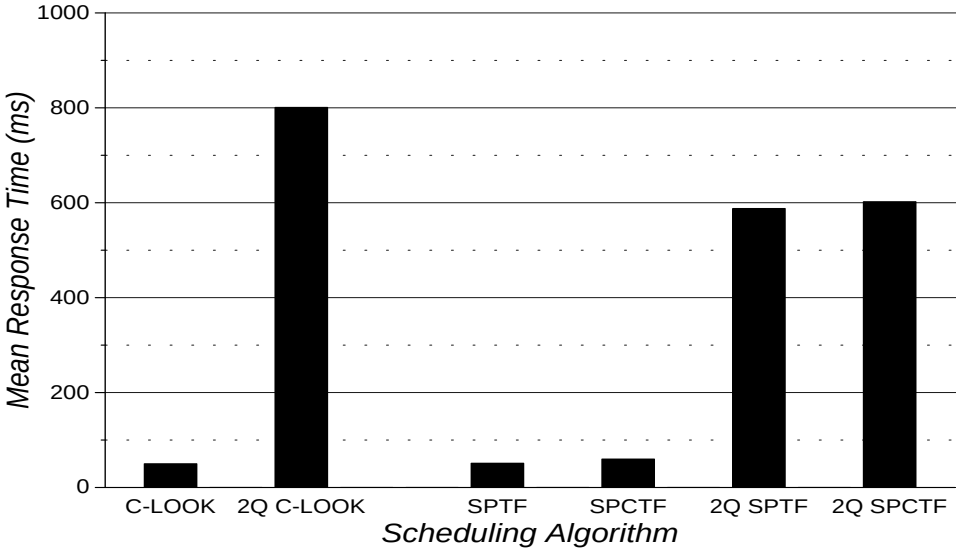
Scheduling with FCFS Command Queued Disks

Figures 5.55–5.59 show performance metrics for simulations of the full system traces using FCFS command queued disks with varying maximum queue lengths and command queueing configurations. To minimize mean response times, a command queue length of two is best for *SynRGen*, and a command queue length of sixteen (or more) is optimal for *Compress*. However, the application-specific metrics show that the best overall system performance occurs when command queueing is disabled. This is because a host-based, priority-sensitive scheduler loses the flexibility to reorder pending requests once it issues them to a FCFS command-queued disk. For example, assume that a disk has a maximum command queue length of two. If a host-based 2Q scheduler issues one high priority request and one low priority request to a disk, the scheduler cannot change the order of service if another high priority request arrives before the first high priority request is complete. For a disk without a command queue, the low priority request would never have been issued, and the scheduler would be free to issue the second high priority request as soon as the first completed.

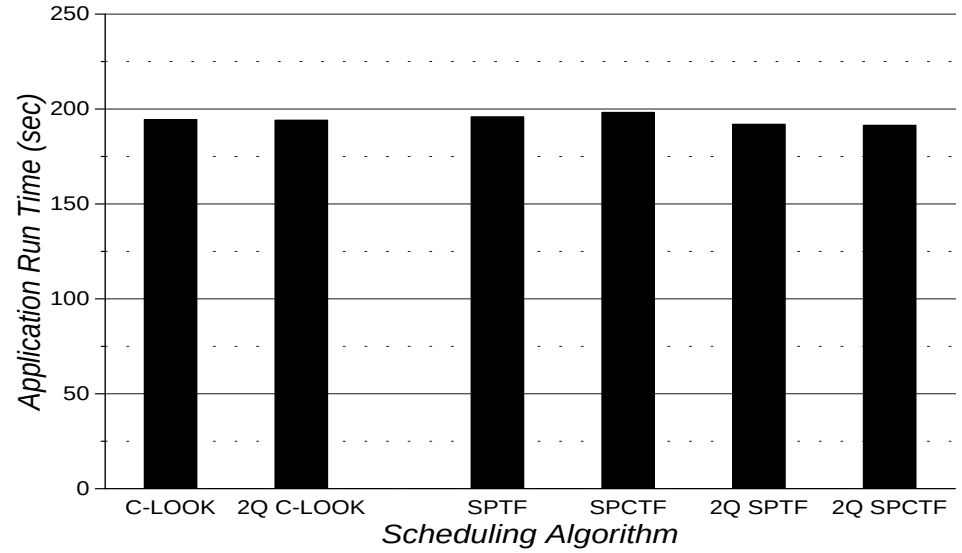
5.2.3 Summary of Conclusions

Adding age-sensitivity to SPTF-based algorithms improves starvation resistance (i.e., the squared coefficient of variation). By correctly "weighting" aging information, the improvement in starvation resistance does not necessarily degrade performance. In some cases, age-sensitivity actually improves mean response times by maintaining the ordering of sequential requests issued in logically ascending order.

For experiments using the full system traces, 2Q scheduling gives precedence to time-critical and time-limited requests (over time-noncritical requests). This results in enhanced system performance, as measured by the application-specific metrics. At the same time, 2Q algorithms produce much larger mean request response times (e.g., over an order of magnitude greater for *Compress*). This demonstrates the importance of including accurate host feedback in disk scheduling experiments; traditional disk subsystem analysis (using only subsystem performance metrics) would label 2Q scheduling as a bad design choice. Host-based 2Q scheduling works best if the disks in the subsystem do not have command queueing. Even a very short command queue length (e.g., 2 or 4) degrades overall system performance (albeit improving mean response times).

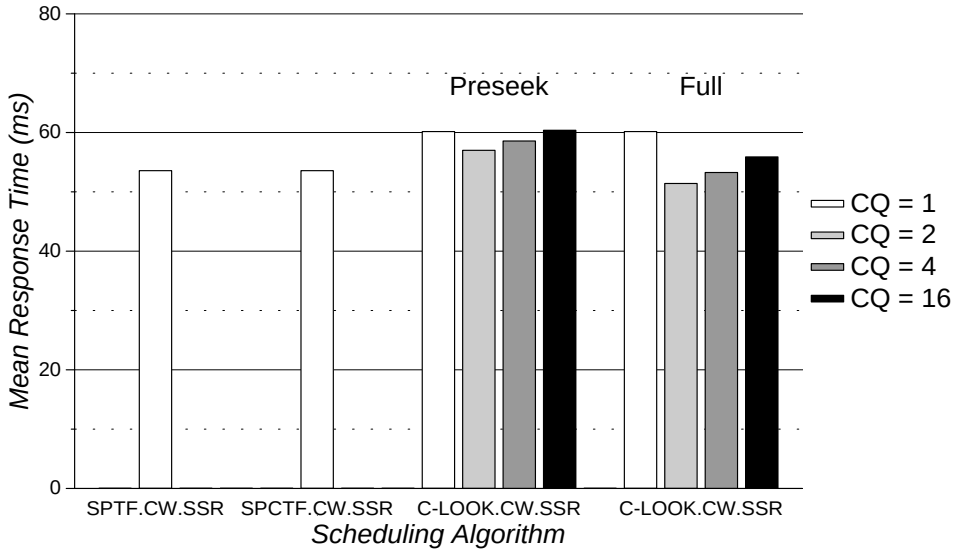


(a) Mean Response Time

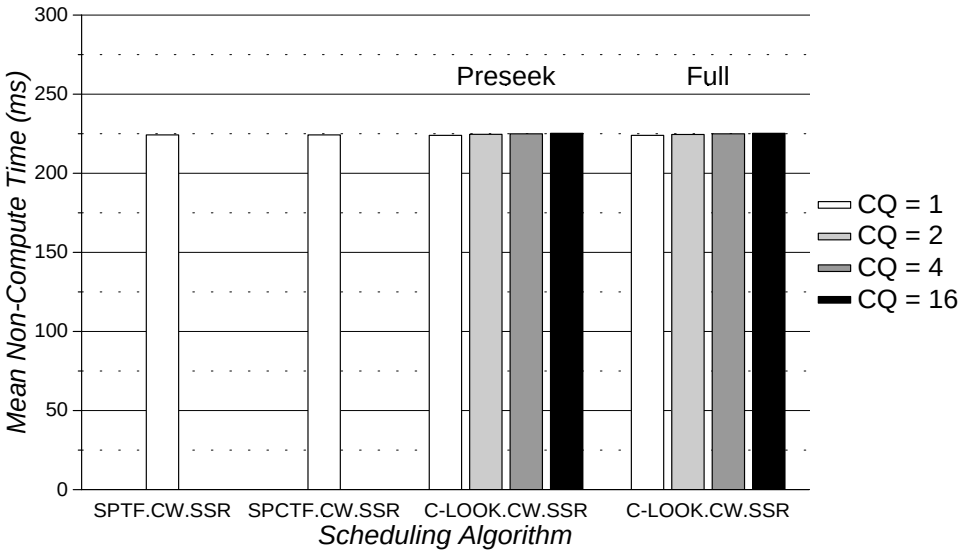


(b) Application Run Time

Figure 5.54: *Compress*: Full-Knowledge 2Q.CW Algorithm Performance

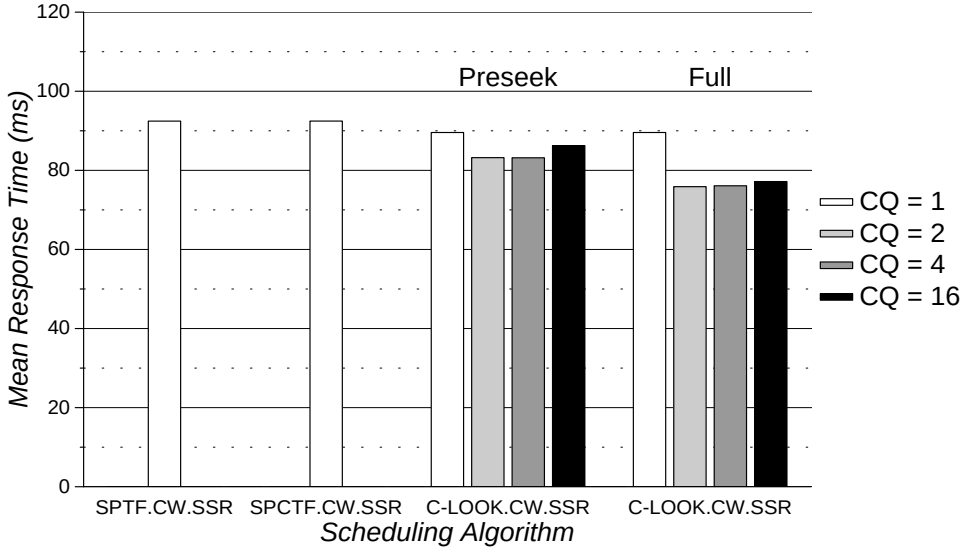


(a) Mean Response Time

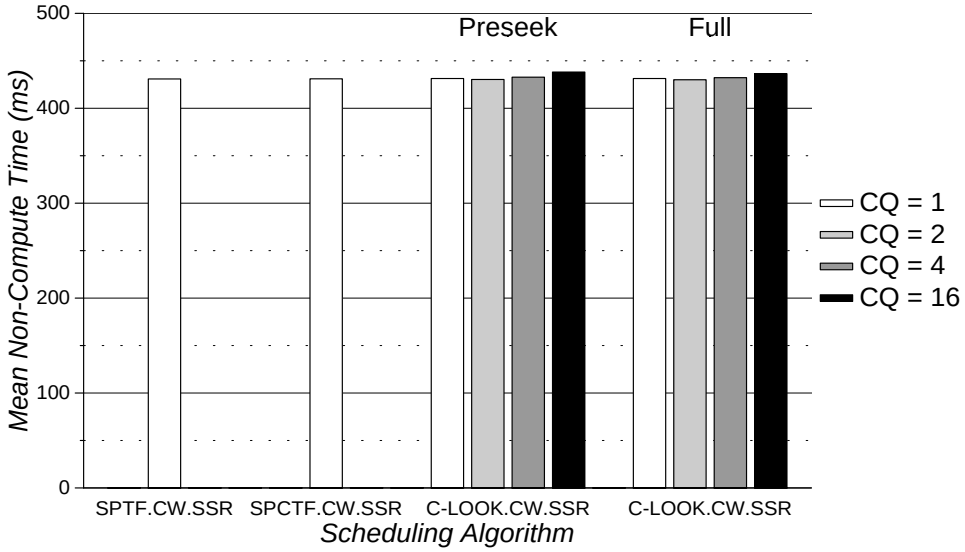


(b) Mean Non-Compute Time

Figure 5.55: *SynRGen*, 1 User: 2Q Scheduling Algorithm Performance for Different Command Queue Lengths

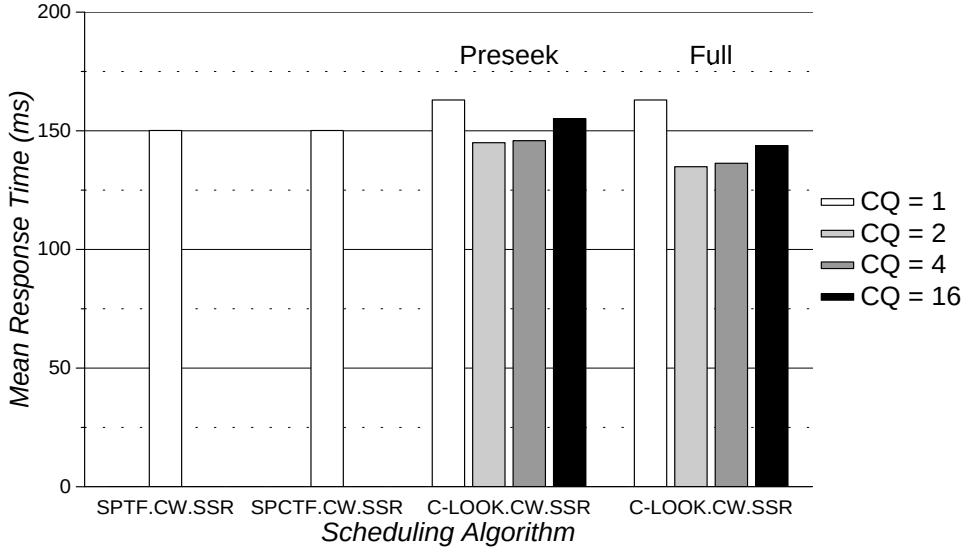


(a) Mean Response Time

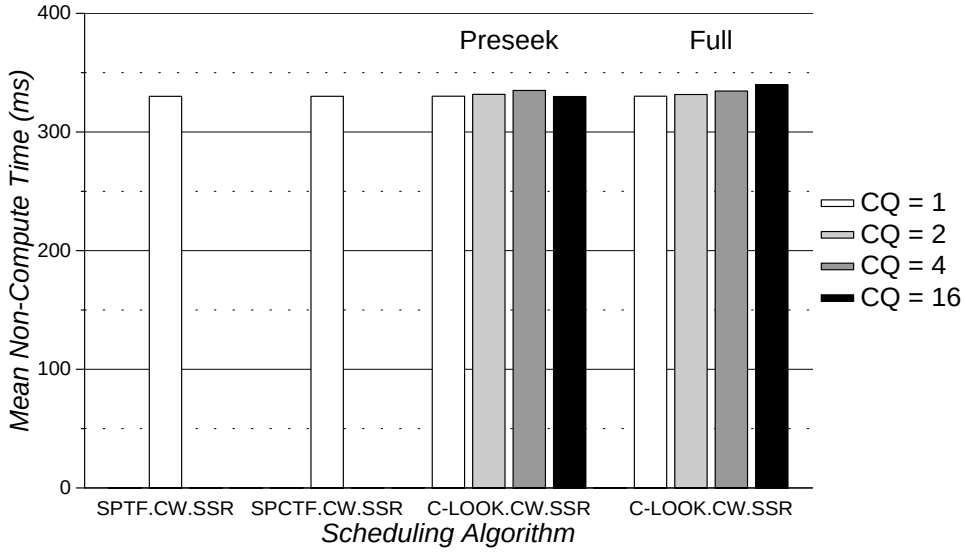


(b) Mean Non-Compute Time

Figure 5.56: *SynRGen*, 2 Users: 2Q Scheduling Algorithm Performance for Different Command Queue Lengths

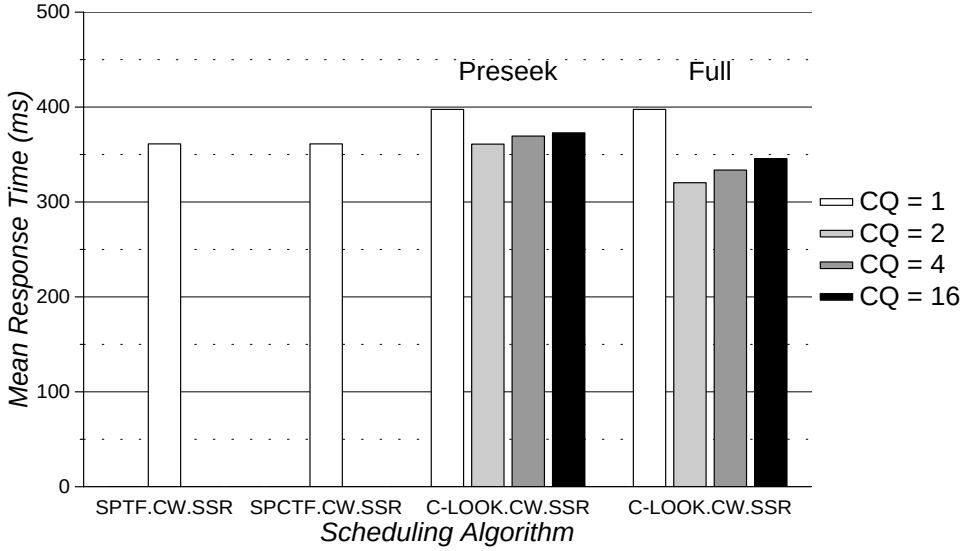


(a) Mean Response Time

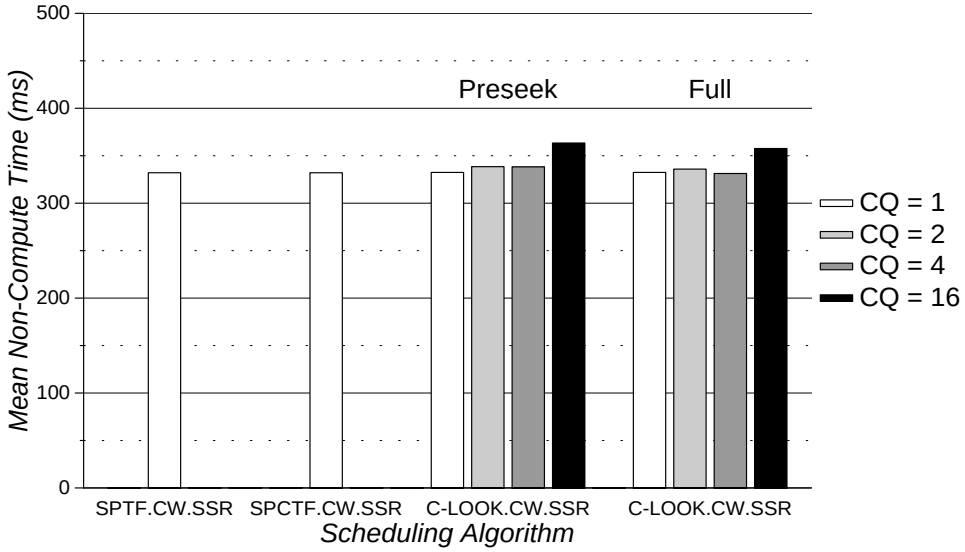


(b) Mean Non-Compute Time

Figure 5.57: *SynRGen*, 4 Users: 2Q Scheduling Algorithm Performance for Different Command Queue Lengths

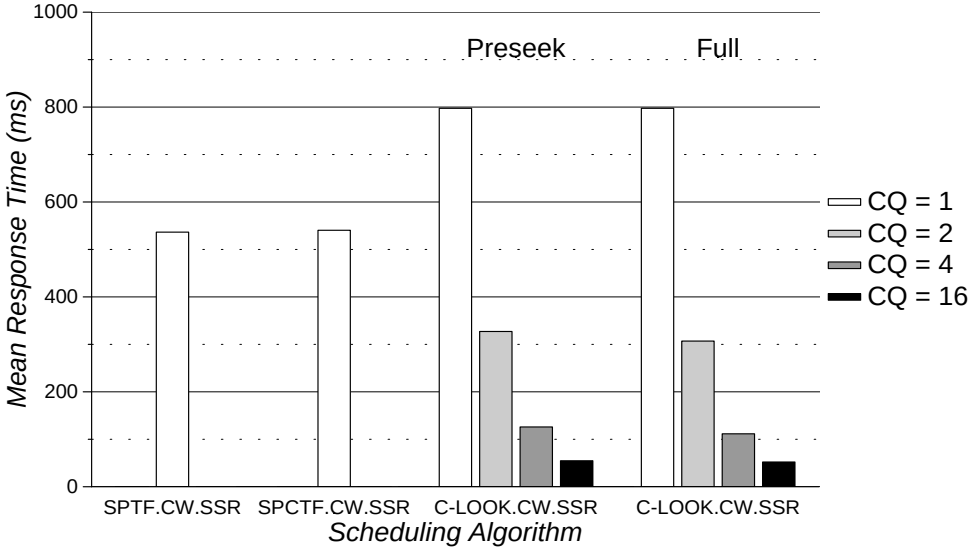


(a) Mean Response Time

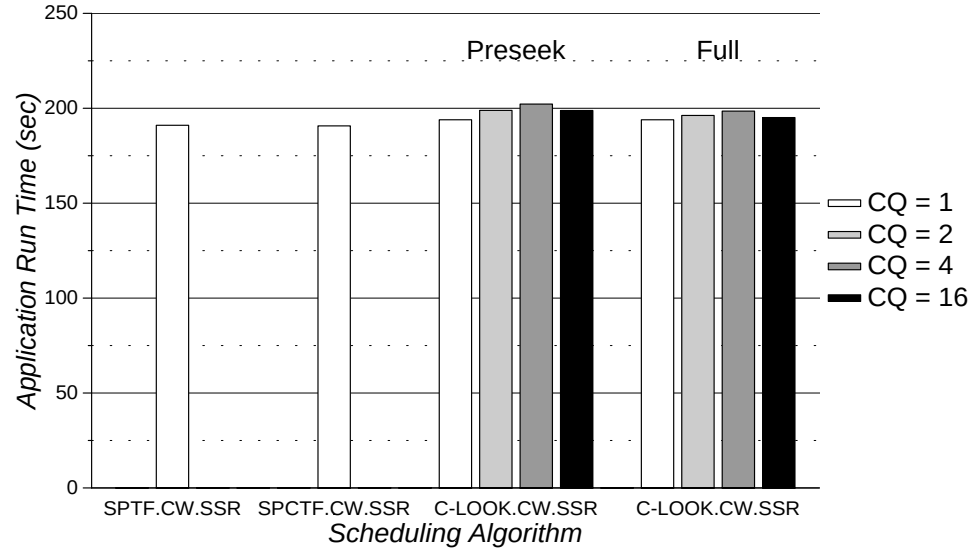


(b) Mean Non-Compute Time

Figure 5.58: *SynRGen*, 8 Users: 2Q Scheduling Algorithm Performance for Different Command Queue Lengths



(a) Mean Response Time



(b) Application Run Time

Figure 5.59: *Compress*: 2Q Scheduling Algorithm Performance with Different Command Queue Lengths

CHAPTER 6

Centralized Scheduling at the Disk

Modern disk drives contain powerful embedded microprocessors to handle peripheral bus interfaces, interpret and service incoming requests, and manage segmented data caches. In addition, disk drive controllers can maintain and reorder on-board queues of pending requests. A disk-based centralized scheduler has the advantage of easy and immediate access to the complete set of disk configuration and state information. On-board firmware (and hardware) can be tuned to exploit the maximum amount of concurrency between requests in service at a disk. On the other hand, system-level scheduling information must be communicated from the host (e.g., using additional fields in the request control blocks). More importantly, the on-board queue of pending requests is limited by the amount of available disk-level memory (excluding the data cache and firmware storage). The limited queue length impairs the performance of a disk-based scheduler; once the on-board queue is full, additional requests must be buffered by the host and are therefore not considered by the disk-based scheduler. Increasing the maximum queue length alleviates this problem at the cost of additional per-disk memory (and possibly computation) resources.

This chapter examines a variety of disk-based centralized scheduling algorithms. The experiments are partitioned based on the type of information used by the scheduler.

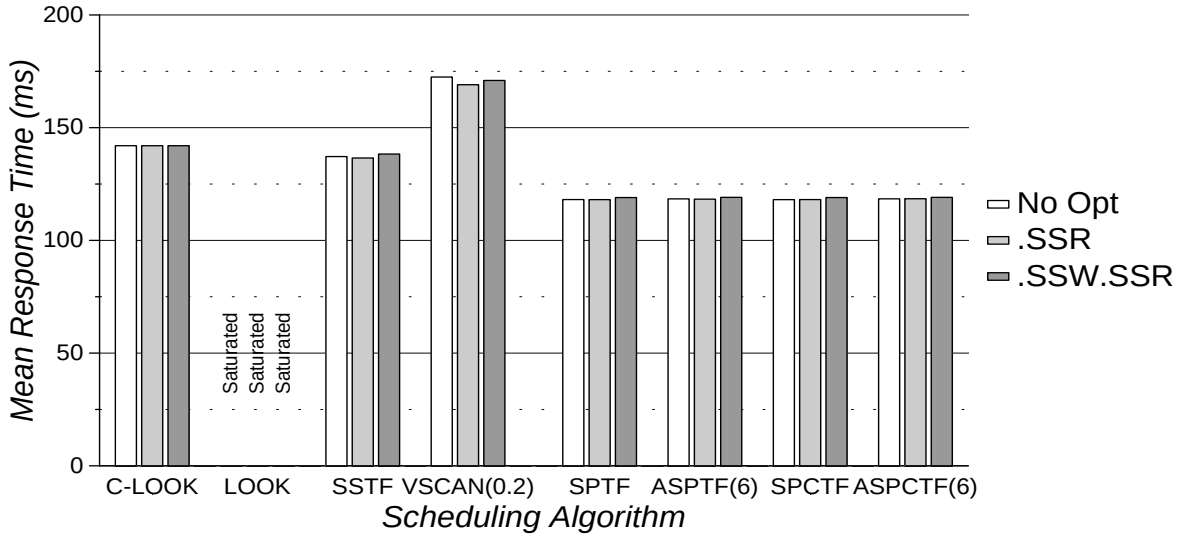
6.1 Scheduling with Information From "Below"

6.1.1 Disk Request Traces

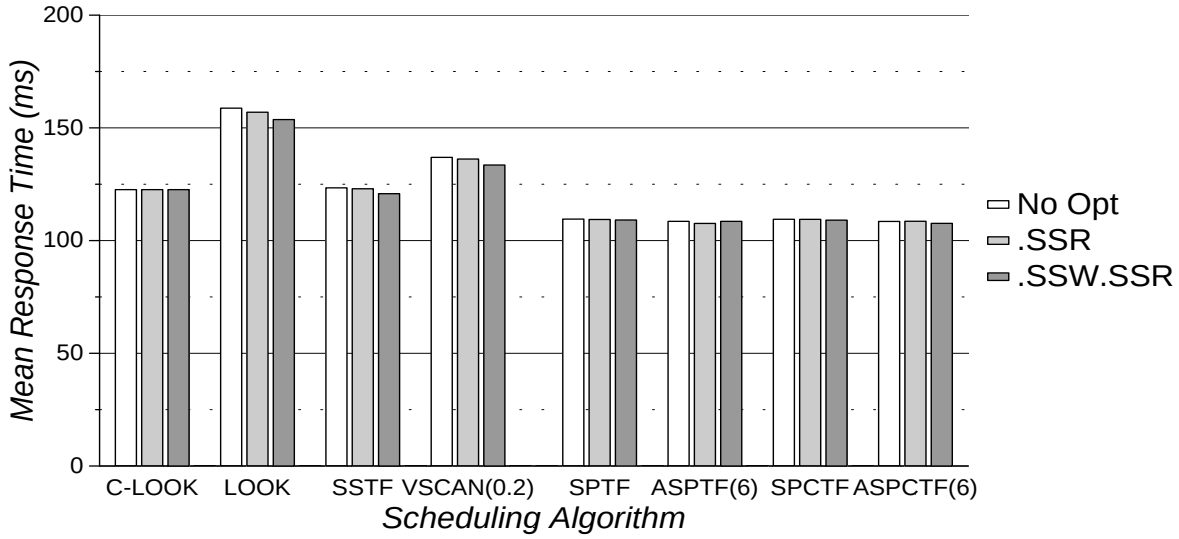
This section uses the six HP and DEC traces to study the effects of sequential stream optimizations and maximum command queue lengths on disk-based schedulers using only hardware-specific knowledge. It concludes with a comparison of disk-based and host-based centralized scheduling.

Sequential Stream Optimizations

As all experiments in this chapter use disks with preseeking capability, there is no advantage to concatenating requests as they arrive at a disk. The sequential scheduling optimization, on the other hand, can improve on-board cache hit rates and reduce mean response times. Figures 6.1–6.6 show the effects of this optimization on disk-based centralized schedul-

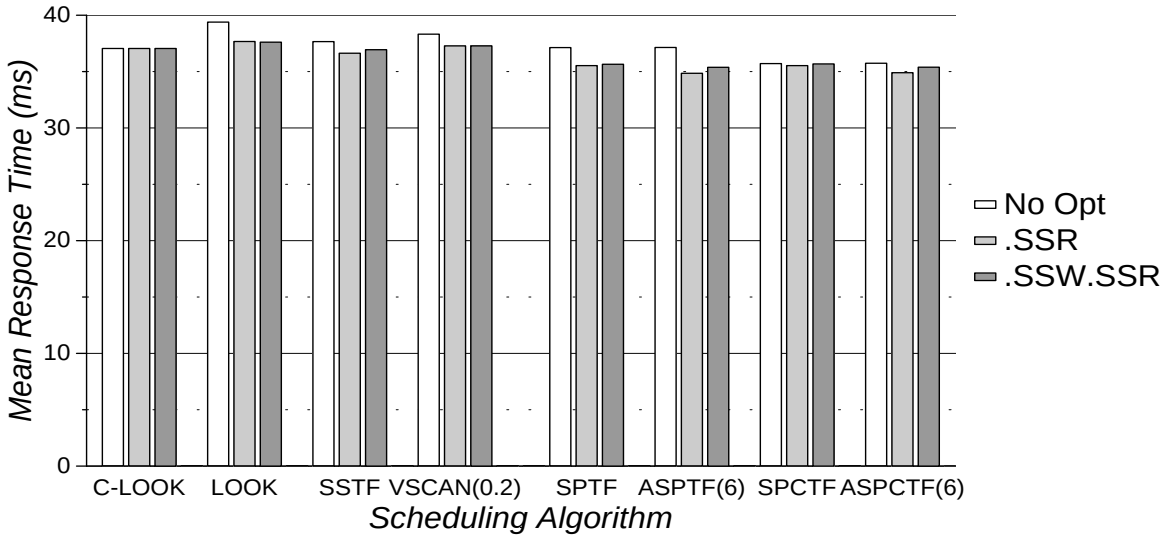


(a) Preseek Command Queueing

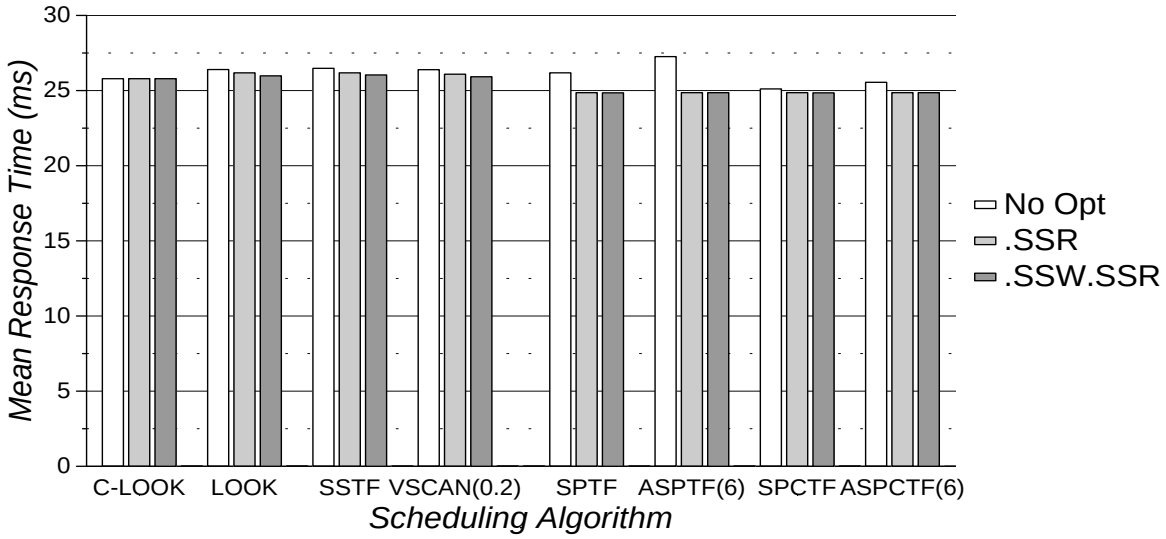


(b) Full Command Queueing

Figure 6.1: *Cello*, 1.75X: Sequential Scheduling Algorithm Performance

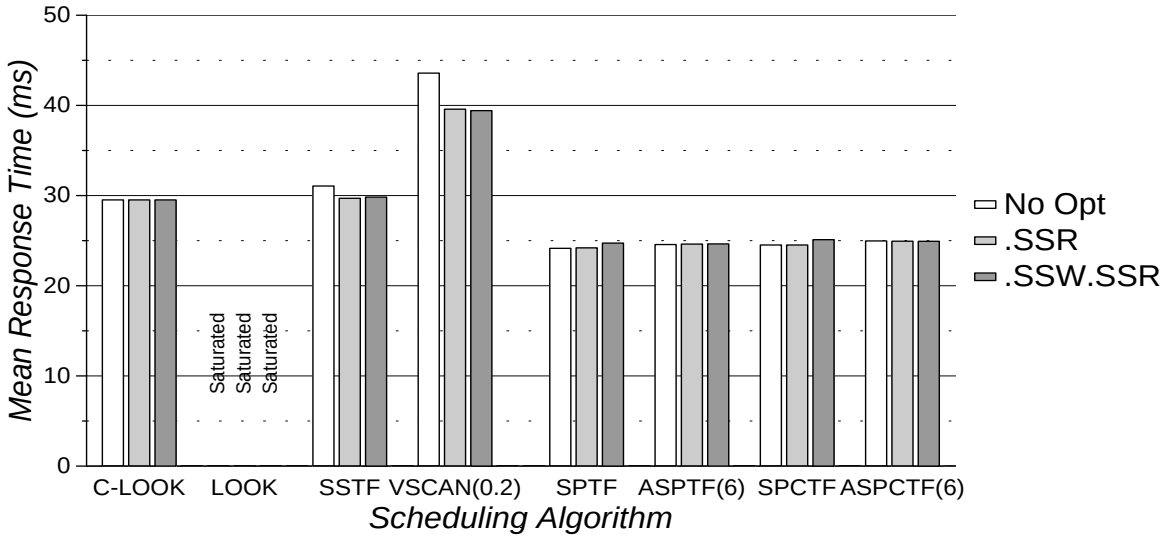


(a) Preseek Command Queueing

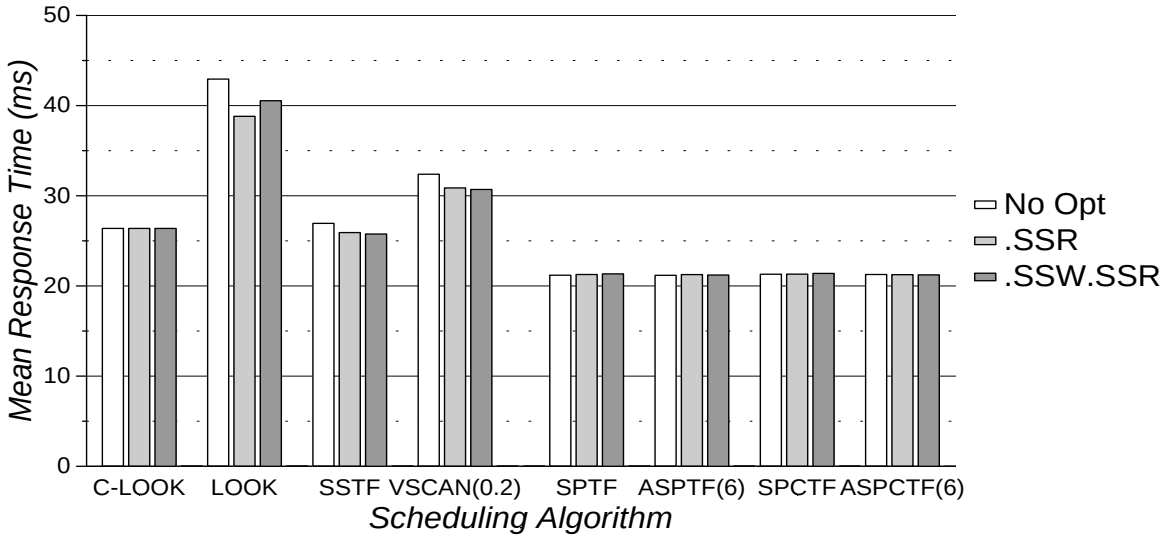


(b) Full Command Queueing

Figure 6.2: *Snake*, 1.25X: Sequential Scheduling Algorithm Performance

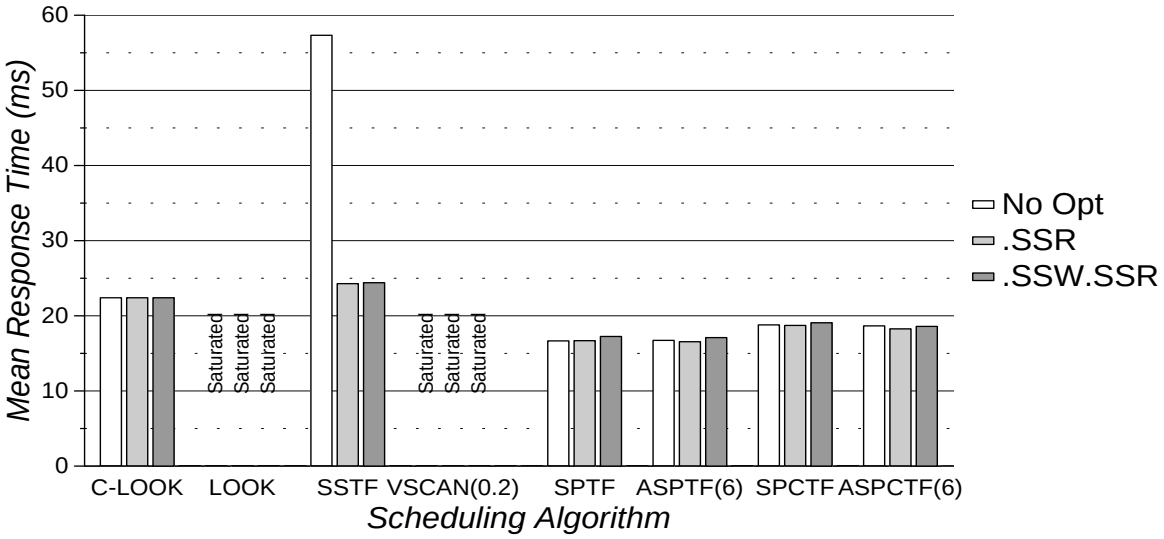


(a) Preseek Command Queueing

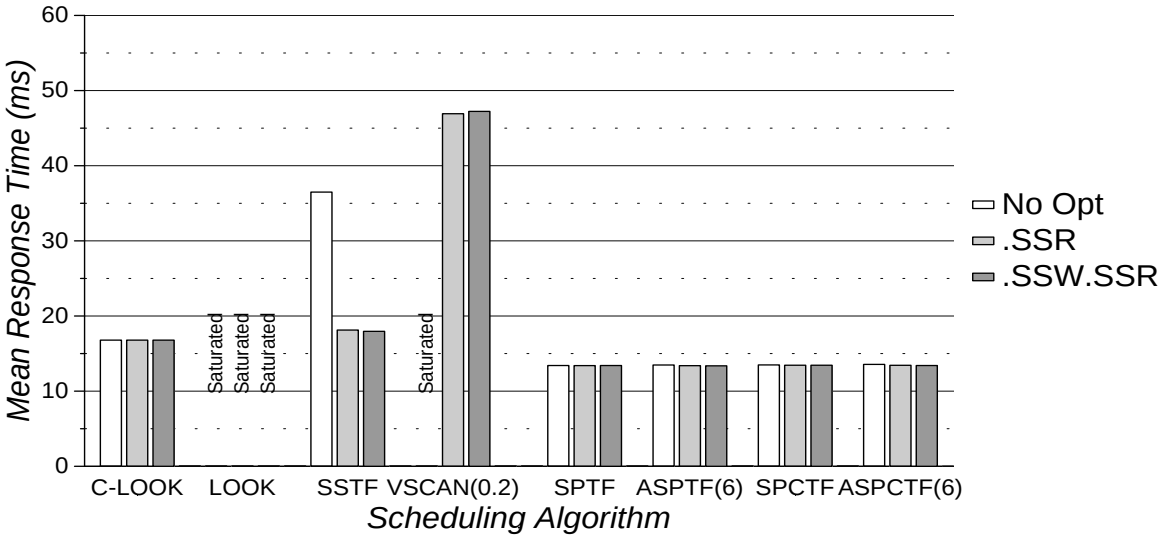


(b) Full Command Queueing

Figure 6.3: *Air-Rsv*, 2.5X: Sequential Scheduling Algorithm Performance

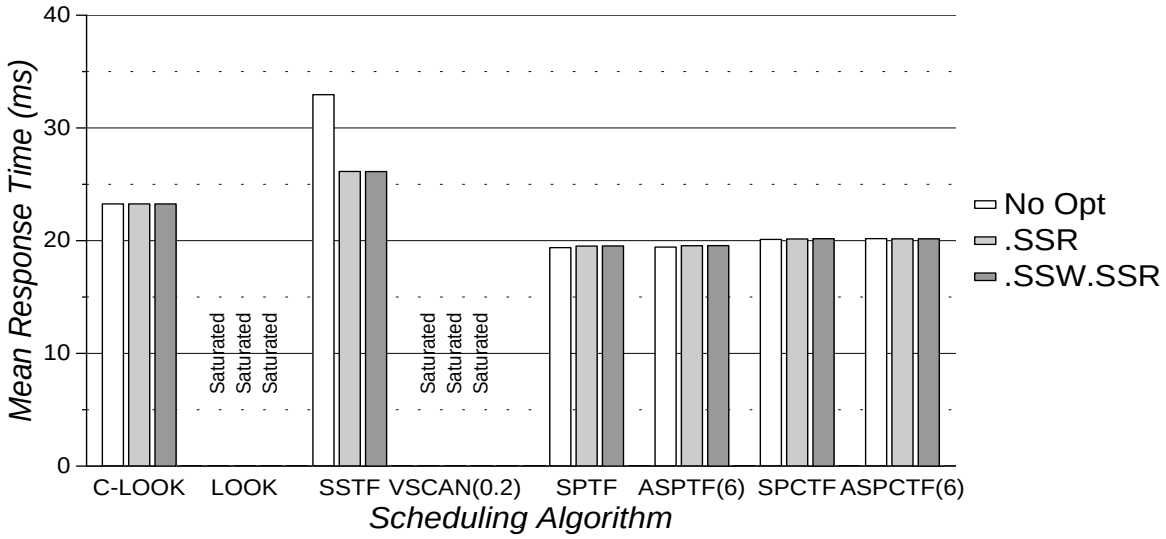


(a) Preseek Command Queueing

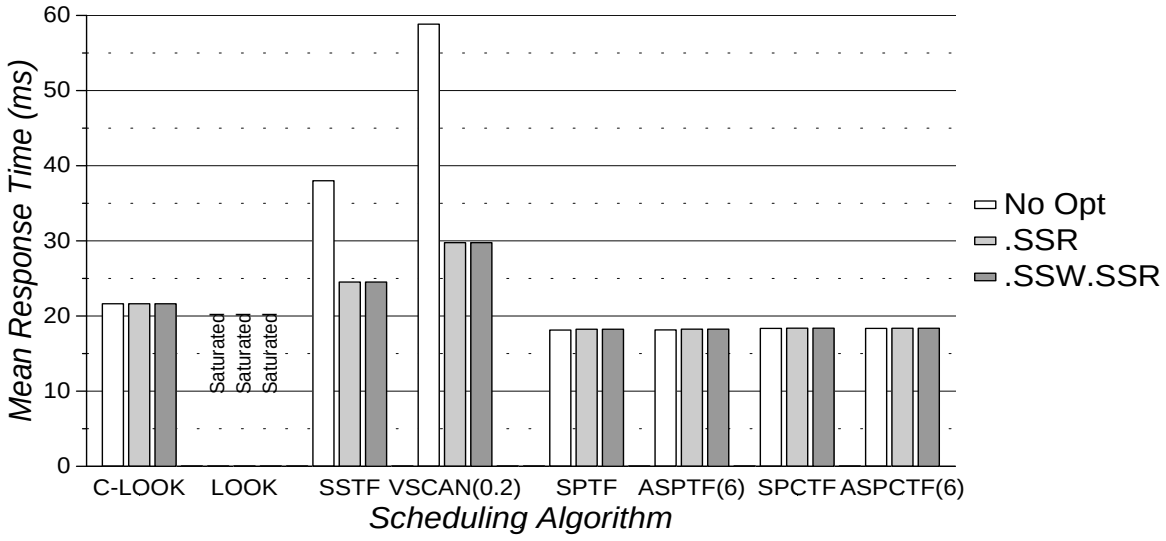


(b) Full Command Queueing

Figure 6.4: *Sci-TS*, 2.5X: Sequential Scheduling Algorithm Performance

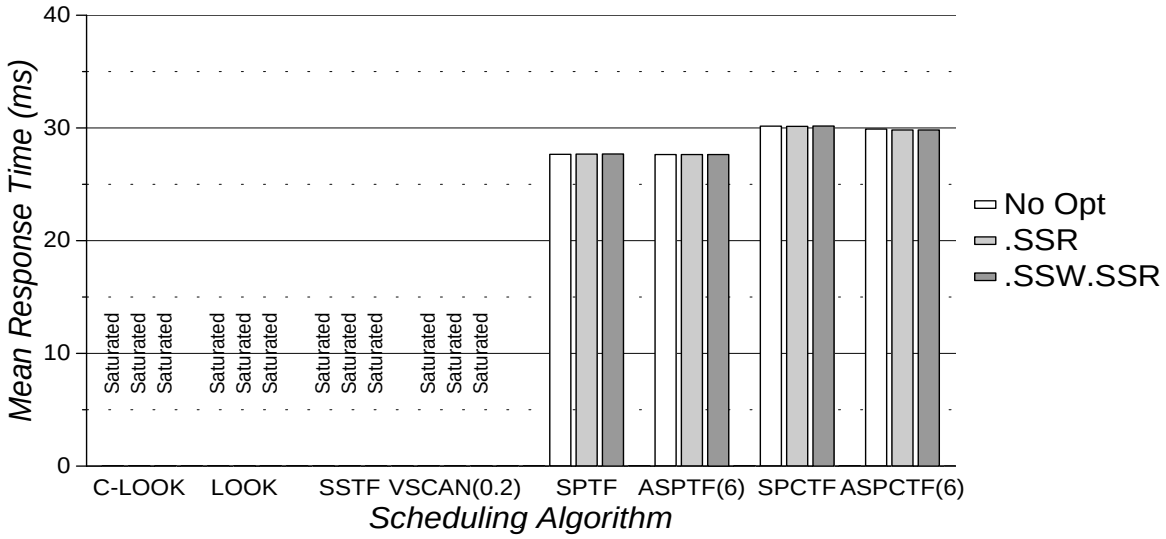


(a) Preseek Command Queueing

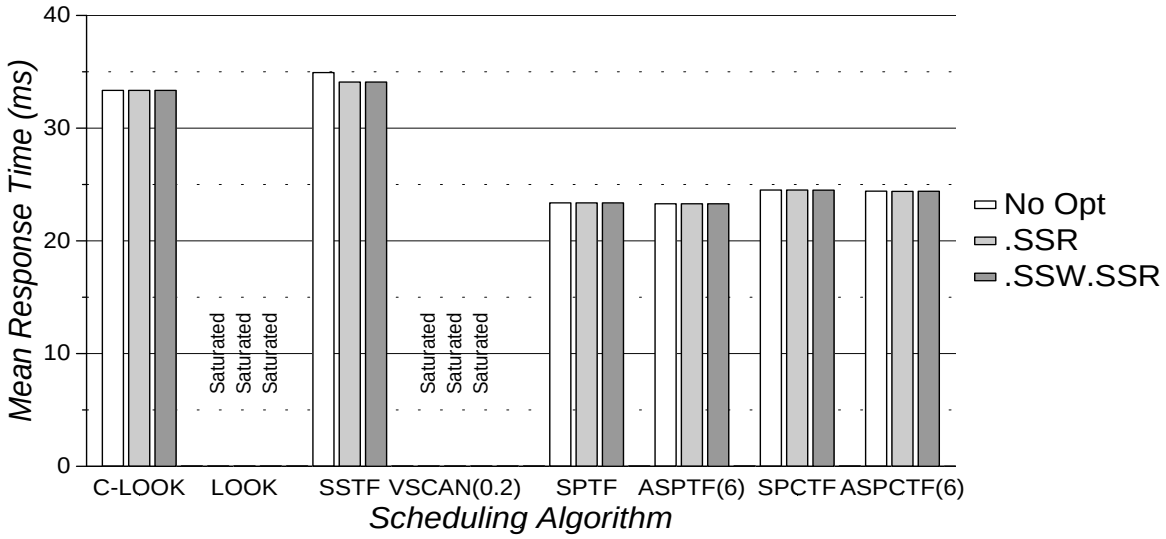


(b) Full Command Queueing

Figure 6.5: *Order*, 1.0X: Sequential Scheduling Algorithm Performance



(a) Preseek Command Queueing



(b) Full Command Queueing

Figure 6.6: *Report*, 1.0X: Sequential Scheduling Algorithm Performance

ing. The maximum command queue length was set to 128 (a value justified below) for all disks. The data points are from the "knee" of each trace's set of response time curves. For *Snake*, sequential scheduling of read requests decreases mean response times by up to 9%. For the other five traces, this optimization affects mean response times by 2% or less when using C-LOOK or the SPTF-based algorithms. This matches the general behavior observed for host-based schedulers with sequential scheduling of reads (see section 5.1.2).

Sequential scheduling of writes has a minimal effect for *Cello*, *Snake*, *Order*, and *Report* when using C-LOOK or the SPTF-based algorithms. For *Air-Rsv* and *Sci-TS*, sequential scheduling of write requests actually degrades performance by approximately 2% for the Preseeking command queueing configuration.[1]

The sequential scheduling optimization strongly affects the performance of LOOK, SSTF, and VSCAN(0.2). Section 5.1.2 discusses how these algorithms handle sequential streams of requests with and without the sequential scheduling optimization (see page 62). The sub-optimal sequential scheduling behavior of LOOK and VSCAN(0.2) is quite obvious in figure 6.1 and figures 6.3–6.6. In several cases, the use of these algorithms results in saturation of the disk subsystem at the selected trace scaling factors. Figures 6.4 and 6.5 show how sequential scheduling allows SSTF to attain performance much closer to that of C-LOOK for *Sci-TS* and *Order*. For *Snake* configurations with Preseek command queueing (see figure 6.2a) and *Air-Rsv* configurations with Full command queueing (see figure 6.3b), the sequential scheduling optimizations enable SSTF to marginally outperform C-LOOK (with 1–2% lower mean response times).

Comparison of Disk-Based Scheduling Algorithms

As was true for host-based scheduling, C-LOOK generally provides the best performance among the seek-reducing algorithms. For *Cello*, SSTF produces up to 4% lower mean response times. A few configurations using *Snake* and *Air-Rsv* also achieve better performance with SSTF (as noted above).

Across all six traces, the SPTF-based algorithms provide the lowest mean response times. In the case of *Report*, a disk subsystem using Preseek command queueing saturates at the selected trace scaling factor for all of the LBN-based algorithms. The full-knowledge algorithms, on the other hand, achieve mean response times of 27–30 ms.

For the four DEC traces, cache-sensitivity degrades the performance of the SPTF-based algorithms by up to 11%. This behavior matches that of SPTF-based scheduling of the *Compress* full system trace (see page 73 in section 5.1.3). As was true for *Compress*, the cache-sensitive algorithms produce higher mean response times for the DEC traces because of automated prefetching on full read hits. When prefetching is enabled only on partial read hits and read misses, SPCTF and ASPCTF(6) outperform SPTF and ASPTF(6), respectively. In fact, all algorithms experience a decrease in mean response times after this change (using the DEC traces), although the LBN-based algorithms maintain the same relative performance ordering.

The full-knowledge algorithms are also sensitive to any preference given to full read hits (over partial read hits). Cache-sensitive SPTF algorithms query the on-board cache model

[1] Section 5.1.2 explains how sequential scheduling of writes to a disk without write prebuffering can degrade performance (see page 51).

to determine if a pending request will hit in the cache. For host-based scheduling, the cache simply indicates either "hit" or "miss". For disk-based scheduling, the cache model also indicates if a read hit is "complete" or "partial" based on whether or not all of the requested sectors are present in the cache. The default scheduler configuration gives precedence to full hits over partial hits. Unfortunately, this design choice has a detrimental effect when scheduling sequential request streams.

Consider the situation when a disk is servicing a stream of sequential read requests. If a request arrives that is not part of the "current" stream and the cache indicates that it will be a full hit (in another cache segment), a cache-sensitive scheduler will give higher priority to that request over the next sequential request in the "current" stream (which will only be a partial hit unless the prefetching activity is far outpacing the servicing of requests). Even if all the sectors for both requests are present in the cache, there is no way for the cache model to indicate that a request is part of the "current" sequential stream and should therefore be given higher priority. This is an artifact of the simulator and does not necessarily reflect a common design choice for current disk drive controllers.[2]

The experiments on host-based SPTF scheduling given in section 5.1.2 do not exhibit the behavior discussed above. All comparisons between SPTF algorithms in that section refer to experiments that use disks without command queueing. As noted in section 4.1.2, experiments using disks without command queueing also use on-board data caches with one read segment (and one write segment). With only one read segment, all read hits occur within (or near) the "current" sequential stream.

Maximum Command Queue Length

The amount of available main memory bounds the pending request queue length for host-based schedulers. Similarly, the amount of on-board controller memory bounds the command queue length for a disk. This section examines the effects of limited command queue lengths on disk-based centralized scheduling.

Figures 6.7–6.12 present mean response times for points along the "knees" of the individual response time curves for C-LOOK and SPCTF. The bargraphs include data for Preseek and Full command queue configurations with maximum command queue lengths (labeled CQ) of 16, 128, and infinity. A maximum queue length of 16 is visibly suboptimal for most of the trace/scheduler combinations. For *Snake*, *Air-Rsv*, *Order*, and *Report*, a maximum queue length of 128 generally provides performance equivalent to an infinite queue length. Mean response times for *Cello* and *Sci-TS* increase by less than 4% when the maximum command queue length changes from infinity to 128. The single exception to the above observations is C-LOOK scheduling of *Report* with the Preseeking command queueing configuration. The mean response time measured for this combination exceeds 1 second (i.e., a saturated workload).

The last few generations of SCSI disks provide queue depths of at least 16, with some disks queueing as many as 64 requests [Seag93]. Since the SCSI-2 protocol provides 256 unique command tags [SCSI93], a maximum command queue length of 128 seems quite reasonable for future generations of disks.

[2]There is little or no documentation available regarding the exact implementation of request schedulers in modern disk drives.

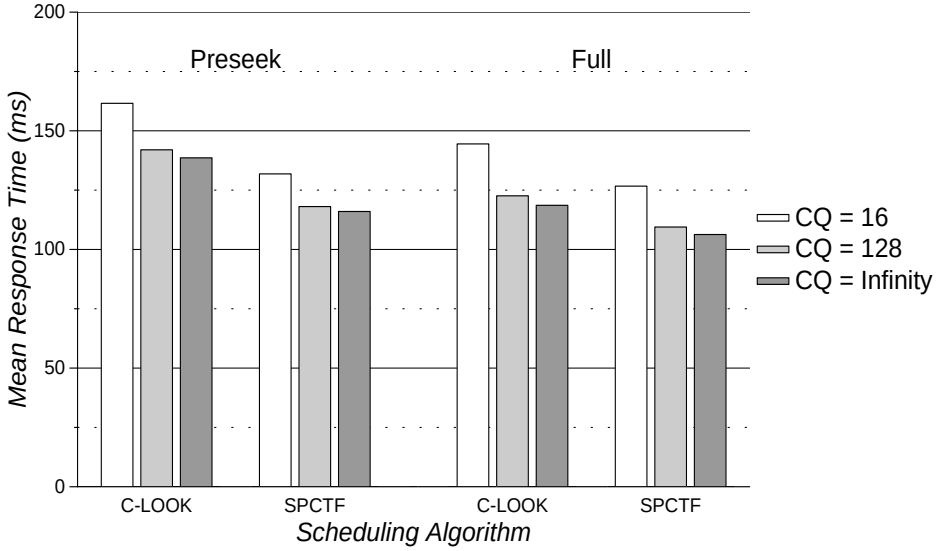


Figure 6.7: *Cello*, 1.75X: Algorithm Performance for Different Command Queue Lengths

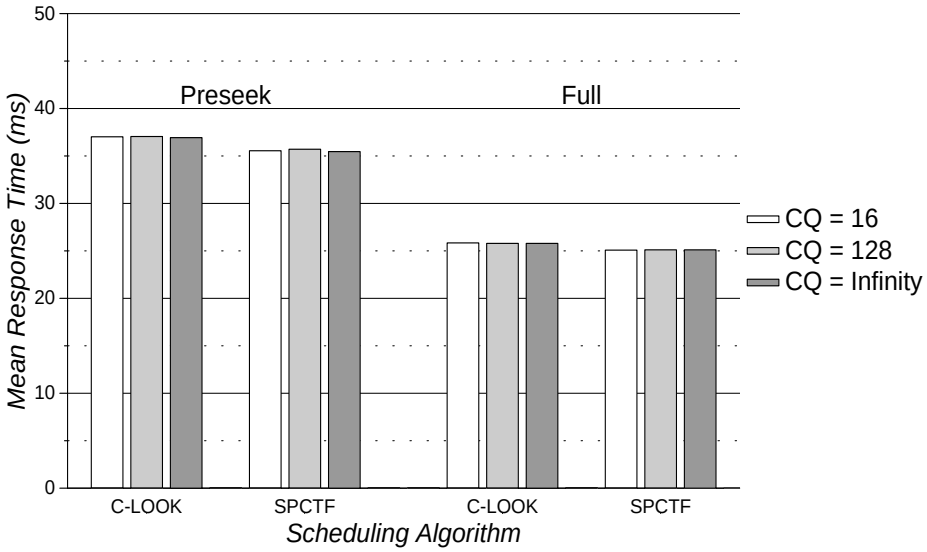


Figure 6.8: *Snake*, 1.25X: Algorithm Performance for Different Command Queue Lengths

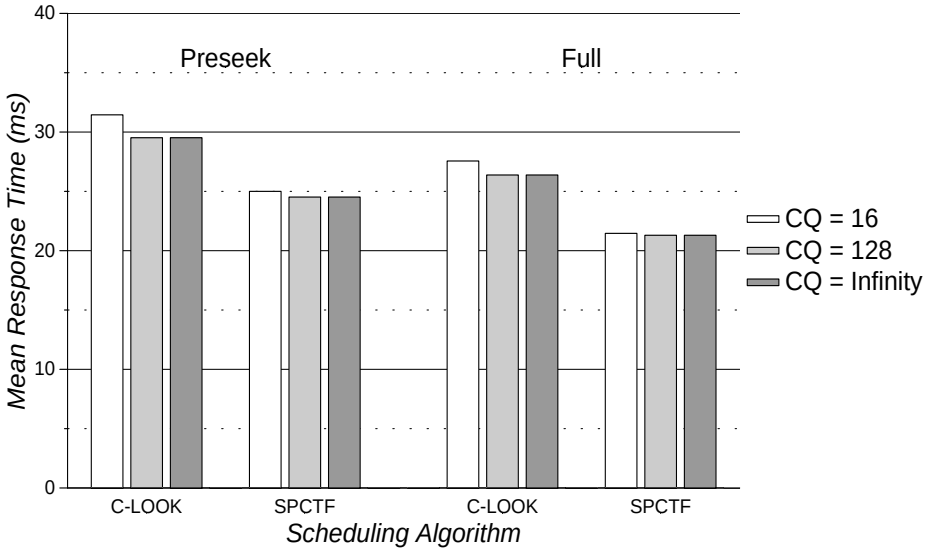


Figure 6.9: *Air-Rsv*, 2.5X: Algorithm Performance for Different Command Queue Lengths

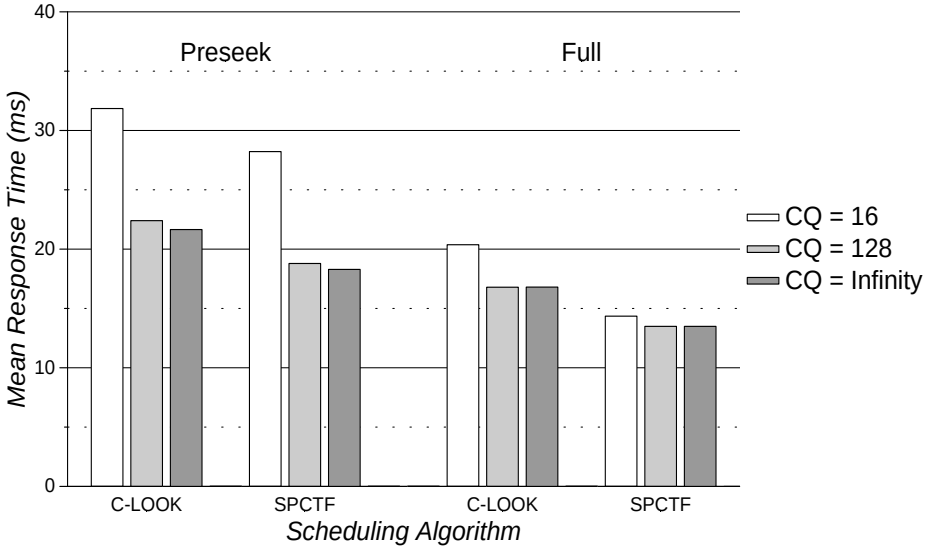


Figure 6.10: *Sci-TS*, 2.5X: Algorithm Performance for Different Command Queue Lengths

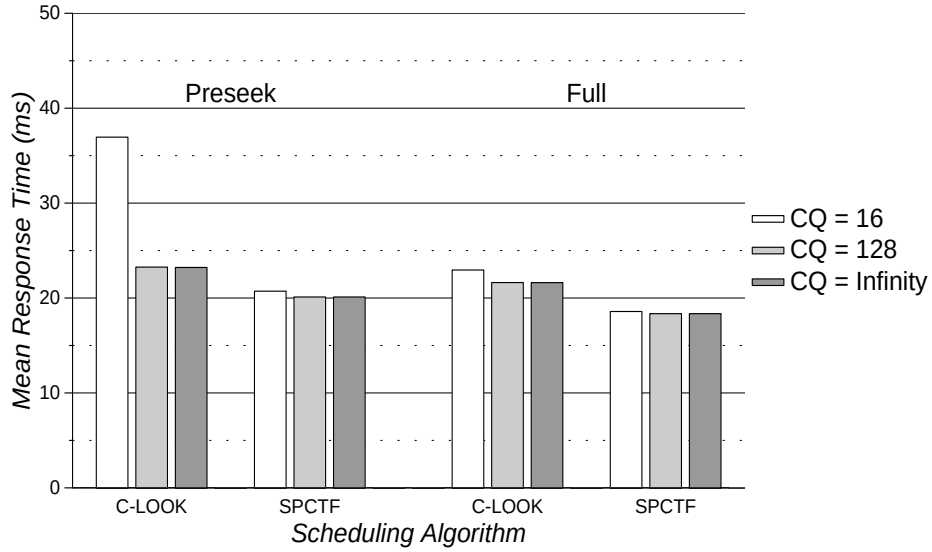


Figure 6.11: *Order*, 1.0X: Algorithm Performance for Different Command Queue Lengths

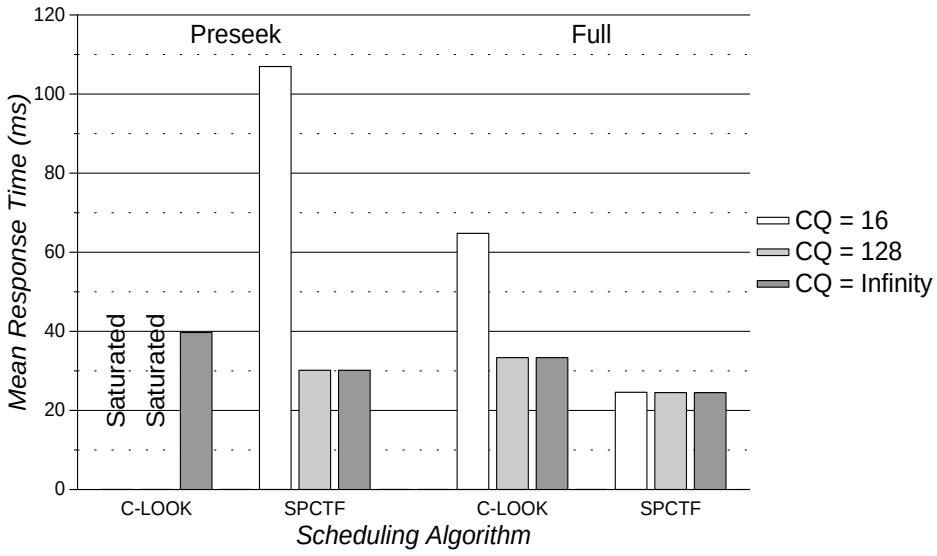


Figure 6.12: *Report*, 1.0X: Algorithm Performance for Different Command Queue Lengths

Trace	Scale Factor	Host-Based C-LOOK.CW.CR CQ = 2	Host-Based SPCTF.CW.CR CQ = 1	Disk-Based C-LOOK CQ = 128	Disk-Based SPCTF CQ = 128
Cello	1.75	**140.49**	**116.72**	141.99	118.07
Snake	1.25	**23.75**	**26.94**	37.05	35.71
Air-Rsv	2.5	30.89	33.56	**29.52**	**24.52**
Sci-TS	2.5	**20.63**	20.62	22.40	**18.79**
Order	1.0	23.90	23.96	**23.26**	**20.11**
Report	1.0	**40.51**	49.21	Saturated	**30.16**

(a) Preseek Command Queueing

Trace	Scale Factor	Host-Based C-LOOK.CW.CR CQ = 2	Host-Based SPCTF.CW.CR CQ = 1	Disk-Based C-LOOK CQ = 128	Disk-Based SPCTF CQ = 128
Cello	1.75	**101.86**	116.72	122.60	**109.45**
Snake	1.25	**22.68**	26.94	25.79	**25.11**
Air-Rsv	2.5	29.76	33.56	**26.38**	**21.30**
Sci-TS	2.5	18.48	20.62	**16.79**	**13.49**
Order	1.0	23.16	23.96	**21.63**	**18.35**
Report	1.0	39.86	49.21	**33.35**	**24.51**

(b) Full Command Queueing

Table 6.1: Mean Response Times (ms) for "Reasonable" Centralized Schedulers

Comparison with Scheduling at the Host

The optimal location for a centralized scheduler is completely dependent on the workload and system configuration, but some general observations can be made based on the experimental results. Table 6.1 lists the mean response times for the six HP and DEC traces using eight different combinations of centralized schedulers and command queueing configurations. These combinations represent "reasonable" implementations of centralized scheduling specifically tuned to service all six traces.

For configurations with Preseek command queueing, the optimal choice for host-based versus disk-based scheduling is evenly split for the six traces (see table 6.1a). A host-based scheduler works best for *Cello* and *Snake*; a disk-based scheduler works best for *Air-Rsv* and *Order*; and for *Sci-TS* and *Report*, the best scheduler is host-based for C-LOOK and disk-based for SPCTF. Host-based scheduling reduces the total amount of command and completion overhead by concatenating sequential requests. Disk-based scheduling enhances

inter-request concurrency through longer on-board command queues (especially for SPCTF). Furthermore, SPCTF scheduling is more accurate when performed at the disk, as host-based schedulers must deal with unpredictable bus delays after selecting a request to service.

Host-based scheduling performance improves when Full command queueing is enabled, but disk-based schedulers gain the most from the upgrade in disk controller complexity. Table 6.1b shows that disk-based SPCTF schedulers for *Cello* and *Snake* and C-LOOK schedulers for *Sci-TS* and *Report* outperform their host-based counterparts when Full command queueing is enabled; only C-LOOK scheduling for *Cello* and *Snake* is best performed by the host.

6.1.2 Full System Traces

This section examines the performance of disk-based centralized scheduling for the NCR traces. Only hardware-specific information is used by the scheduler, but both subsystem and system (i.e., application-specific) metrics are reported.

Sequential Scheduling

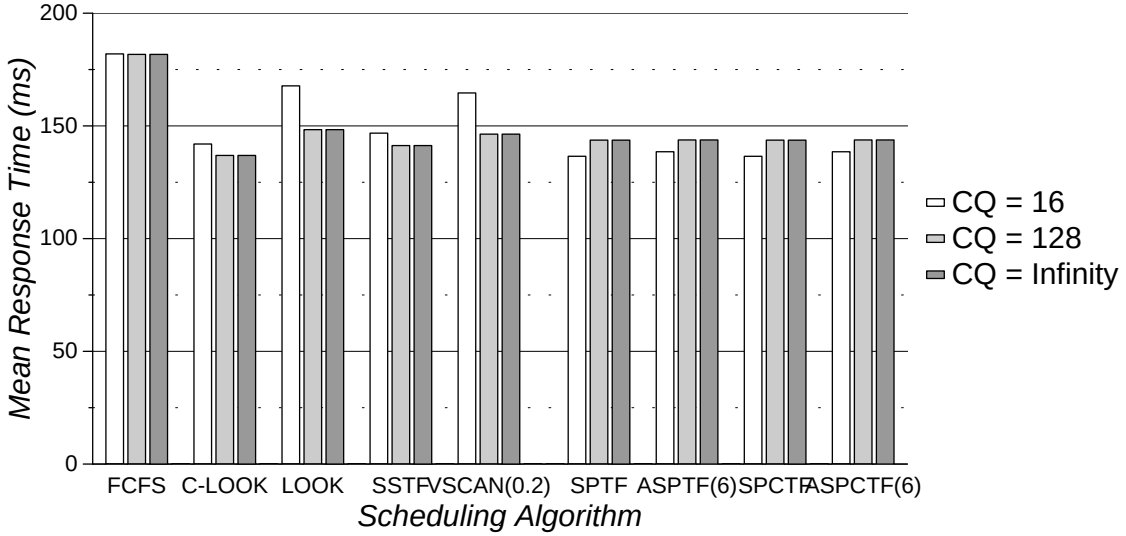
For disks with maximum command queue lengths of 128, sequential scheduling of reads and/or writes affects the performance metrics for the *SynRGen* workload by less than 1%. For the *Compress* workload, some of the algorithms experience a 1–2% drop in mean response times when scheduling sequential write requests in logically ascending order for disks with Full command queueing. Therefore, the remaining experiments in this section use no sequential scheduling optimizations for configurations with Preseek command queueing and sequential scheduling of both reads and writes for configurations with Full command queueing.[3]

Maximum Command Queue Length

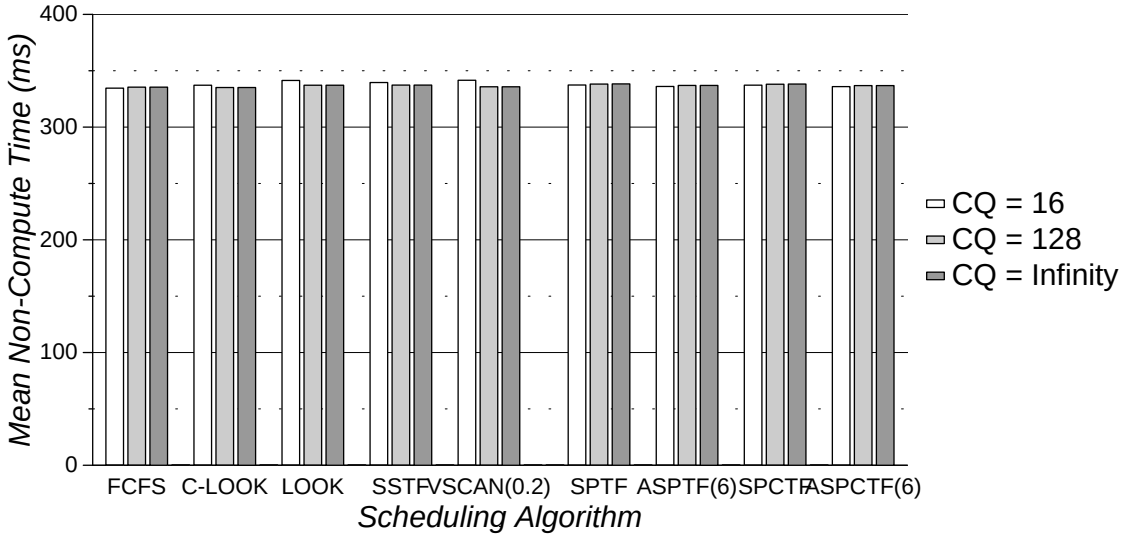
For 1-user and 2-user *SynRGen* workloads, the choice of maximum command queue length affects mean non-compute times by less than 1%. Figures 6.13 and 6.14 show the mean response times and mean non-compute times for 4-user configurations with Preseek and Full command queueing, respectively. Maximum command queue lengths of 128 and infinity provide equivalent performance across all algorithms. Reducing the maximum queue length to 16 degrades performance for the LBN-based algorithms, increasing mean response times by up to 13% and mean non-compute times by up to 2%. A maximum queue length of 16 provides marginally superior performance for the full-knowledge algorithms, with mean non-compute times decreasing less than 1%. SPTF-based algorithms are greedy in nature and therefore highly susceptible to request starvation. Limiting the number of pending requests visible to the scheduler results in increased age-sensitivity, thereby reducing the potential for starvation of high priority requests.

Figures 6.15 and 6.16 present the performance data for the 8-user *SynRGen* configurations. Again, a maximum command queue length of 128 provides good performance across

[3]Sequential scheduling of reads is included for Full command queueing configurations only for consistency with other experiments. Little or no benefit is gained from this optimization in this context.

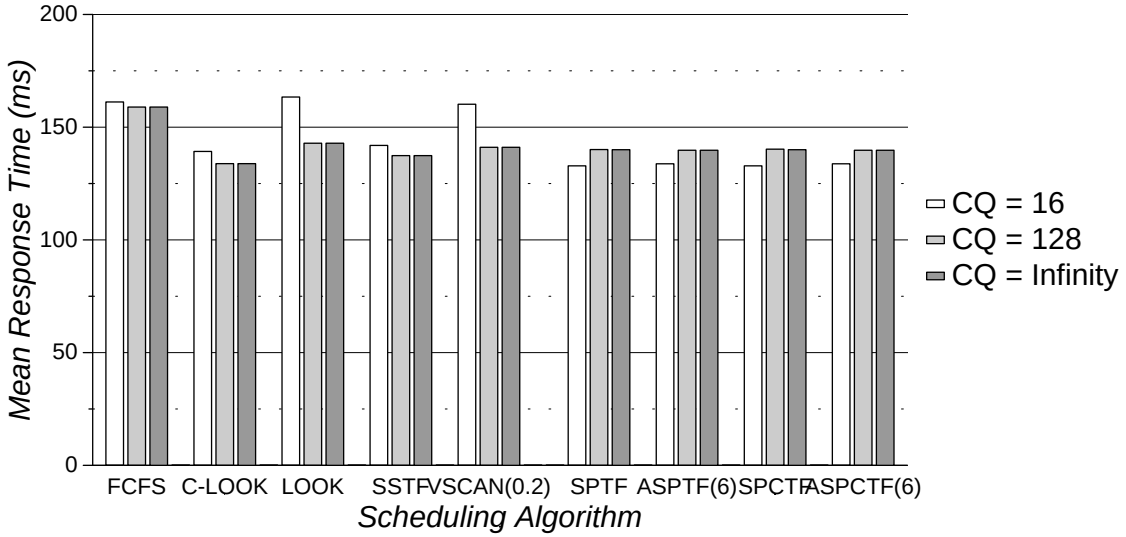


(a) Mean Response Time

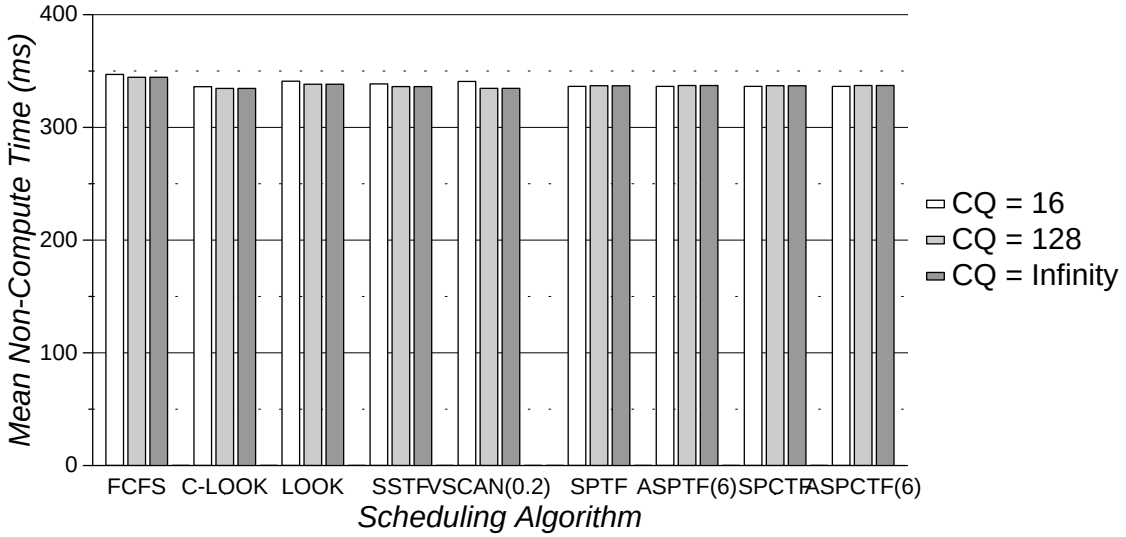


(b) Mean Non-Compute Time

Figure 6.13: *SynRGen*, 4 Users: Scheduling Algorithm Performance for Disks with Preseek Command Queueing

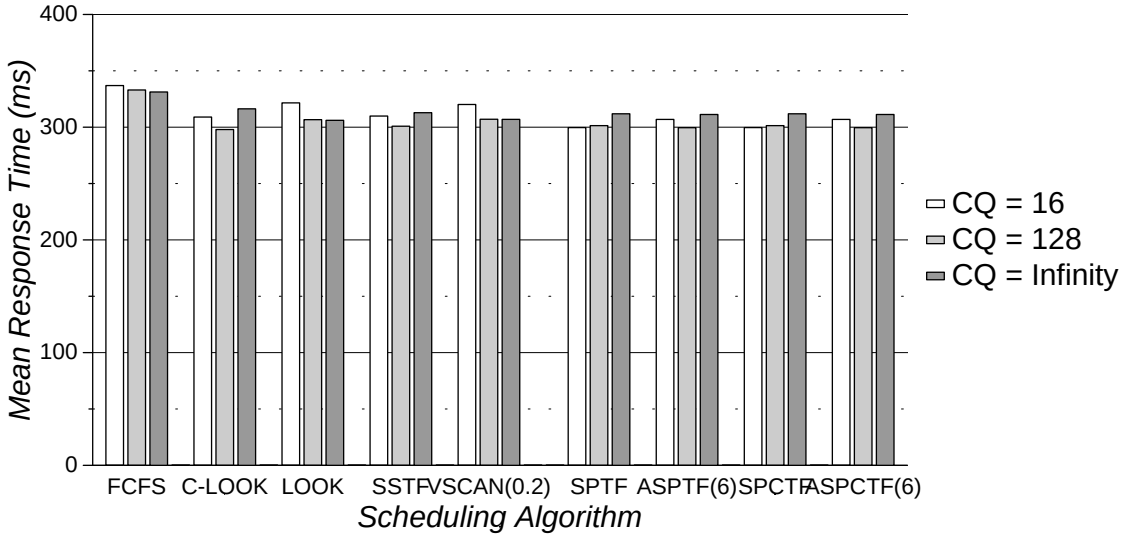


(a) Mean Response Time

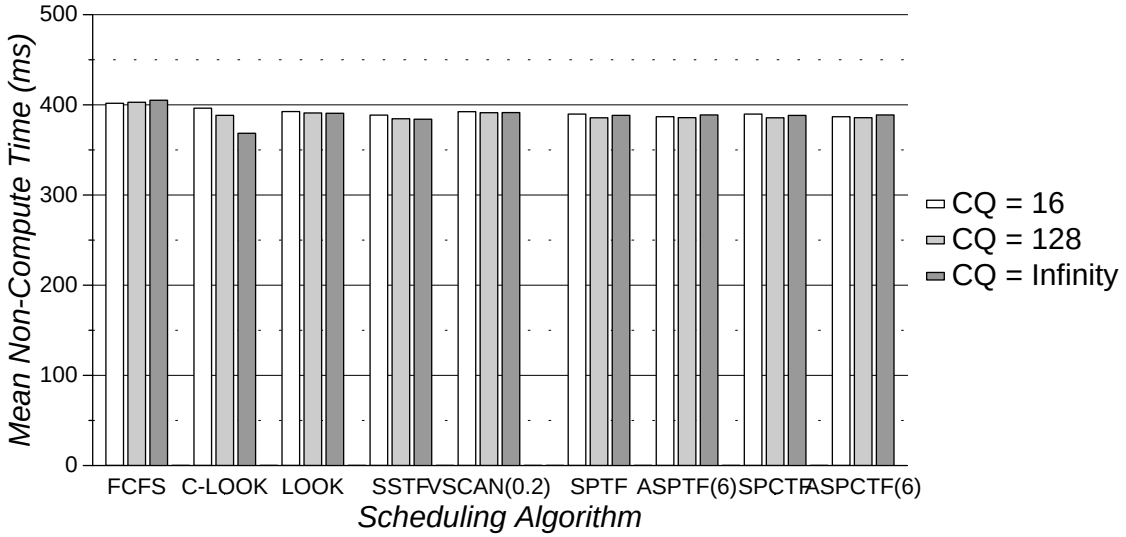


(b) Mean Non-Compute Time

Figure 6.14: *SynRGen*, 4 Users: Scheduling Algorithm Performance for Disks with Full Command Queueing

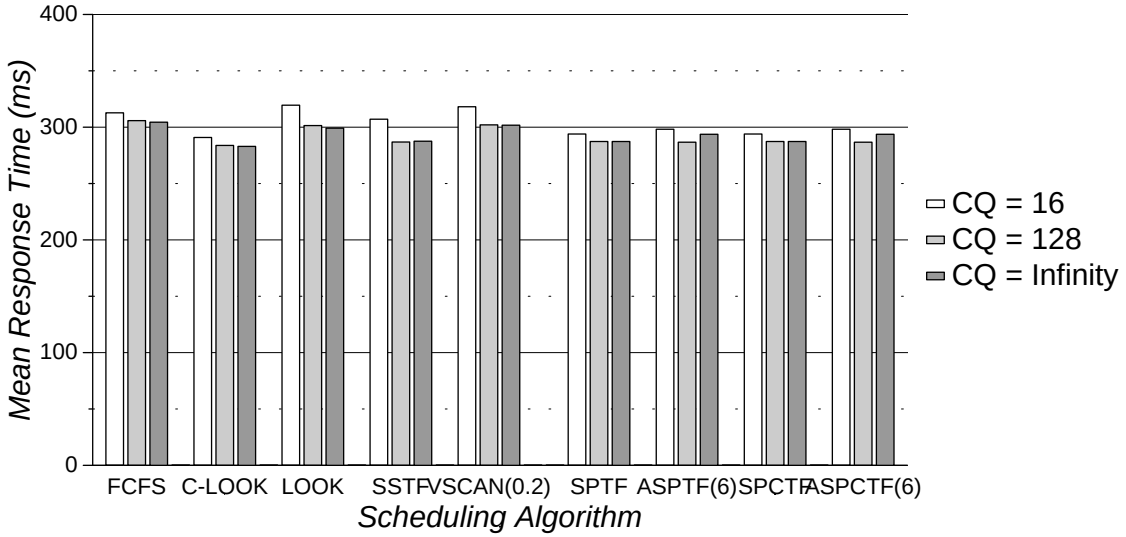


(a) Mean Response Time

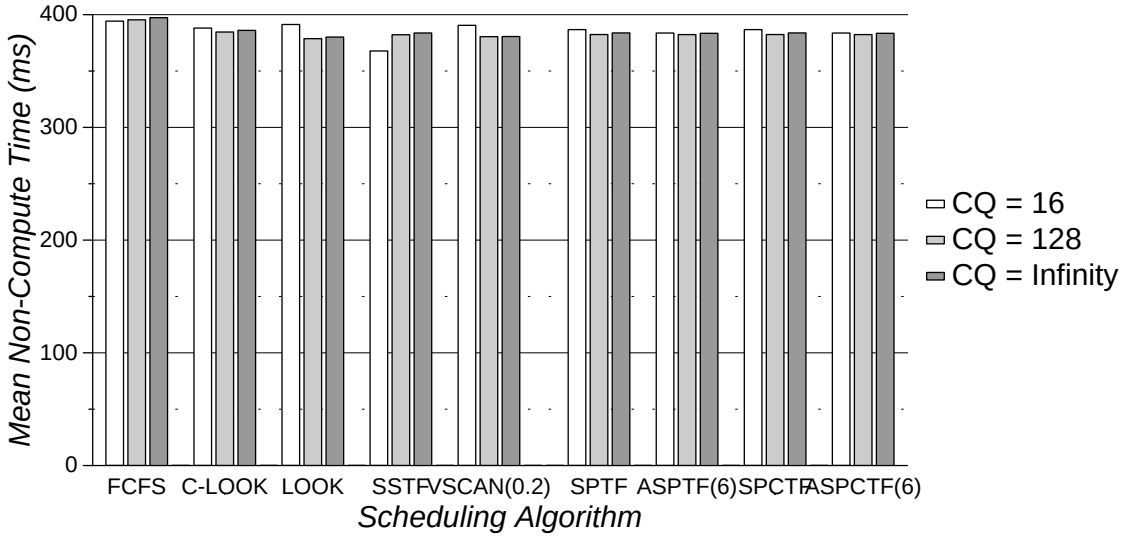


(b) Mean Non-Compute Time

Figure 6.15: *SynRGen*, 8 Users: Scheduling Algorithm Performance for Disks with Preseek Command Queueing



(a) Mean Response Time



(b) Mean Non-Compute Time

Figure 6.16: *SynRGen*, 8 Users: Scheduling Algorithm Performance for Disks with Full Command Queueing

the range of scheduling algorithms. In many cases, performance degrades when the maximum command queue length changes to infinity. That is, it is often better if the scheduler considers only the "most recent" 128 requests. There are two anomalous points in the graphs that bear further discussion. For C-LOOK scheduling of disks with Preseek command queueing, C-LOOK appears to achieve much better performance with an infinite command queue length. For SSTF scheduling of disks with Full command queueing, SSTF appears to achieve superior performance with a maximum command queue length of 16. In both cases, the number of "tasks" completed before the process-flow model halts is actually lower. That is, the processes simulated in the host model experienced an unusual ordering such that the model had to "abort" earlier for these simulation runs.[4] Therefore, these two configurations represent inferior design points, contrary to the displayed performance metric values.

Figures 6.17 and 6.18 show the mean response times and mean non-compute times for *Compress* configurations with Preseek and Full command queueing, respectively. For the LBN-based algorithms, maximum command queue lengths of 128 and infinity produce slightly lower (less than 1%) mean non-compute times. For the SPTF-based algorithms, the converse is true; configurations with a maximum queue length of 16 have a marginal performance advantage over those with longer queue lengths. This matches the behavior of the 4-user *SynRGen* workload, discussed above.

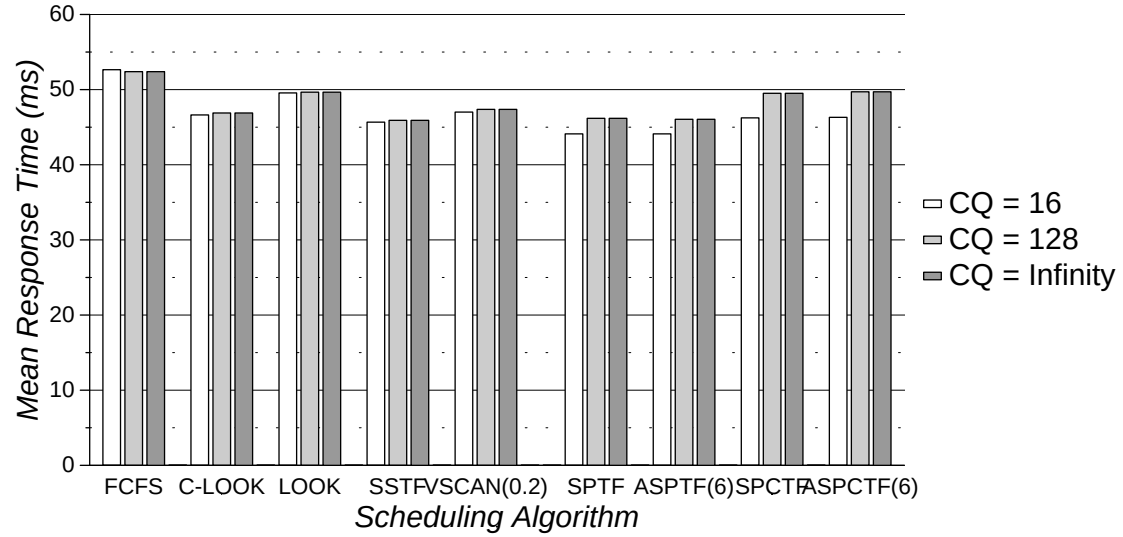
Comparison of Disk-Based Scheduling Algorithms

For 1-user and 2-user *SynRGen* workloads, all algorithm but FCFS provide equivalent performance. For the 2-user traces, scheduling with FCFS degrades mean non-compute times by up to 2%. For the 4-user traces, C-LOOK shows a slight (1% or less) performance advantage over the other LBN-based algorithms and the SPTF-based algorithms. For the 8-user traces, SSTF provides the lowest mean non-compute time among the LBN-based algorithms, even though C-LOOK has the lowest mean response time. For *SynRGen*, C-LOOK apparently improves the performance of low priority requests at the expense of high priority requests. For the *Compress* workload, all algorithms but FCFS provide roughly equivalent performance.

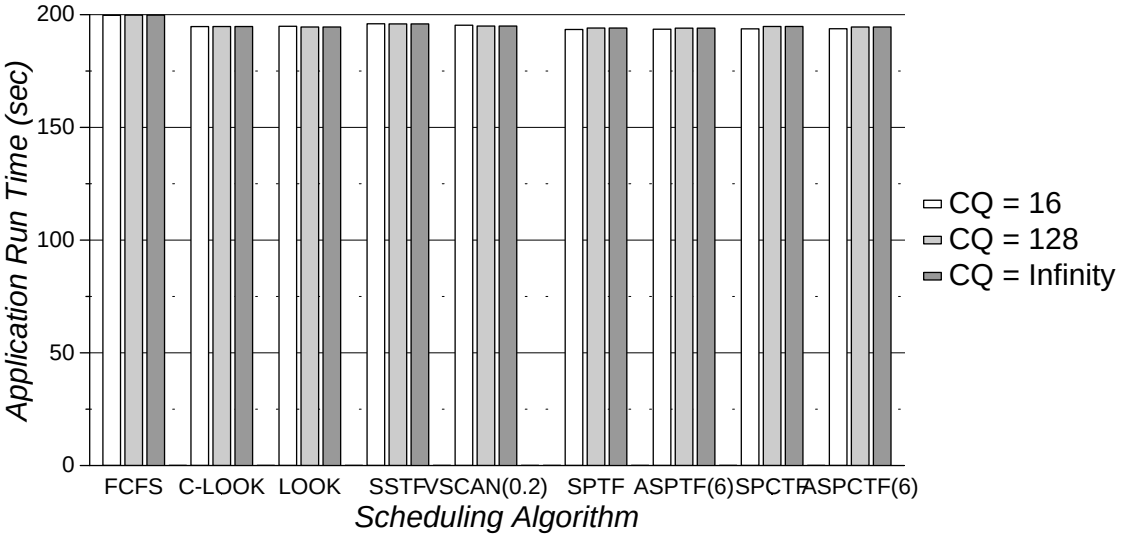
Comparison with Scheduling at the Host

Table 6.2 presents application-specific performance data for "reasonable" host-based and disk-based centralized scheduling of *SynRGen* and *Compress*. For light workloads (i.e., the 1-user and 2-user *SynRGen* traces), disk-based scheduling has a slight advantage. As the workload intensity increases, using host-based scheduling improves the application-specific performance metrics by approximately 2%. Comparing tables 6.2a and 6.2b, the advantage of host-based scheduling lessens as the complexity of the disk controller increases (from Preseek to Full command queueing).

[4]The process-flow model stops executing as soon as it reaches a state where there are no more events in the trace for a process that is "active" at the end of the trace.

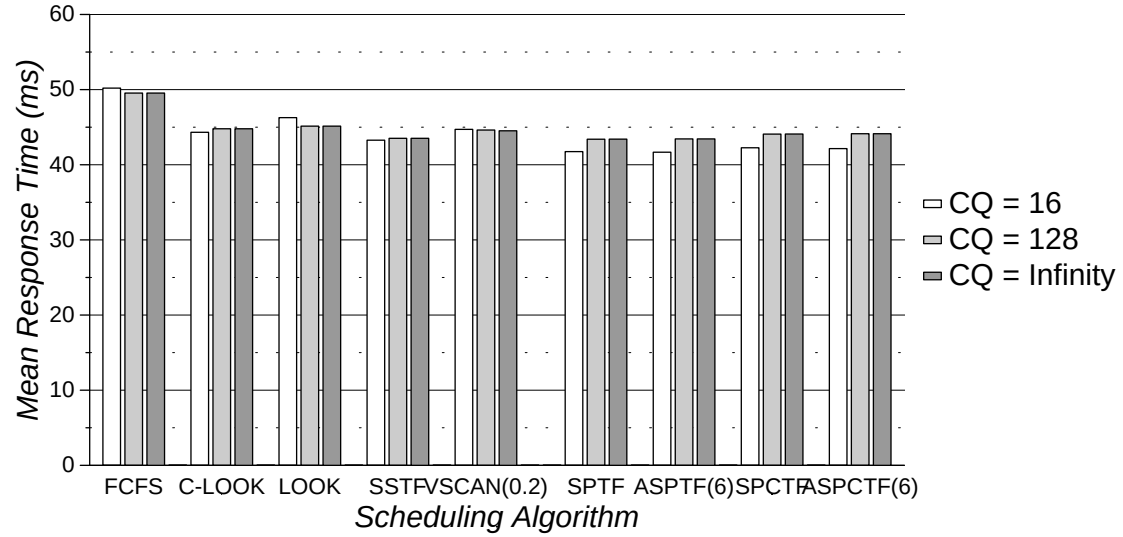


(a) Mean Response Time

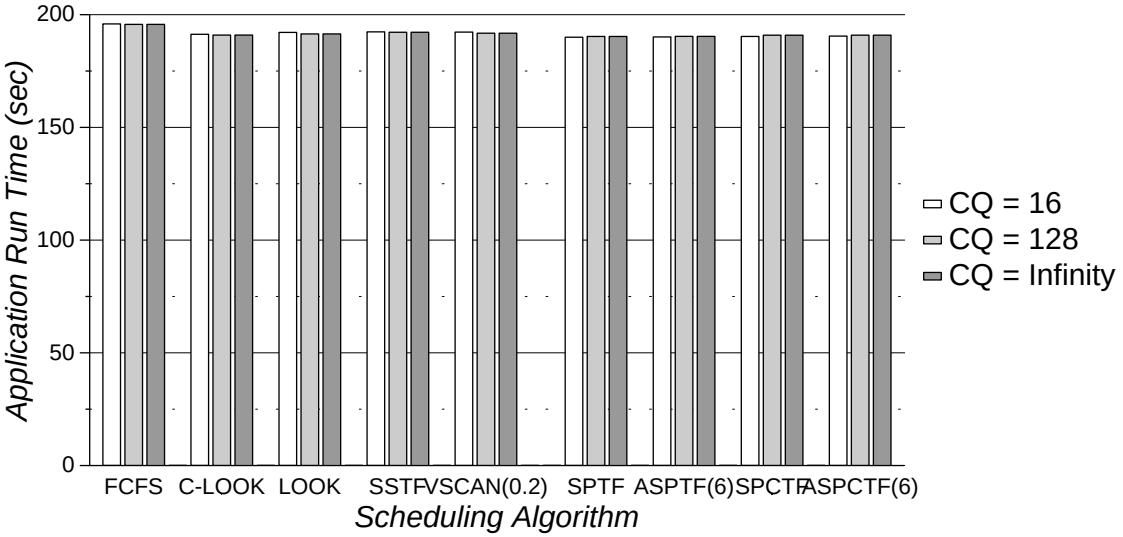


(b) Mean Non-Compute Time

Figure 6.17: *Compress*: Scheduling Algorithm Performance for Disks with Preseek Command Queueing



(a) Mean Response Time



(b) Mean Non-Compute Time

Figure 6.18: *Compress*: Scheduling Algorithm Performance for Disks with Full Command Queueing

Trace	Host-Based (all .CW.SSR) C-LOOK CQ = 2	Host-Based (all .CW.SSR) SPCTF CQ = 1	Disk-Based C-LOOK CQ = 128	Disk-Based SPCTF CQ = 128
SynRGen(1)	225.09	225.19	**225.02**	**224.60**
SynRGen(2)	433.31	436.11	**433.23**	**430.98**
SynRGen(4)	**334.08**	339.27	335.12	**338.05**
SynRGen(8)	**380.90**	**377.74**	388.34	385.65
Compress	**191.82**	**191.86**	194.74	194.76

(a) Preseek Command Queueing

Trace	Host-Based (all .CW.SSR) C-LOOK CQ = 2	Host-Based (all .CW.SSR) SPCTF CQ = 1	Disk-Based (all .SSW.SSR) C-LOOK CQ = 128	Disk-Based (all .SSW.SSR) SPCTF CQ = 128
SynRGen(1)	225.00	225.19	**224.95**	**224.56**
SynRGen(2)	**432.42**	436.11	432.85	**430.94**
SynRGen(4)	336.33	339.27	**334.57**	**336.97**
SynRGen(8)	**377.01**	**377.74**	384.60	382.38
Compress	**190.05**	191.86	190.95	**190.90**

(b) Full Command Queueing

Table 6.2: Mean Non-Compute Times (ms) or Application Run Times (sec) for "Reasonable" Centralized Schedulers

6.1.3 Summary of Conclusions

For disk-based centralized scheduling using only hardware-specific knowledge, C-LOOK generally provides the lowest mean response times among the seek-reducing algorithms. Full-knowledge scheduling algorithms, such as SPCTF, provide the highest levels of performance. On-board caches and prefetching algorithms should be examined before using SPCTF or ASPCTF(6), as cache-sensitivity degrades SPTF-based algorithm performance for some on-board cache configurations. In particular, automatic prefetching on full read hits can cause excessive actuator motion as well as lower on-board cache hit rates.

Sequential scheduling of read requests improves the performance of disk-based centralized schedulers for some workloads (such as the *Snake* trace). Sequential scheduling of write requests provides a marginal performance improvement if write prebuffering is enabled. Otherwise, it degrades performance by increasing mean rotational latencies.

Disks with maximum command queue lengths of 128 often provide performance near that of disks with unbounded command queues. In some cases, a finite command queue length helps to reduce the starvation of high priority requests.

Disk-based centralized scheduling is a reasonable alternative to traditional host-based scheduling. For several of the DEC and HP traces and some of *SynRGen* traces with 4 users or less, disk-based schedulers outperform host-based schedulers of comparable complexity. The performance advantage of disk-based scheduling increases with the amount of inter-request concurrency available at each disk.

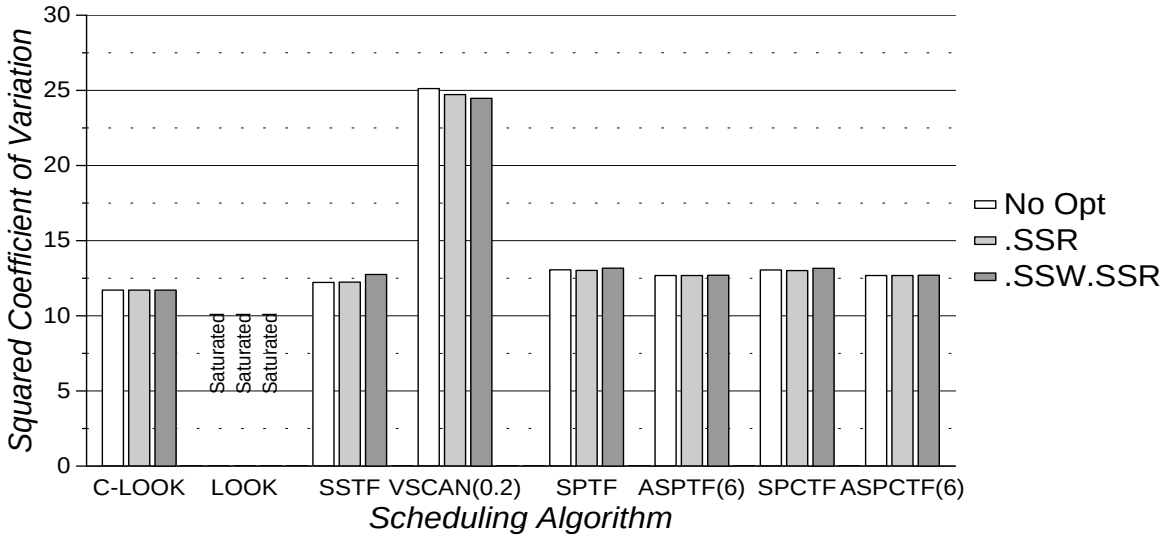
6.2 Scheduling with Information From "Above" and "Below"

This section examines the overall system performance effects of adding system-level knowledge to disk-based centralized scheduling. When scheduling the disk request traces, the lack of system-level information prevents the use of algorithms that optimize for system performance metrics. However, a scheduler can still attempt to prevent the starvation of high-priority requests by minimizing the request response time variance. For experiments using the full system traces, the host model passes a simple hi/lo priority flag to the disk subsystem model along with each request. This allows a disk-based scheduler to give priority to time-critical and time-limited requests.

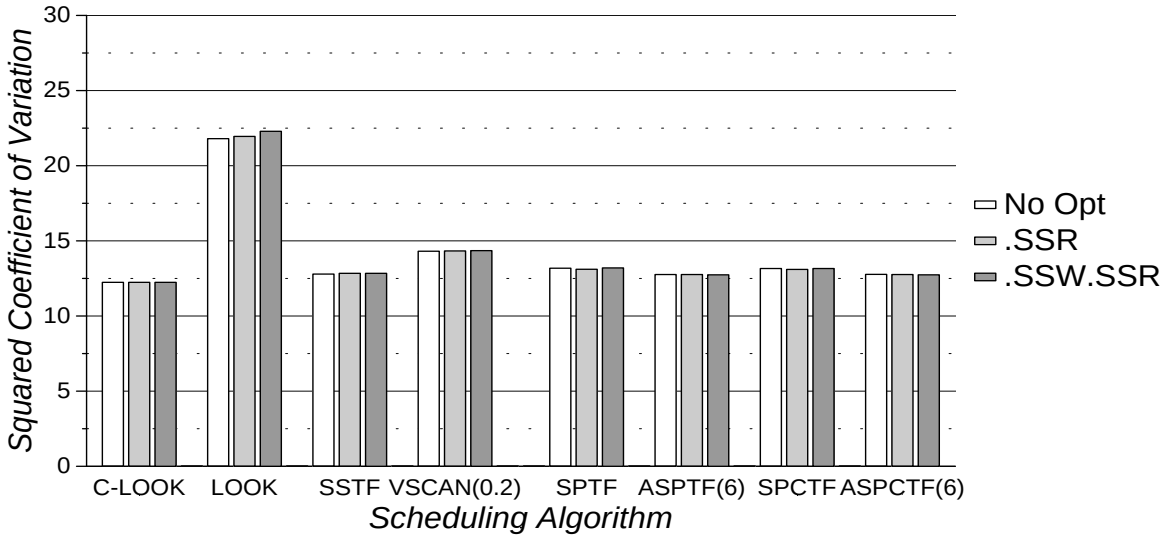
6.2.1 Disk Request Traces

Section 6.1.1 shows that the ASPTF(6) and ASPCTF(6) algorithms have mean response times equivalent (within a few percent) to SPTF and SPCTF, respectively. This section demonstrates that the age-sensitive SPTF algorithms provide much better starvation resistance, which leads to the conclusion that adding age-sensitivity to disk-based SPTF algorithms is generally advisable. Furthermore, age-sensitive SPTF algorithms usually provide better starvation resistance than LBN-based algorithms.

Figures 6.19–6.24 present the squared coefficients of variation corresponding to the mean response times given in figures 6.1–6.6. As expected, C-LOOK provides the best starvation

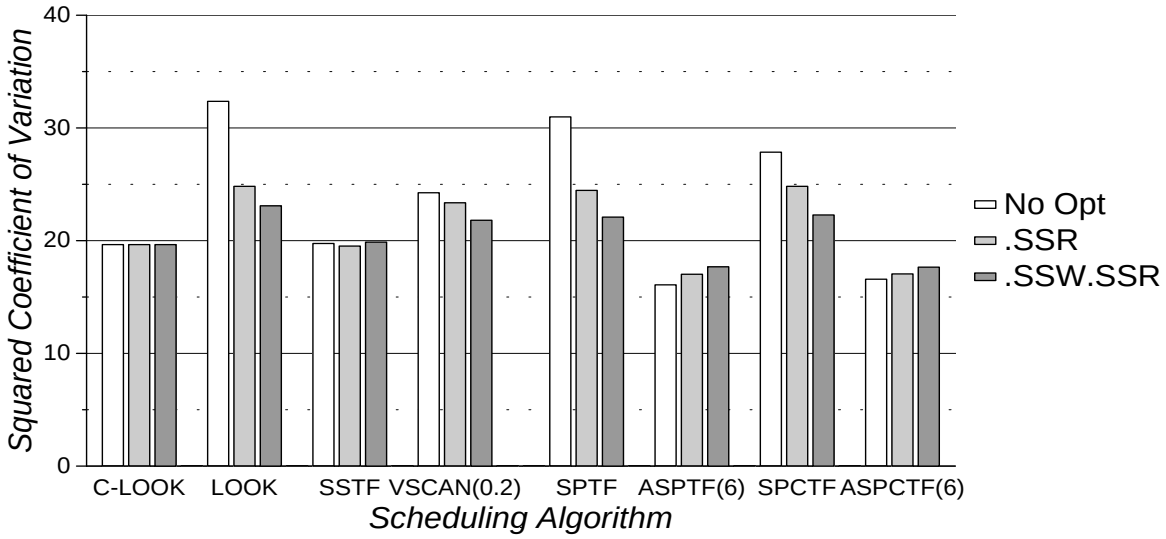


(a) Preseek Command Queueing

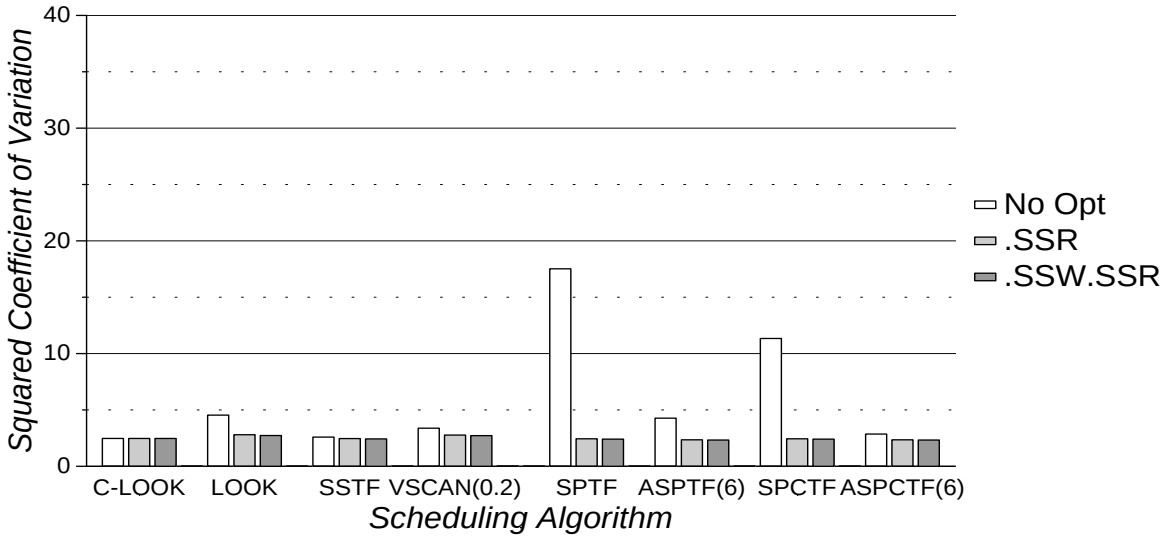


(b) Full Command Queueing

Figure 6.19: *Cello*, 1.75X: Sequential Scheduling Algorithm Performance

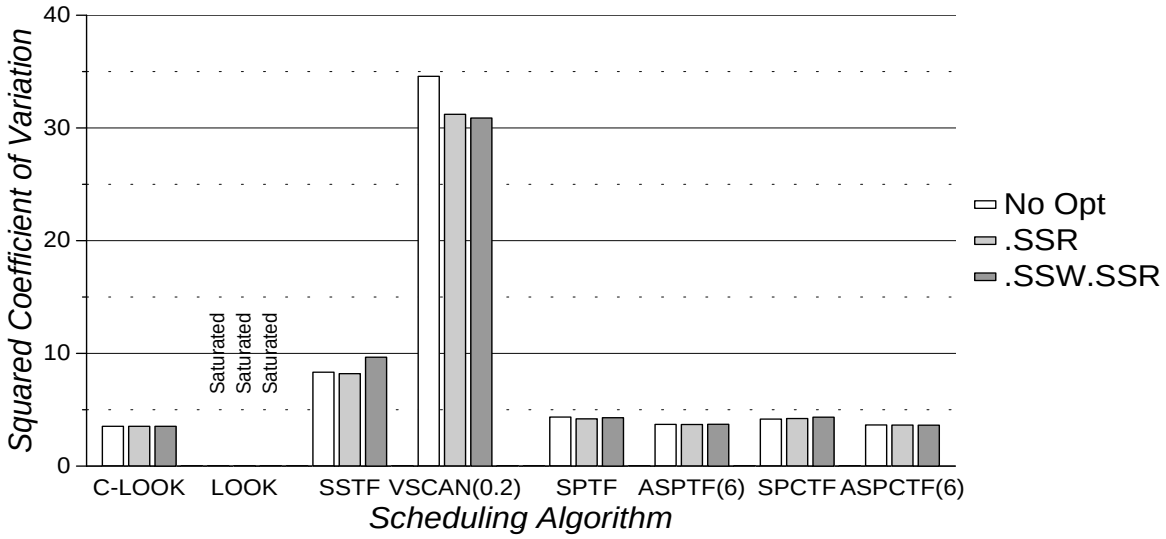


(a) Preseek Command Queueing

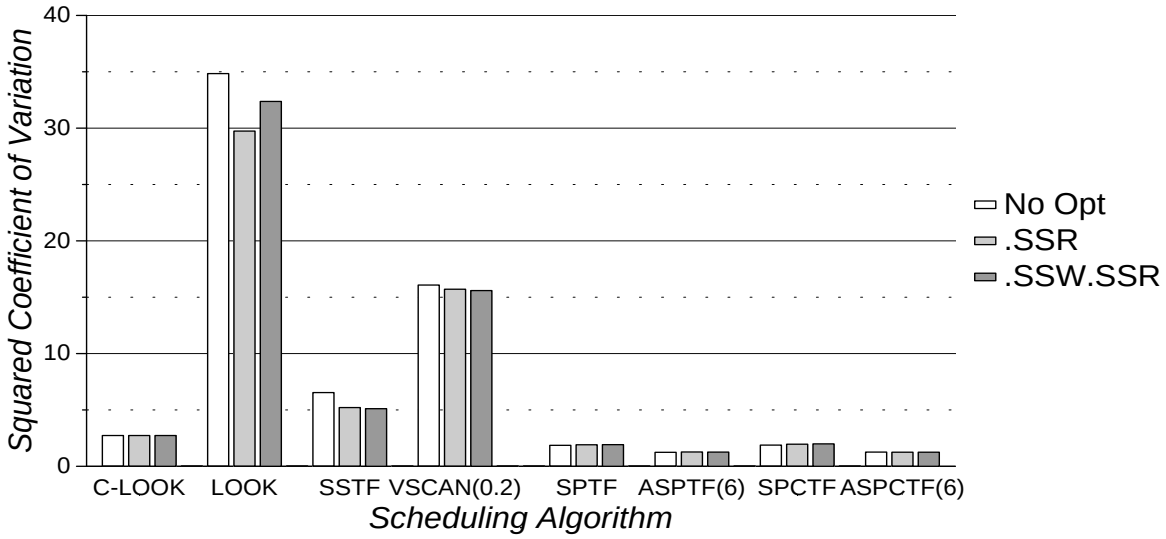


(b) Full Command Queueing

Figure 6.20: *Snake*, 1.25X: Sequential Scheduling Algorithm Performance

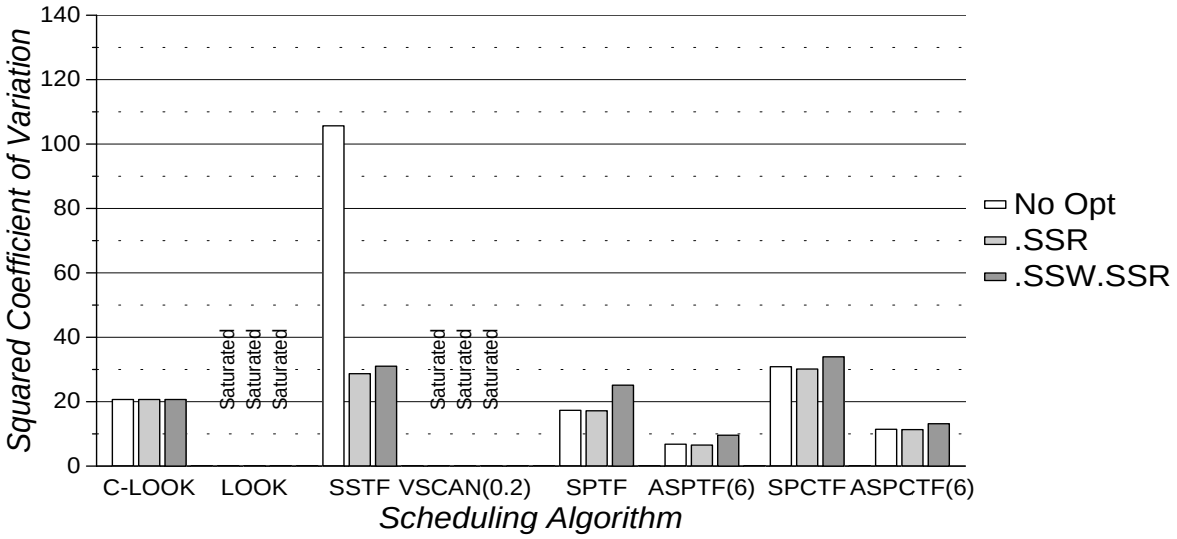


(a) Preseek Command Queueing

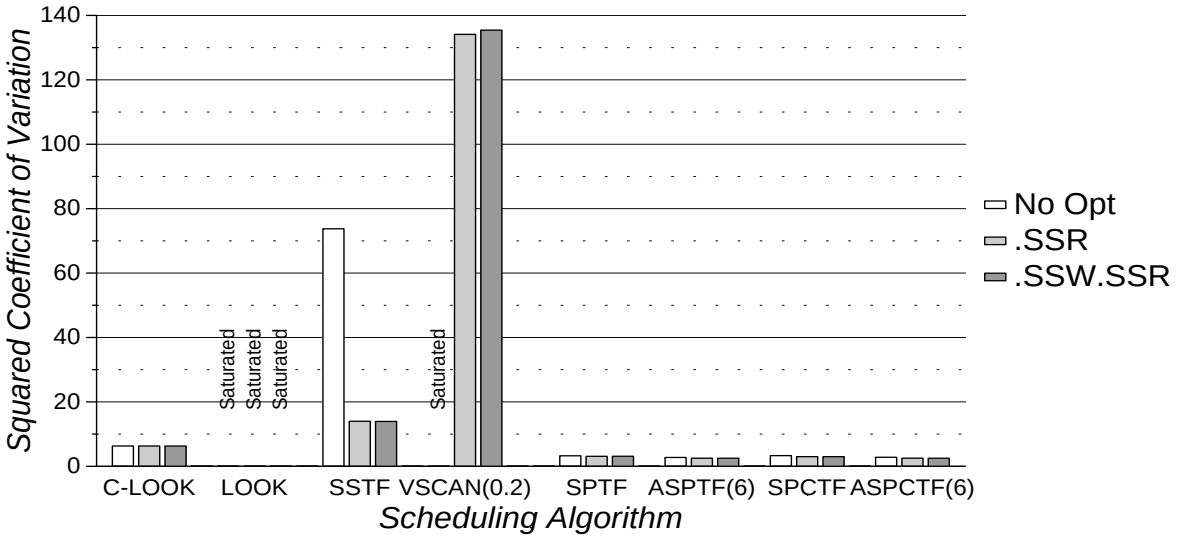


(b) Full Command Queueing

Figure 6.21: *Air-Rsv*, 2.5X: Sequential Scheduling Algorithm Performance

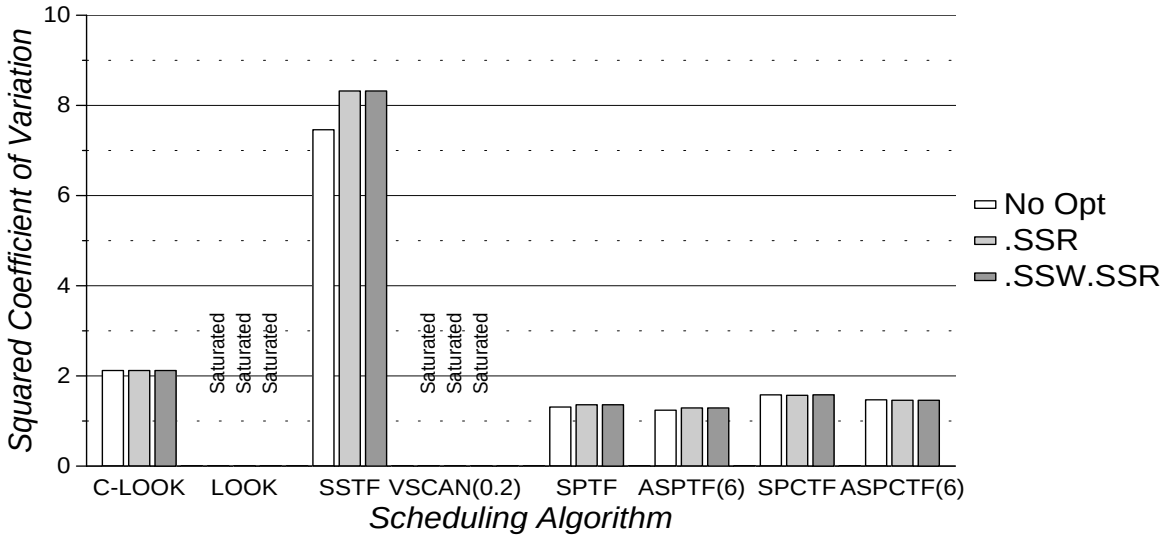


(a) Preseek Command Queueing

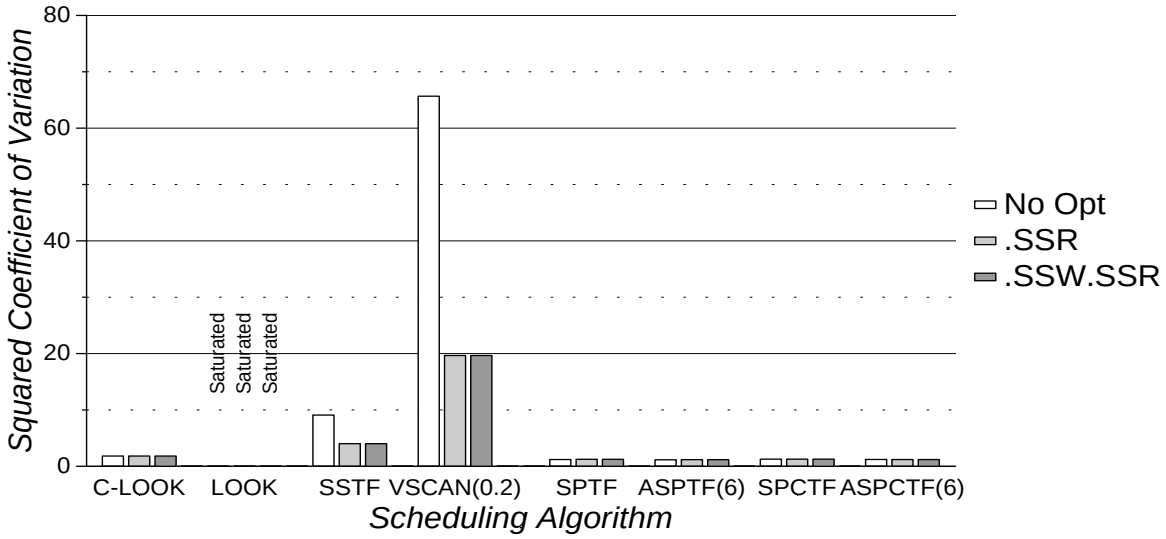


(b) Full Command Queueing

Figure 6.22: *Sci-TS*, 2.5X: Sequential Scheduling Algorithm Performance

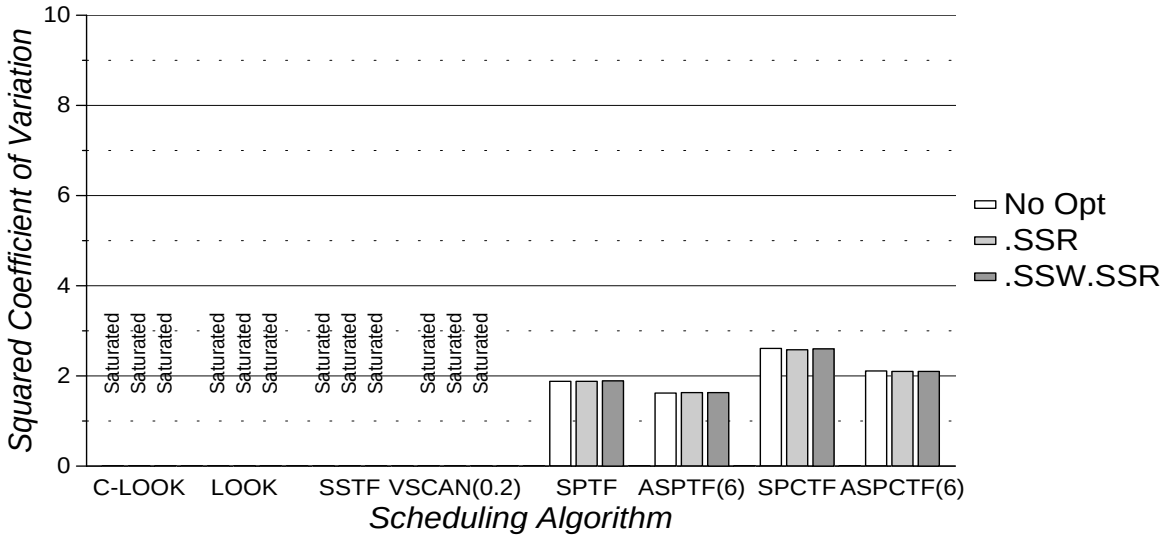


(a) Preseek Command Queueing

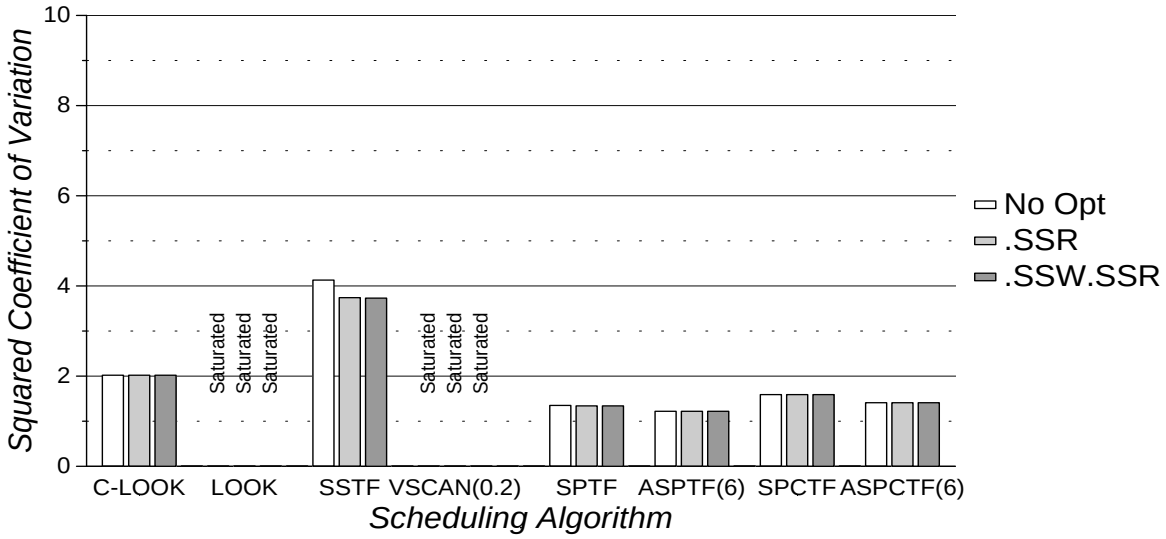


(b) Full Command Queueing

Figure 6.23: *Order*, 1.0X: Sequential Scheduling Algorithm Performance



(a) Preseek Command Queueing



(b) Full Command Queueing

Figure 6.24: *Report*, 1.0X: Sequential Scheduling Algorithm Performance

resistance for the LBN-based algorithms. The age-sensitive SPTF algorithms have lower squared coefficients of variation than C-LOOK for the majority of the disk-based scheduler implementations. For Full command queueing implementations of *Air-Rsv* and *Sci-TS*, the squared coefficients of variation for the age-sensitive SPTF algorithms are less than 50% of the corresponding values for C-LOOK. In all cases, ASPTF(6) and ASPCTF(6) provide better starvation resistance than SPTF and ASPCTF, respectively. For the *Snake* trace, age-sensitivity decreases the squared coefficient of variation by as much as 76%.

Comparison with Scheduling at the Host

Table 6.3 shows the mean response times and squared coefficients of variation for "reasonable" implementations of host-based and disk-based centralized schedulers using the ASPCTF(6) algorithm. The mean response times displayed are for configurations that do not automatically prefetch data on full read hits. The table does not contain ASPTF(6) performance data, since ASPCTF(6) provides superior mean response times for configurations with prefetching on full read hits disabled.

For the two HP traces, disk-based scheduling only results in lower mean response times for Full command queueing configurations. For the four DEC traces, disk-based scheduling is superior for both Preseek and Full command queueing. Table 6.3b shows that disk-based schedulers for Full command queueing configurations generally have the best (or roughly equivalent) starvation resistance. For Preseek configurations, four of the six workloads achieve lower response time variances with host-based scheduling.

6.2.2 Full System Traces

The experiments in this section assume that each request arriving at a disk has already been marked high or low priority (e.g., by an application or file system at the host). A disk-based scheduler using a 2Q algorithm then gives precedence to the high priority (time-critical and time-limited) requests. Common peripheral bus protocols (such as SCSI-2) do not necessarily provide for passing this type of control information. However, the capability could easily be incorporated into subsequent versions (or future bus protocols).

For 1-user *SynRGen* traces, adding 2Q scheduling reduces mean non-compute times by less than 1% for FCFS, C-LOOK, SSTF, SPTF, and SPCTF. For 2-user traces, the performance of FCFS improves up to 2.5% and the LBN-based algorithms improve 1–2% for maximum command queue lengths of 128 or greater. A maximum queue length of 16 prevents some high priority requests from quickly reaching the disk-based scheduler. Doubling the number of users approximately doubles the performance improvement of 2Q scheduling. Mean response times for FCFS drop up to 5% for configurations with maximum command queue lengths of 128 or greater. Mean response times for LBN-based and SPTF-based algorithms decrease by up to 3%. For *SynRGen* workloads of up to 4 users, disks with a maximum command queue length of 128 provide the same level of performance as disks with unbounded command queues.

Figures 6.25 and 6.26 present performance data for LBN-based 2Q scheduling of 8-user *SynRGen* traces using Preseek and Full command queueing, respectively. Figures 6.27 and 6.28 present corresponding data for SPTF-based 2Q scheduling. All of the 2Q algo-

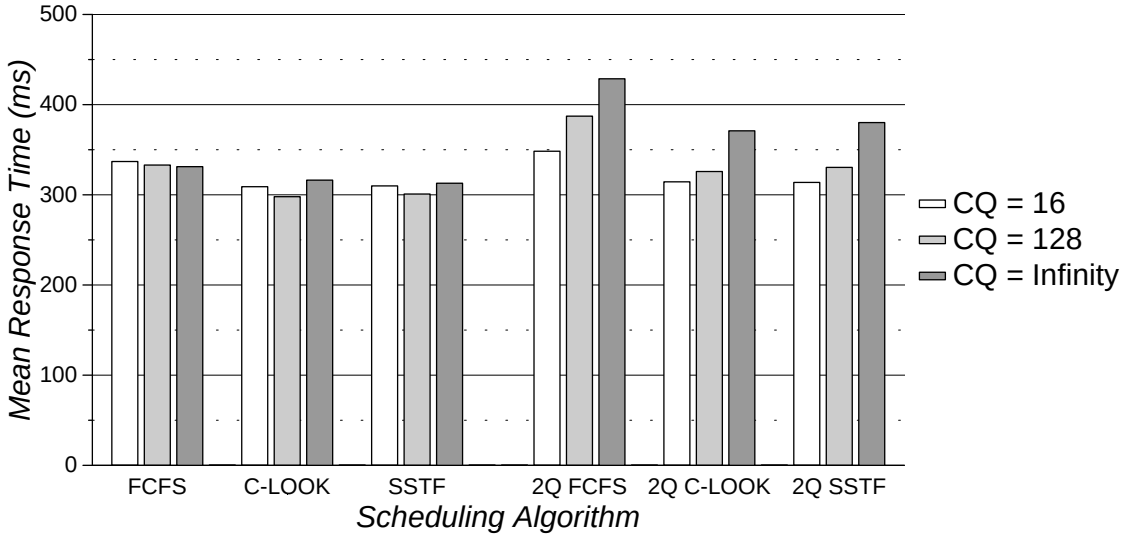
Trace	Scale Factor	Host-Based ASPCTF(6).CW.CR CQ = 1	Disk-Based ASPCTF(6).SSR, CQ = 128 Preseek	Full
Cello	1.75	115.38	118.43	108.66
Snake	1.25	33.92	34.98	25.04
Air-Rsv	2.5	30.18	23.50	21.13
Sci-TS	2.5	15.52	14.64	12.79
Order	1.0	20.94	18.09	17.75
Report	1.0	37.66	23.96	23.45

(a) Mean Response Times (ms)

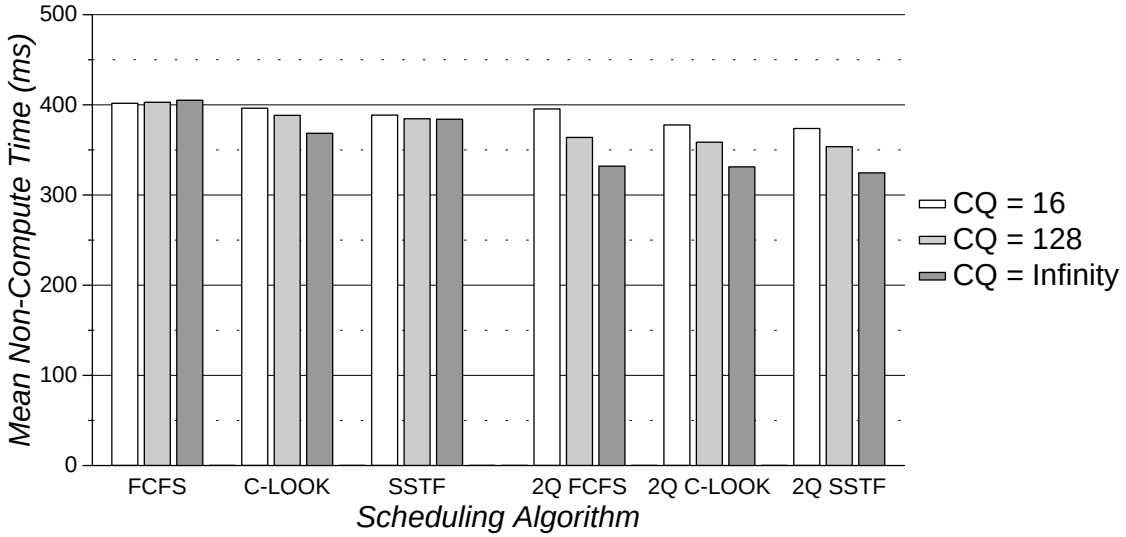
Trace	Scale Factor	Host-Based ASPCTF(6).CW.CR CQ = 1	Disk-Based ASPCTF(6).SSR, CQ = 128 Preseek	Full
Cello	1.75	10.38	12.64	12.71
Snake	1.25	14.12	16.82	2.30
Air-Rsv	2.5	3.64	3.69	1.22
Sci-TS	2.5	3.82	6.14	2.25
Order	1.0	2.06	1.12	1.18
Report	1.0	2.93	1.25	1.28

(b) Squared Coefficients of Variation

Table 6.3: Performance Metrics for "Reasonable" Age-Sensitive Centralized Schedulers. Prefetching disabled for full read hits.

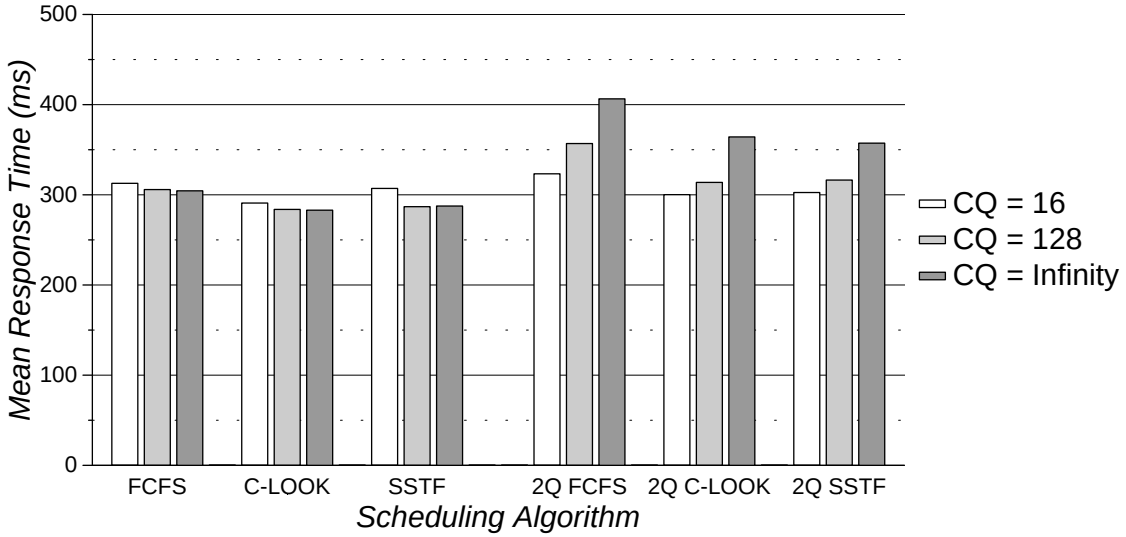


(a) Mean Response Time

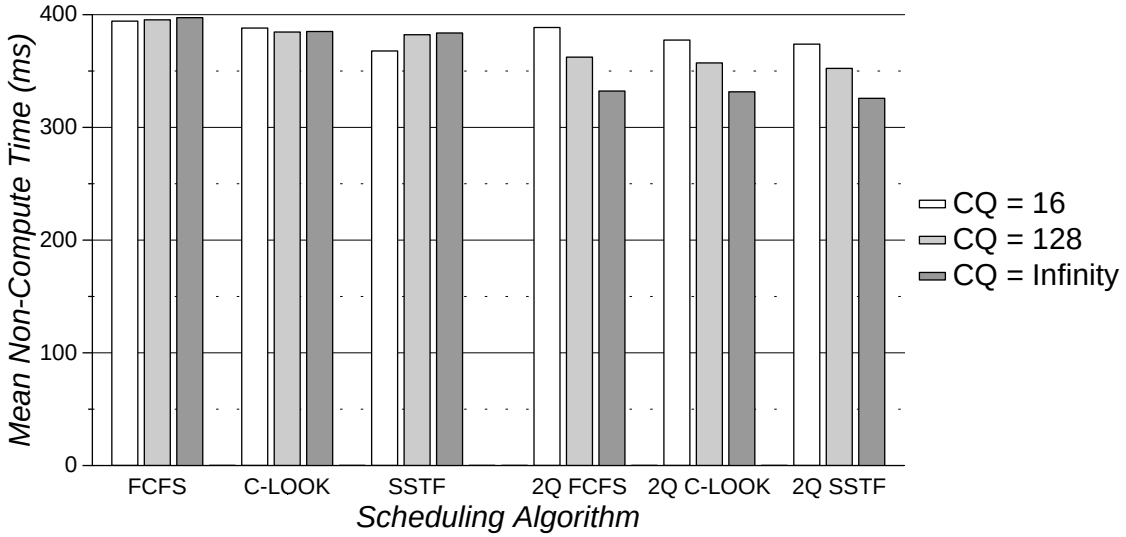


(b) Mean Non-Compute Time

Figure 6.25: *SynRGen*, 8 Users: 2Q LBN-Based Scheduling Algorithm Performance for Disks with Preseek Command Queueing

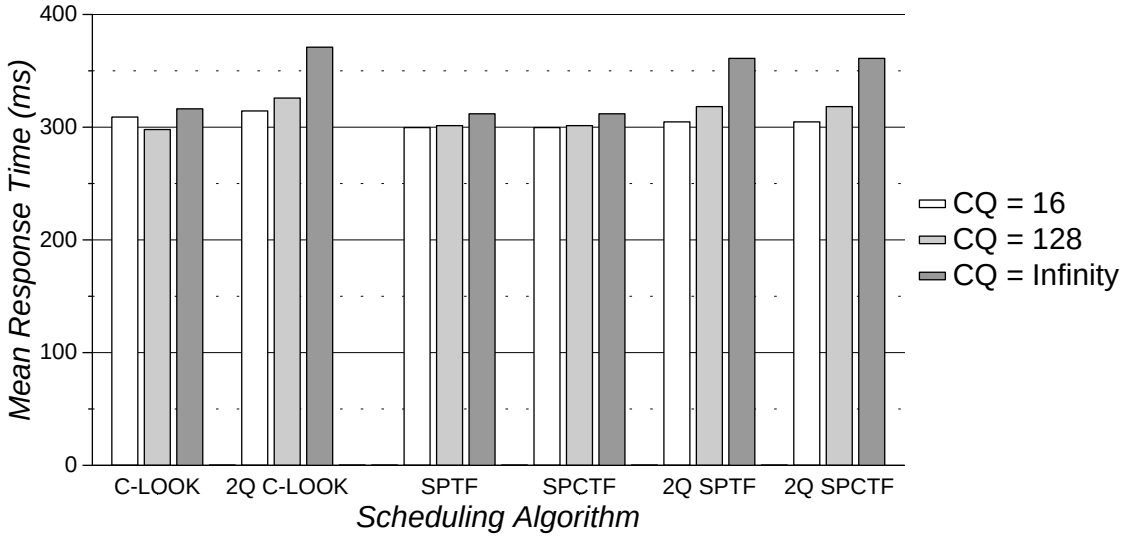


(a) Mean Response Time

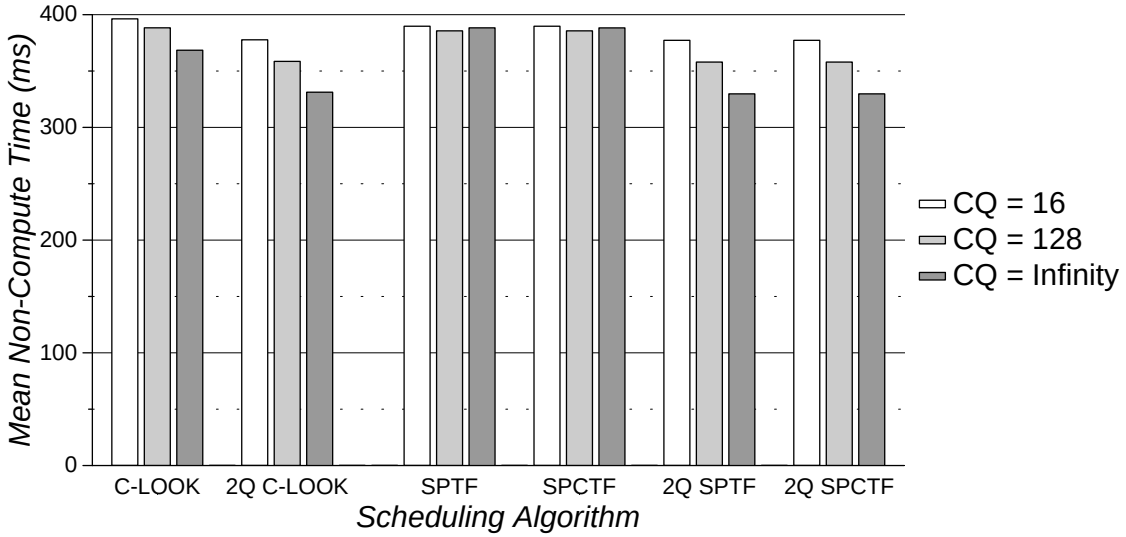


(b) Mean Non-Compute Time

Figure 6.26: *SynRGen*, 8 Users: 2Q LBN-Based Scheduling Algorithm Performance for Disks with Full Command Queueing

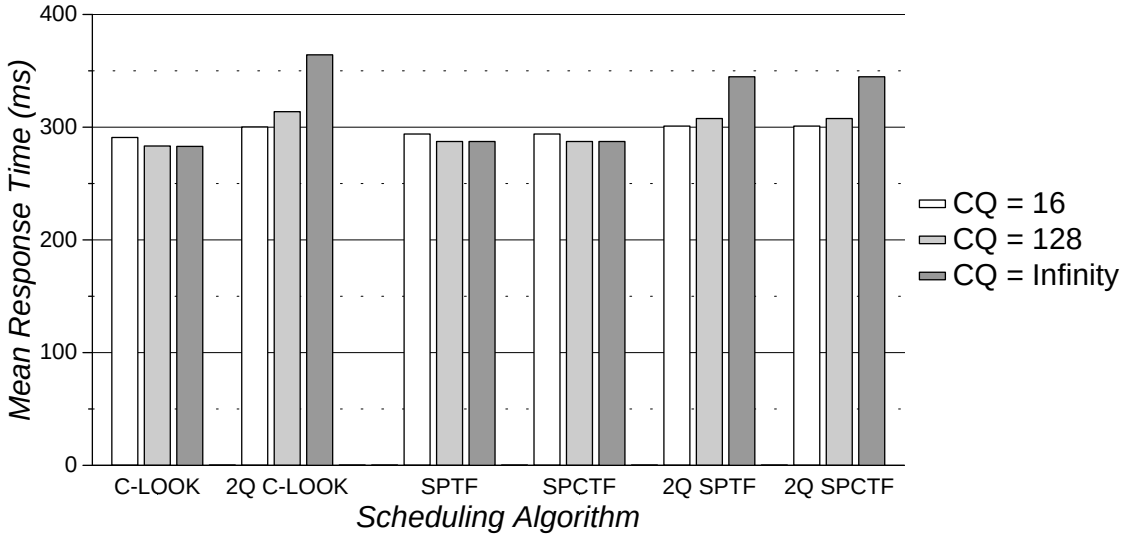


(a) Mean Response Time

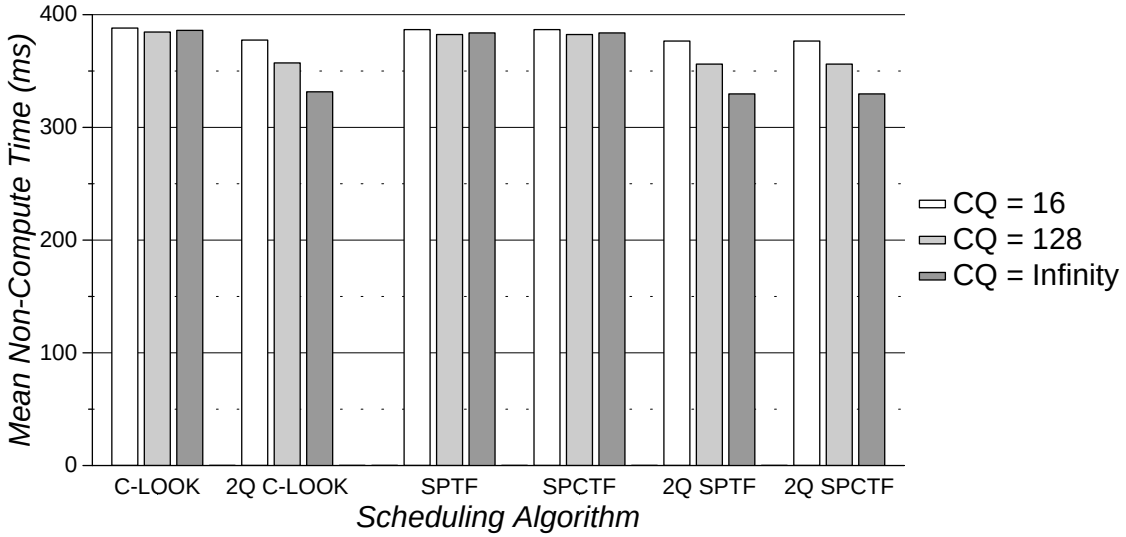


(b) Mean Non-Compute Time

Figure 6.27: *SynRGen*, 8 Users: 2Q Full-Knowledge Scheduling Algorithm Performance for Disks with Preseek Command Queueing



(a) Mean Response Time



(b) Mean Non-Compute Time

Figure 6.28: *SynRGen*, 8 Users: 2Q Full-Knowledge Scheduling Algorithm Performance for Disks with Full Command Queueing

rithms show similar performance improvement when the maximum command queue length is increased. Mean response times rise with the growing starvation of time-noncritical requests, and mean non-compute times decrease with the improvement in service times for time-critical and time-limited requests.

In contrast to the single-queue schedulers discussed in section 6.1.2, all of the disk-based 2Q schedulers perform best with unbounded command queues. Mean non-compute times drop 7–9% when the maximum command queue length changes from 128 to infinity. Finite-length command queues impair a disk-based scheduler's flexibility during bursts of heavy activity. Once a disk command queue reaches its maximum length, any subsequent requests must wait at the host (e.g., in a FCFS device driver queue) until the on-board queue length drops below the maximum. If high priority requests are blocked at the host, overall system performance may suffer. For some of the 8-user traces, a disk command queue length of 700 or more is necessary to prevent queueing of requests at the host.

For all configurations, 2Q scheduling with the FCFS, LBN-based, or full-knowledge algorithms improves performance over the single-queue implementations discussed in section 6.1.2.[5] For a maximum queue length of 128, 2Q scheduling decreases mean non-compute times by 7–10%.

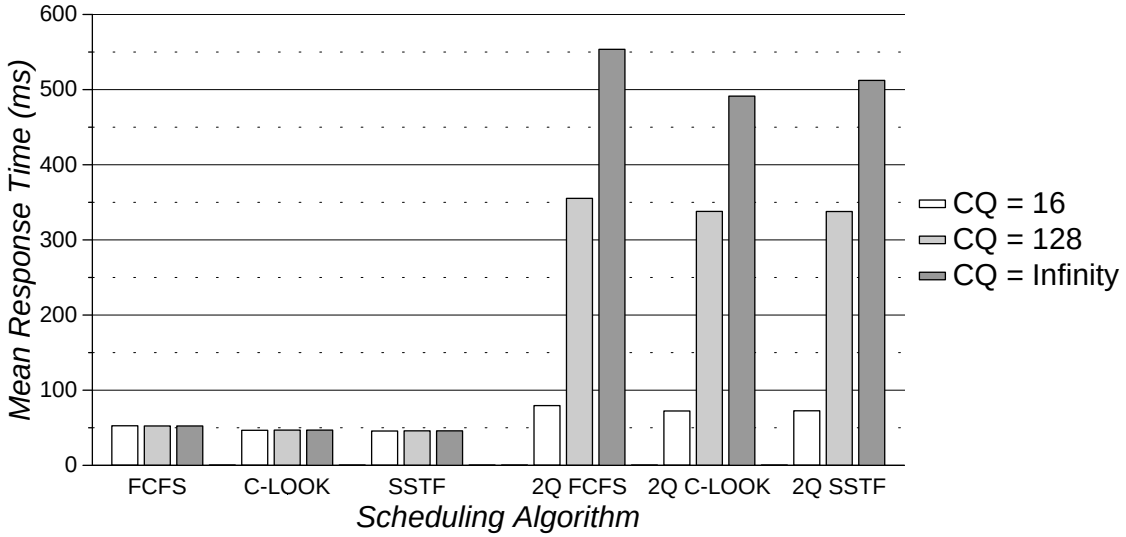
For 8-user *SynRGen* traces, SSTF provides the lowest mean non-compute times. C-LOOK or SPTF-based scheduling results in approximately 1.5% longer mean non-compute times, with FCFS trailing by an additional 1.5%.

The benefits of 2Q scheduling are less significant for the *Compress* workload. Figures 6.29–6.32 present performance data for LBN-based and full-knowledge 2Q scheduling with Preseek and Full command queueing configurations. Mean response times increase almost an order of magnitude as the maximum command queue length reaches infinity. Unlike *SynRGen*, the *Compress* traces gain little benefit from a maximum command queue length greater than 128. Even with FCFS scheduling, the maximum number of outstanding requests at any point in the *Compress* traces is only 163 (for the implementations studied). 2Q scheduling with a maximum command queue length of 128 provides a slight (1–2%) decrease in total run times for the compression workload. The run times for both SPTF-based algorithms are up to 3% faster than FCFS and the LBN-based algorithms. Although the advantage of SPTF-based algorithms is minimal when scheduling for *Compress*, they provide a much higher margin of performance improvement for heavier workloads.

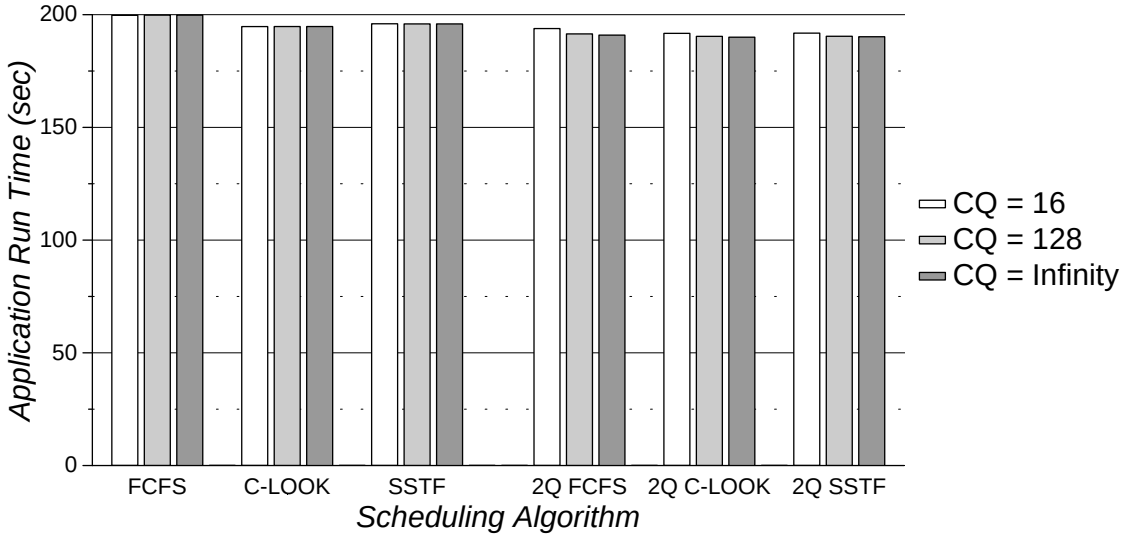
Comparison with Scheduling at the Host

Table 6.4 presents application-specific performance data for "reasonable" host-based and disk-based centralized 2Q scheduling of 8-user *SynRGen* and *Compress*. For *SynRGen*, the host-based schedulers provide the best performance because of the limited command queue length imposed on the disk-based schedulers. With unbounded command queues, the disk-based schedulers obtain a marginal performance advantage. The *Compress* traces are best served by a disk-based scheduler, since the pending request queues for *Compress* are usually short enough to completely fit into the on-board command queues.

[5] A few of the single-queue simulation runs produce anomalous results, in that a smaller number of "tasks" are completed before the process-flow model halts. See page 119 in section 6.1.2 for a description of this phenomenon.

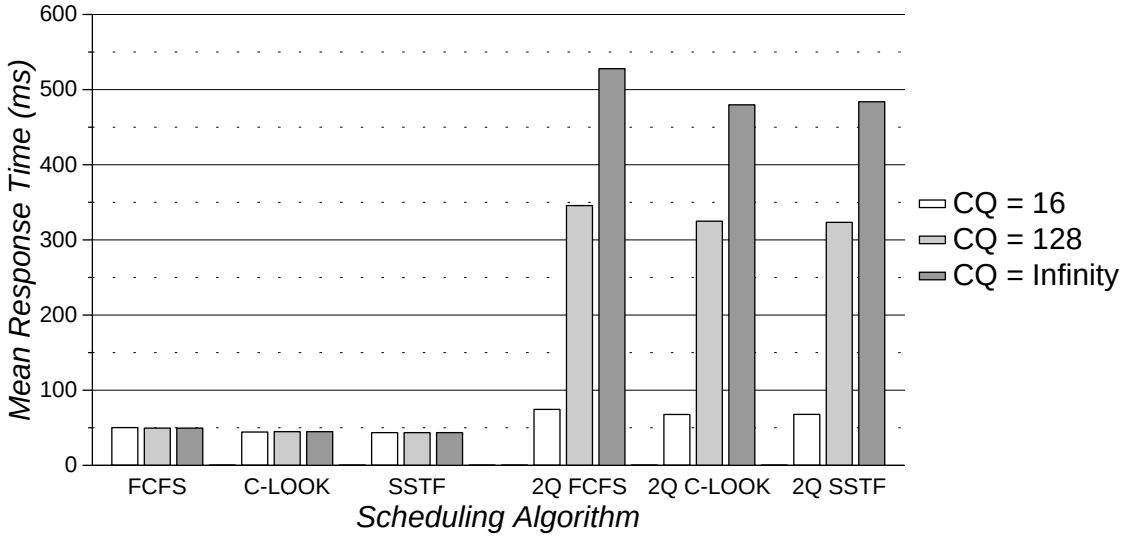


(a) Mean Response Time

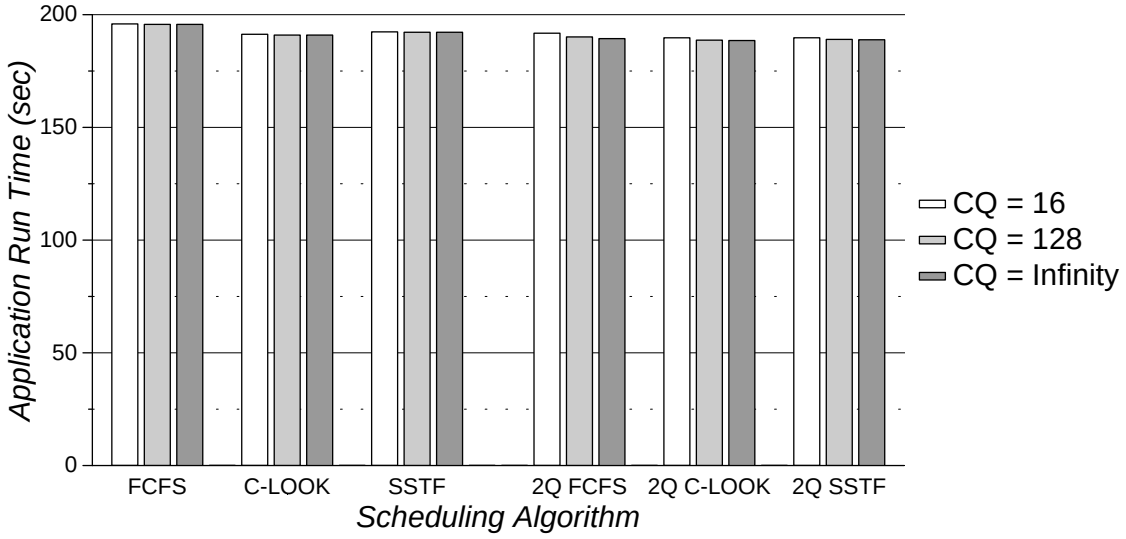


(b) Mean Non-Compute Time

Figure 6.29: *Compress*: 2Q LBN-Based Scheduling Algorithm Performance for Disks with Preseek Command Queueing

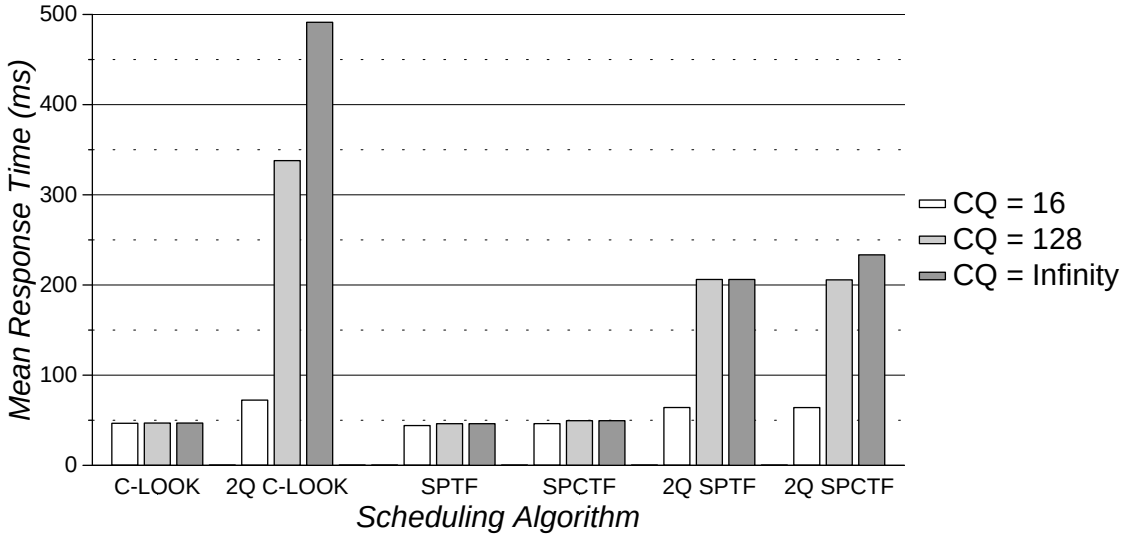


(a) Mean Response Time

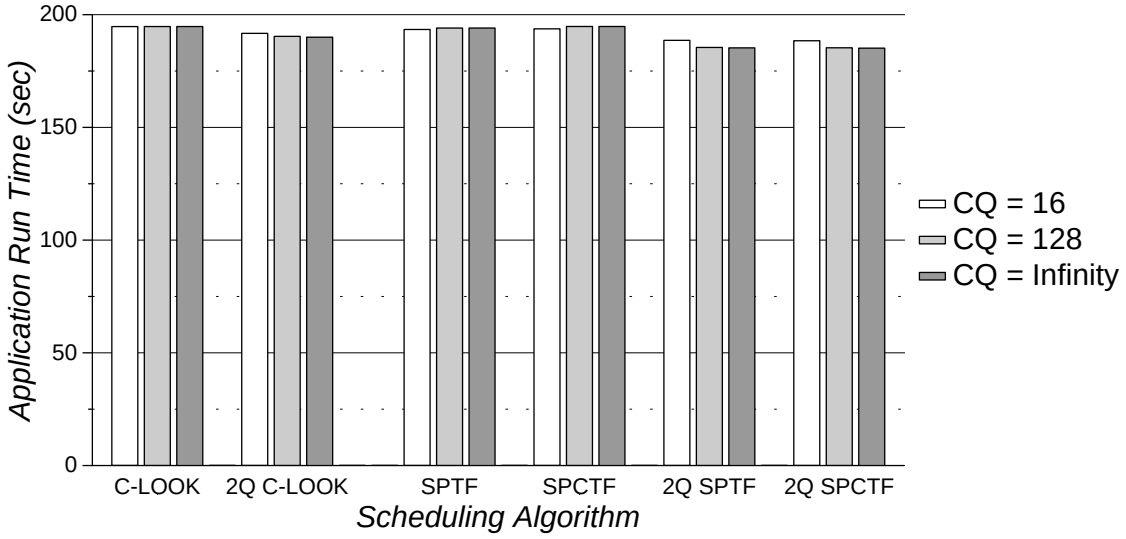


(b) Mean Non-Compute Time

Figure 6.30: *Compress*: 2Q LBN-Based Scheduling Algorithm Performance for Disks with Full Command Queueing

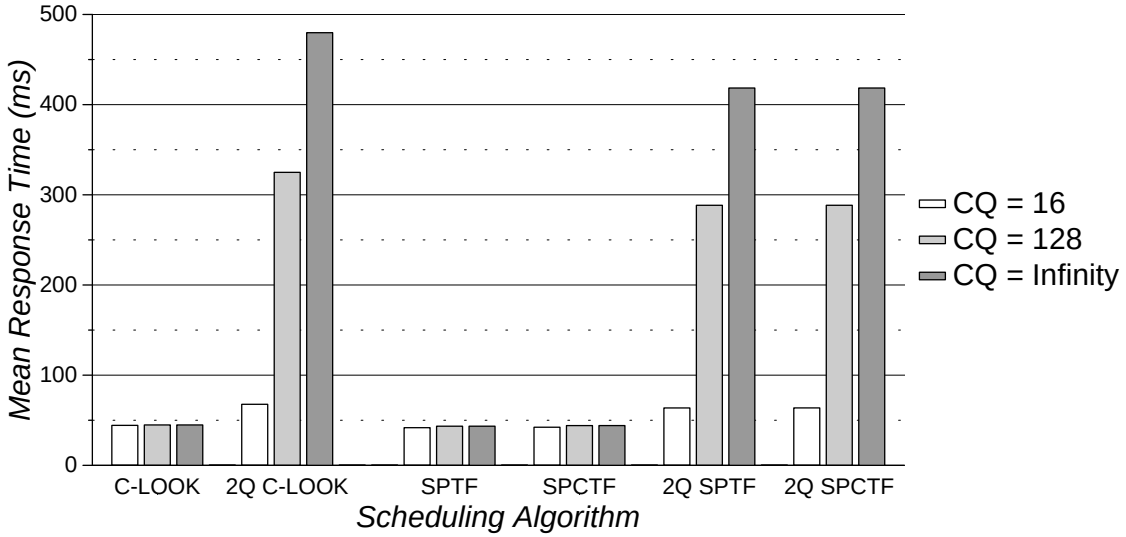


(a) Mean Response Time

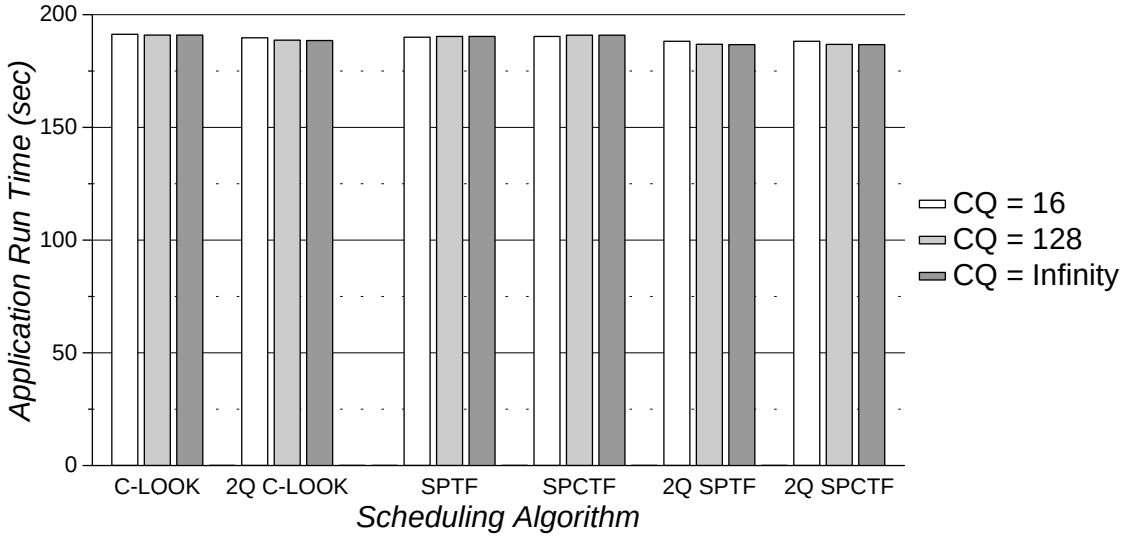


(b) Mean Non-Compute Time

Figure 6.31: *Compress*: 2Q Full-Knowledge Scheduling Algorithm Performance for Disks with Preseek Command Queueing



(a) Mean Response Time



(b) Mean Non-Compute Time

Figure 6.32: *Compress*: 2Q Full-Knowledge Scheduling Algorithm Performance for Disks with Full Command Queueing

Trace	Host-Based CQ = 1	Disk-Based, CQ = 128 Preseek	Full
8-User SynRGen	SSTF.CW.SSR **333.56**	SSTF 353.62	SSTF.SSR.SSW 352.34
	SPCTF.CW.SSR **332.05**	SPCTF 357.95	SPCTF.SSR.SSW 356.15
Compress	C-LOOK.CW.SSR 193.94	C-LOOK **190.38**	C-LOOK.SSR.SSW **188.70**
	SPCTF.CW.SSR 190.72	SPCTF **185.31**	SPCTF.SSR.SSW **186.82**

Table 6.4: Mean Non-Compute Times (ms) or Application Run Times (sec) for "Reasonable" Centralized 2Q Schedulers

6.2.3 Summary of Conclusions

Adding age-sensitivity to SPTF-based algorithms implemented in disk-based schedulers improves starvation resistance (i.e., response time variance). By correctly "weighting" aging information, the improvement in starvation resistance does not significantly degrade performance. In some cases, age-sensitivity actually improves mean response times by maintaining the ordering of sequential requests issued in logically ascending order. Disk-based SPTF schedulers with age-sensitivity usually outperform their host-based counterparts.

For experiments using the full system traces, 2Q scheduling improves overall system performance (as measured by application-specific performance metrics). This improvement results from giving priority to time-critical and time-limited requests. Time-noncritical requests are often starved, resulting in much higher mean request response times.

Finite command queue lengths impair the flexibility of disk-based 2Q schedulers. For heavy workloads, high priority requests may have to wait at the host because of full on-board command queues. Disk-based 2Q schedulers only outperform host-based 2Q schedulers when the maximum number of outstanding requests does not greatly exceed the maximum command queue length.

CHAPTER 7

Distributed Scheduling

Distributed scheduling implies two or more entities along the I/O path cooperatively reordering disk requests. This chapter examines distributed scheduler implementations where host-based schedulers (e.g., in O/S device drivers) cooperate with disk-based schedulers (in disk controllers). Most of the advantages of host-based and disk-based centralized scheduling are also found in distributed scheduling. Each component scheduler has easy and immediate access to local scheduling information. That is, the host-based scheduler has knowledge of system goals and guarantees, while each disk-based scheduler has accurate disk configuration and state information. The host-based component can perform scheduling optimizations such as request concatenation, while each disk-based component can maximize the concurrency between requests receiving service. Most of the disadvantages of host-based and disk-based centralized scheduling are obviated by cooperation between the distributed components. For example, host-based schedulers can take steps to reduce the negative impact of limited disk command queue lengths (see section 7.2.2).

The only disadvantage of distributed scheduling is the need to modify existing protocols to allow the necessary levels of cooperation between scheduling components. As with disk-based centralized scheduling, a small amount of system-level information must be passed between the host and each disk-based scheduler to expedite high priority requests. If the host scheduler is to reorder requests based on hardware-specific information, some configuration and state knowledge must be extracted and transferred to the host as well. However, if the amount of control information required for cooperation is minimized, only slight protocol modifications should be necessary.

All distributed scheduling experiments assume that the host-based scheduler has insufficient computation resources, hardware-specific knowledge, and/or hardware support to compute positioning times for a set of pending requests. Host-based schedulers in this chapter therefore use single-queue and 2Q versions of the C-LOOK.CW.SSR and SSTF.CW.SSR seek-reducing algorithms. Chapter 5 describes how these two algorithms generally provide the best performance for LBN-based scheduling at the host. The distributed scheduling experiments include disk-based schedulers with LBN-based and full-knowledge scheduling algorithms. Disk-based schedulers with full-knowledge algorithms are assumed to have sufficient resources and hardware information to compute positioning times for finite-length queues of pending requests. The experiments in this chapter are partitioned based on the type of information used by the schedulers.

7.1 Scheduling with Information From "Below"

This section examines the performance of distributed scheduler implementations using disk-based C-LOOK, SSTF, SPCTF, and ASPCTF(6). Both the host-based and the disk-based schedulers are therefore attempting to minimize mean response times (and response time variances, in the case of the age-sensitive SPCTF algorithm).

7.1.1 Disk Request Traces

LBN-based Scheduling at the Disk

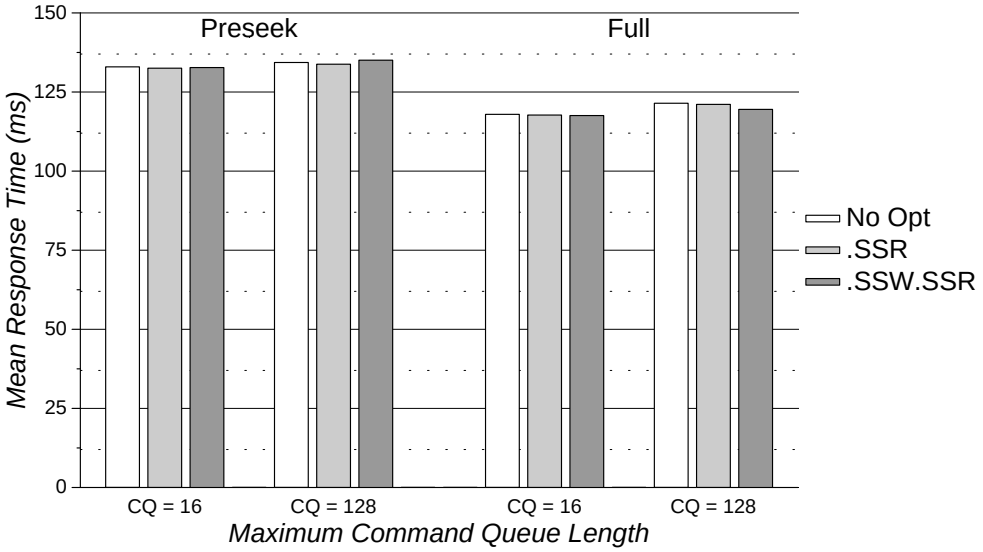
The experiments reported below examine the performance effects of sequential scheduling of read and write requests on disk-based SSTF implementations. C-LOOK is unaffected by the sequential scheduling optimization, since it already schedules requests in logically ascending order. Sequential scheduling of read requests at the disk consistently improves SSTF performance, especially for configurations with large maximum command queue lengths. Sequential scheduling of write requests marginally degrades performance for disks with Preseek command queueing and improves performance for disks with Full command queueing. This matches the previous observation that sequential scheduling of writes adversely affects the performance of disks without write prebuffering (see page 51 in section 5.1.2 for a full discussion of this behavior).

Figures 7.1–7.6 display mean response times for distributed scheduling of the six HP and DEC workloads. These figures show the improvement (or degradation) caused by adding sequential scheduling of reads and writes to a disk-based SSTF scheduler. Each graph shows data for four configurations, comparing maximum command queue lengths of 16 and 128 for disks with either Preseek or Full command queueing. With a host-based C-LOOK.CW.SSR scheduler, sequential scheduling of read requests by a disk-based SSTF scheduler decreases mean response times by up to 24% and 35% for maximum command queue lengths of 16 and 128, respectively (using the selected trace scaling factors). With a host-based SSTF.CW.SSR scheduler, the mean response times decrease by up to 30% and 36%. For experiments using Full command queueing, sequentially scheduling write requests further decreases mean response times by up to 1.5%.

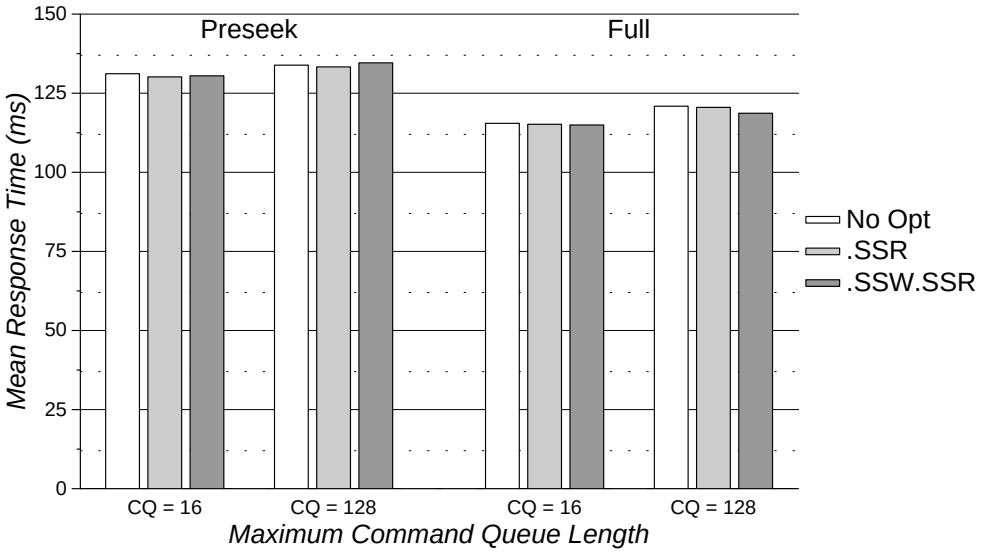
As the *Snake* workload is better served by a host-based scheduler that concatenates read requests (rather than sequentially scheduling them), figure 7.7 provides additional data for distributed scheduling using host-based C-LOOK.CW.CR and SSTF.CW.CR. The mean response times for the configurations with a maximum command queue length of 16 drop approximately 9% when the host-based scheduler concatenates reads instead of sequentially scheduling them. Configurations with a maximum queue length of 128 gain little benefit from this change, as the host only concatenates requests when the disk subsystem is extremely busy (i.e., when more than 129 requests are outstanding for a single disk).

Full-Knowledge Scheduling at the Disk

SPCTF and ASPCTF(6) experience the same general reaction to sequential scheduling of read and write requests. Figures 7.8–7.14 display mean response times for distributed implementations using disk-based SPCTF scheduling. They show that sequential scheduling

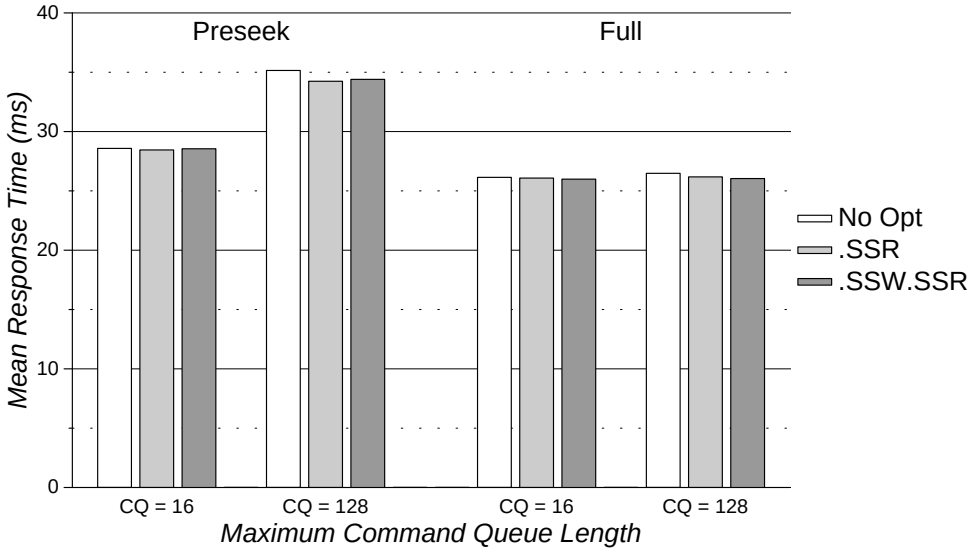


(a) Host-Based C-LOOK.CW.SSR

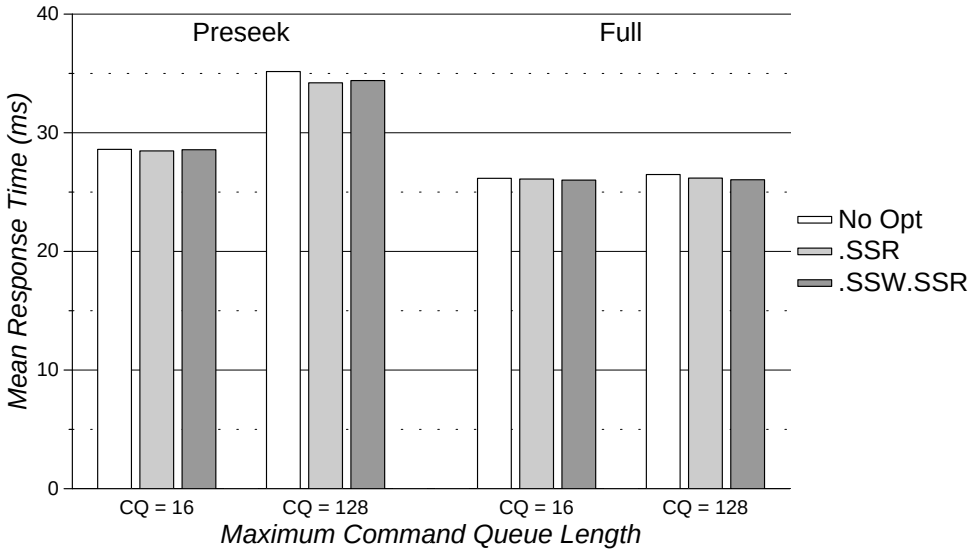


(b) Host-Based SSTF.CW.SSR

Figure 7.1: *Cello*, 1.75X: Disk-Based SSTF Performance

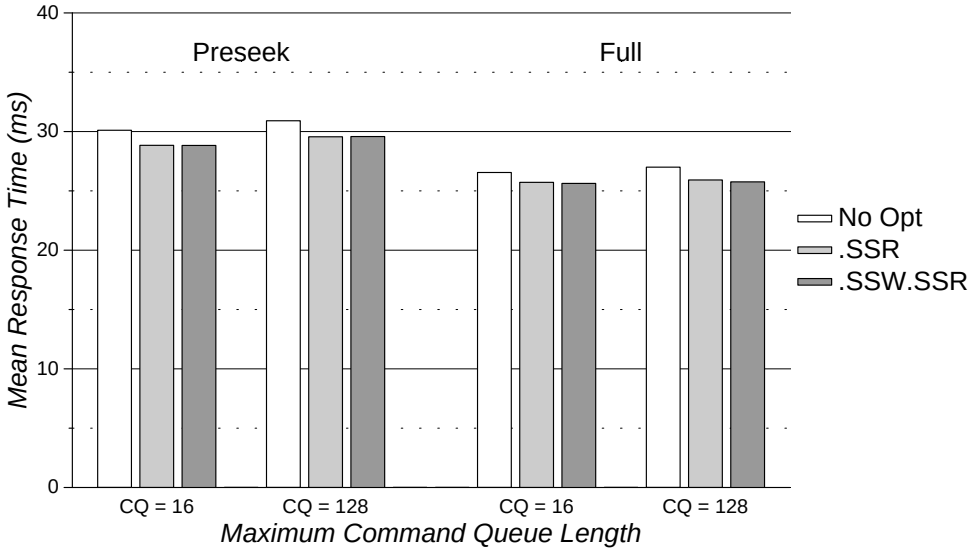


(a) Host-Based C-LOOK.CW.SSR

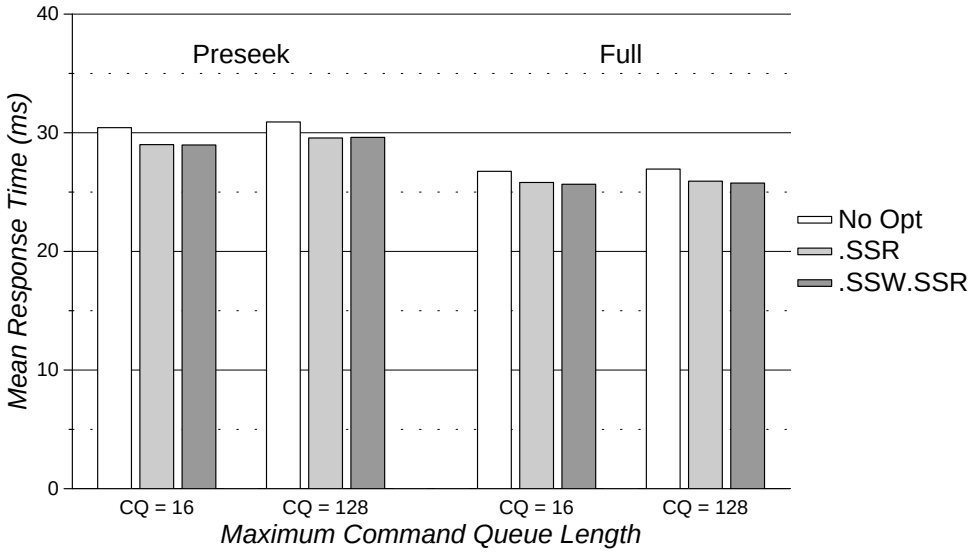


(b) Host-Based SSTF.CW.SSR

Figure 7.2: *Snake*, 1.25X: Disk-Based SSTF Performance

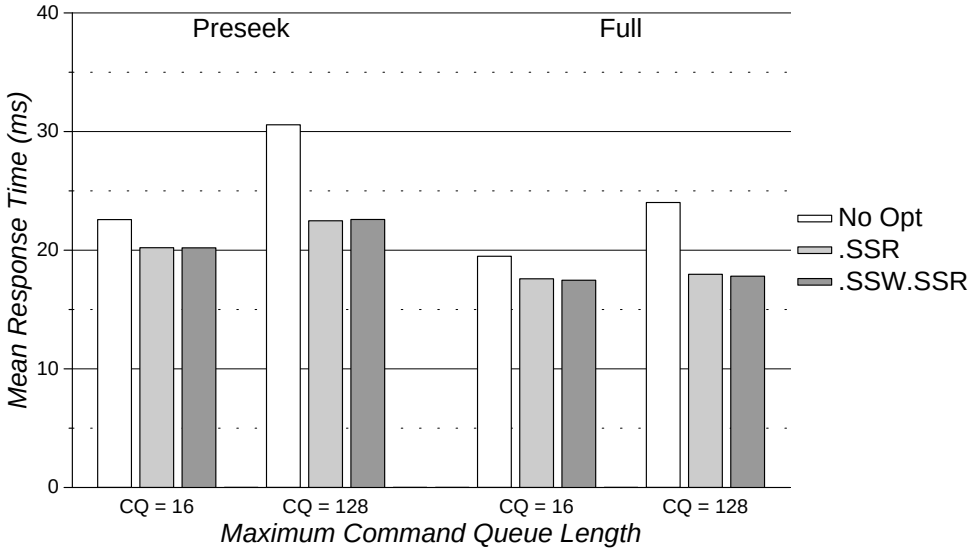


(a) Host-Based C-LOOK.CW.SSR

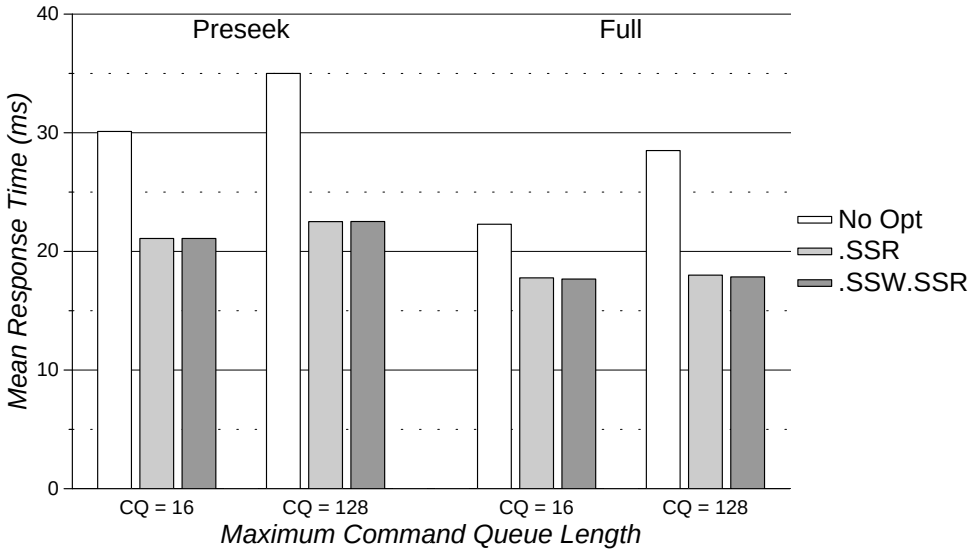


(b) Host-Based SSTF.CW.SSR

Figure 7.3: *Air-Rsv*, 2.5X: Disk-Based SSTF Performance

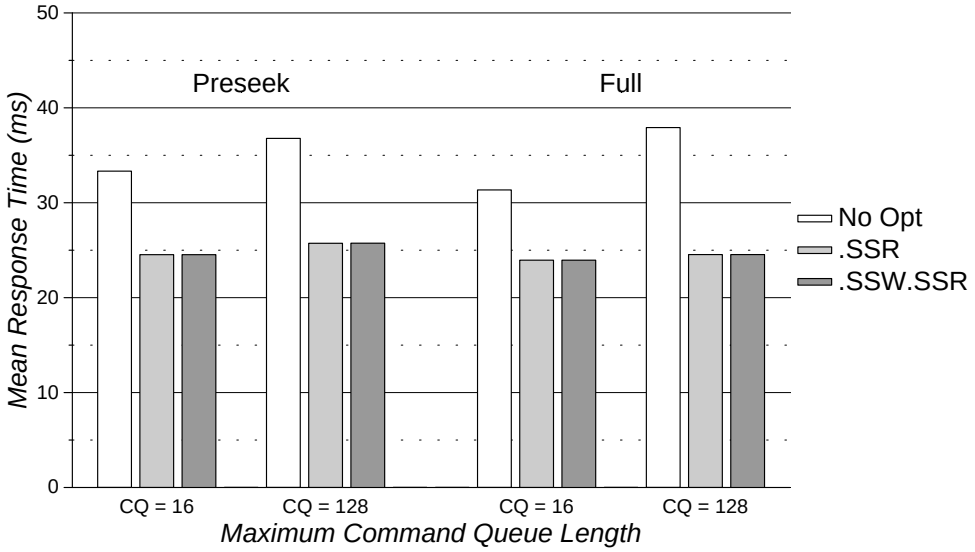


(a) Host-Based C-LOOK.CW.SSR

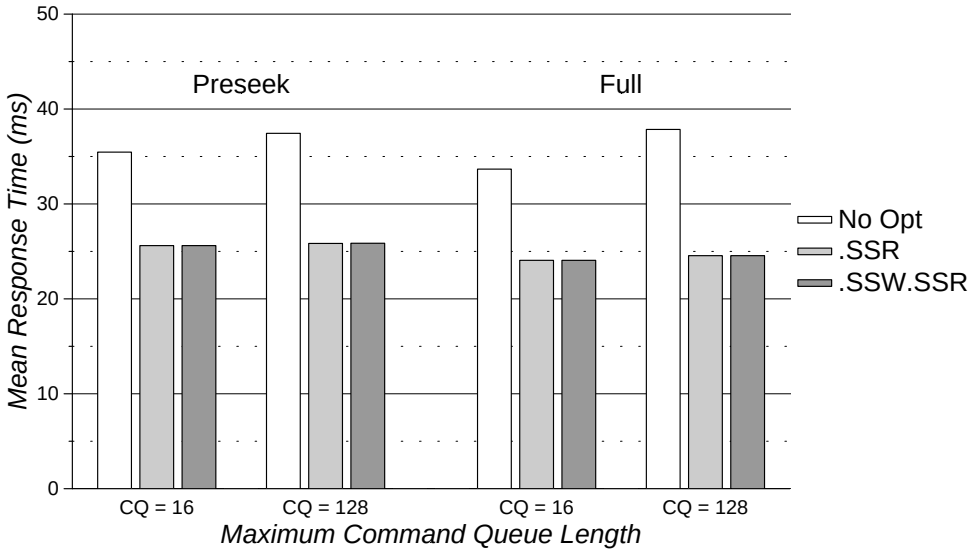


(b) Host-Based SSTF.CW.SSR

Figure 7.4: *Sci-TS*, 2.5X: Disk-Based SSTF Performance

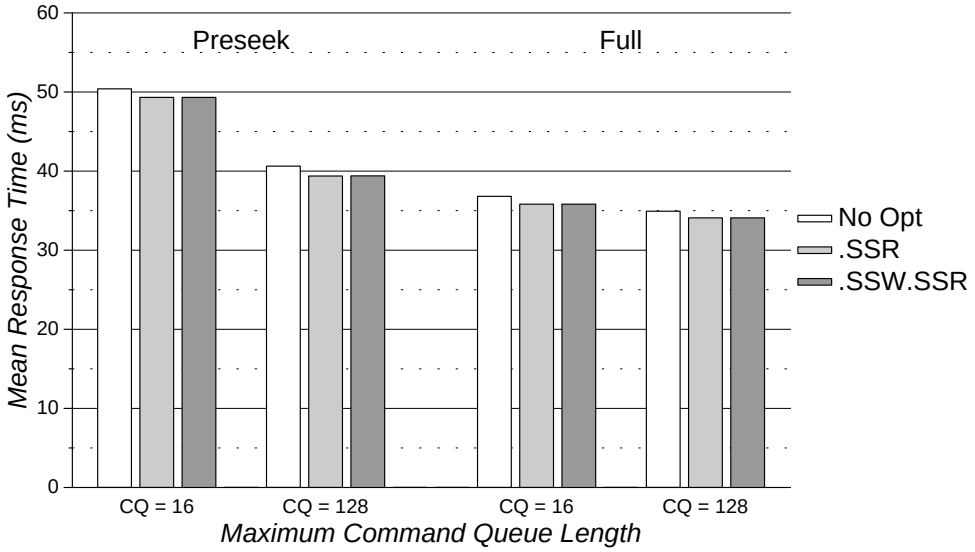


(a) Host-Based C-LOOK.CW.SSR

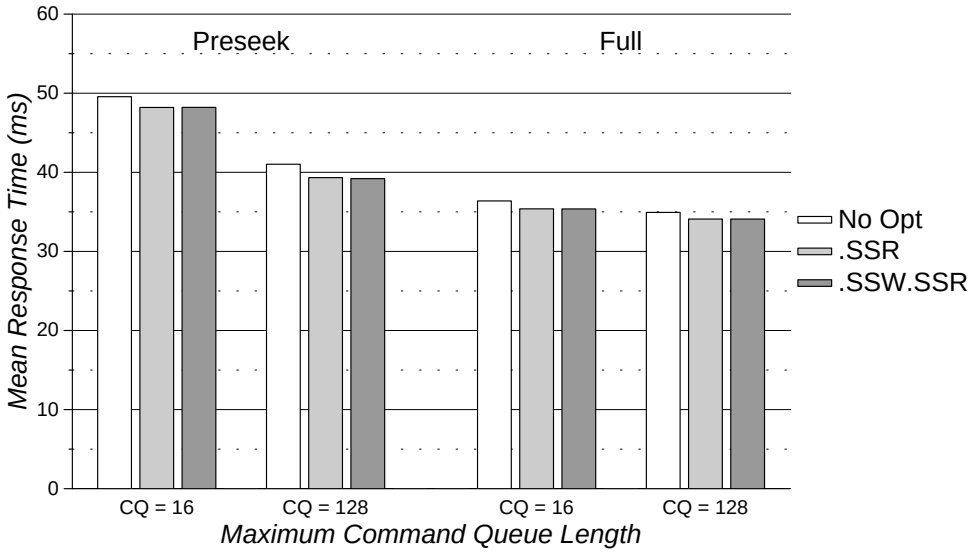


(b) Host-Based SSTF.CW.SSR

Figure 7.5: *Order* 1.0X: Disk-Based SSTF Performance

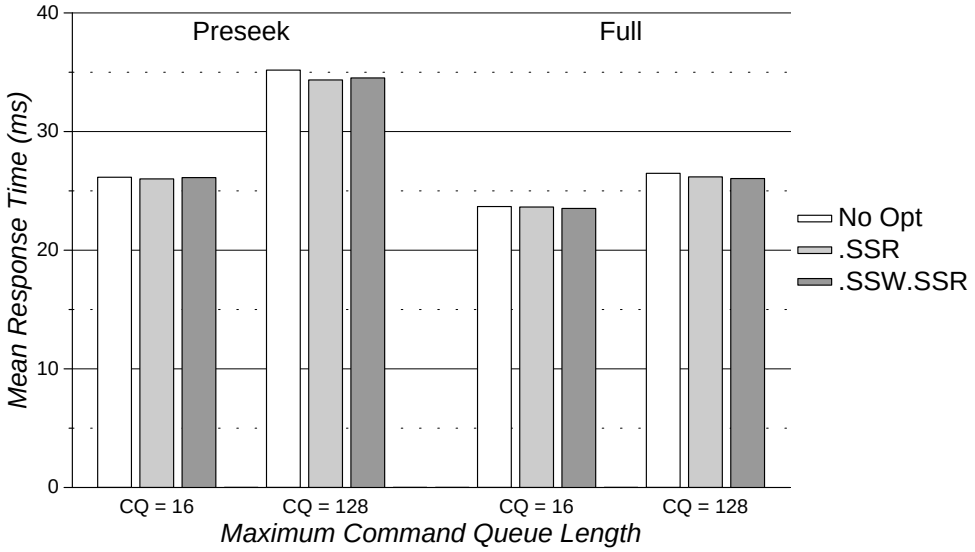


(a) Host-Based C-LOOK.CW.SSR

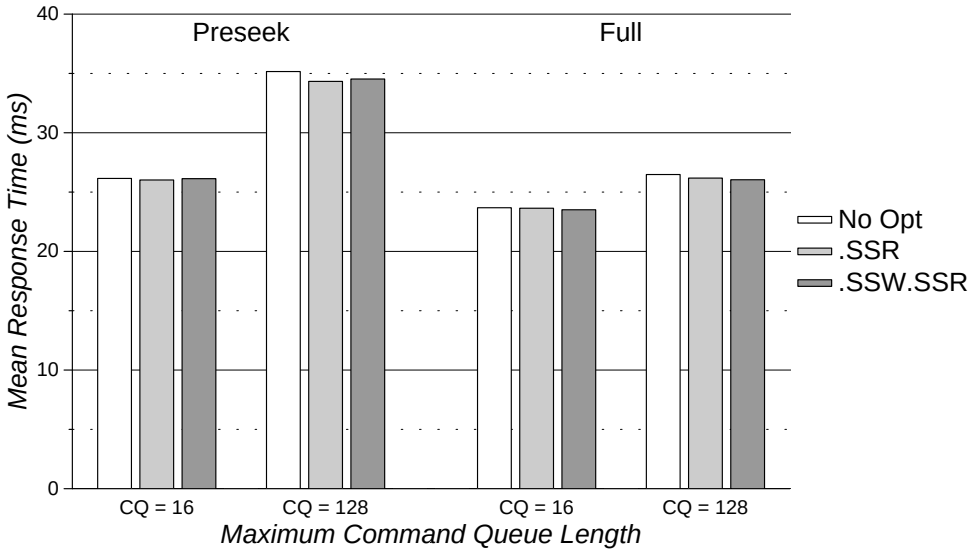


(b) Host-Based SSTF.CW.SSR

Figure 7.6: *Report*, 1.0X: Disk-Based SSTF Performance

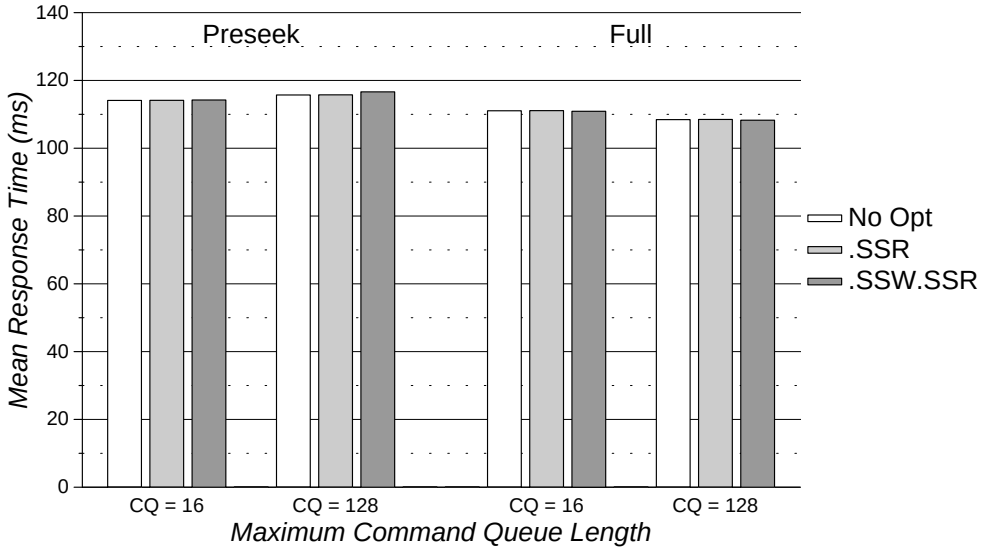


(a) Host-Based C-LOOK.CW.CR

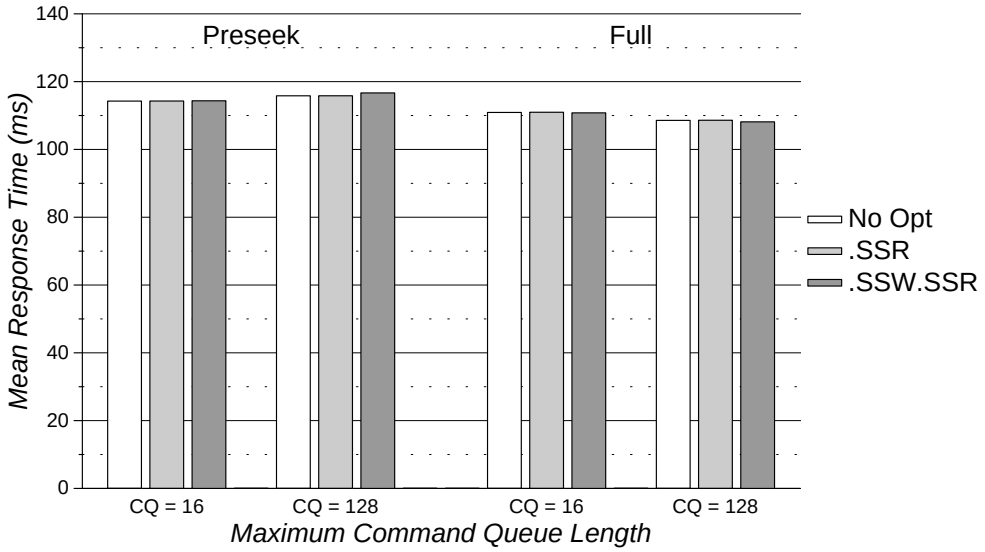


(b) Host-Based SSTF.CW.CR

Figure 7.7: *Snake*, 1.25X: Disk-Based SSTF Performance

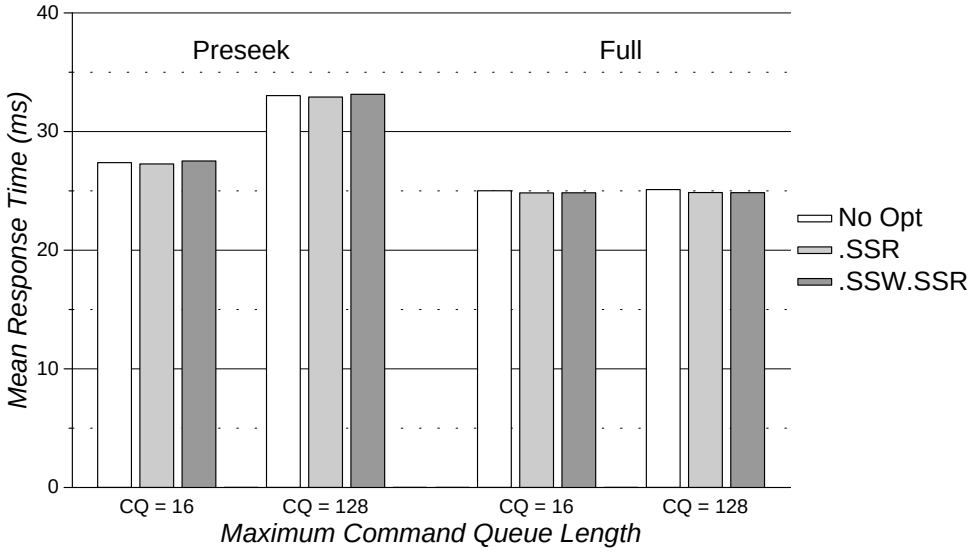


(a) Host-Based C-LOOK.CW.SSR

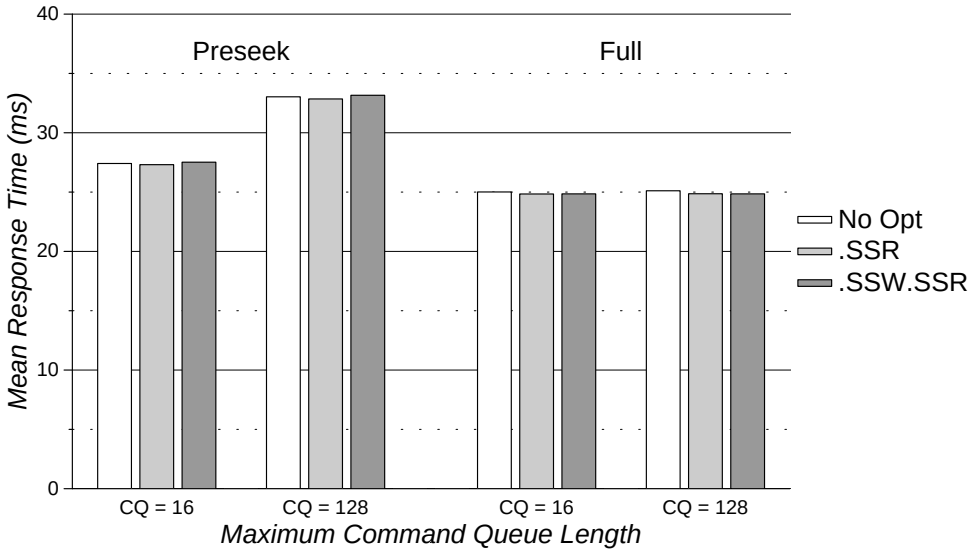


(b) Host-Based SSTF.CW.SSR

Figure 7.8: *Cello*, 1.75X: Disk-Based SPCTF Performance

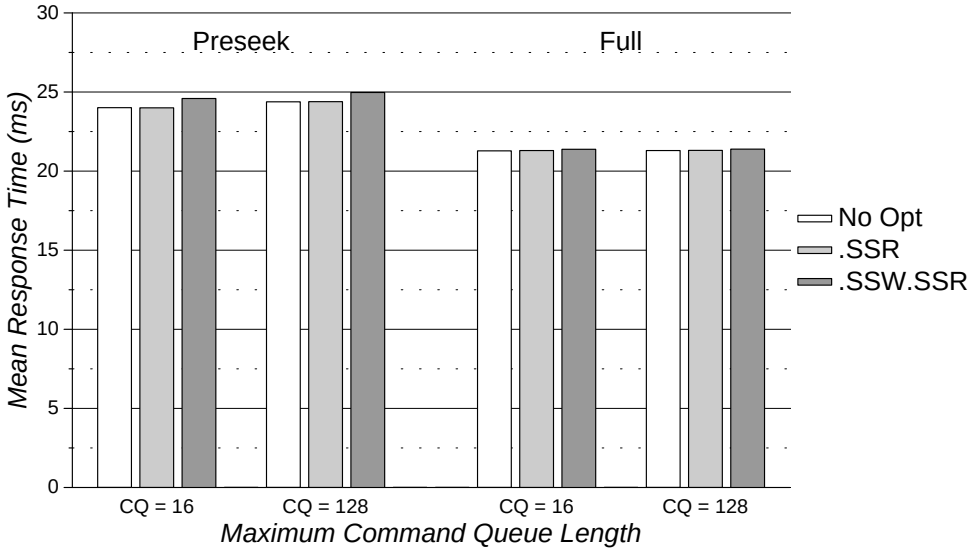


(a) Host-Based C-LOOK.CW.SSR

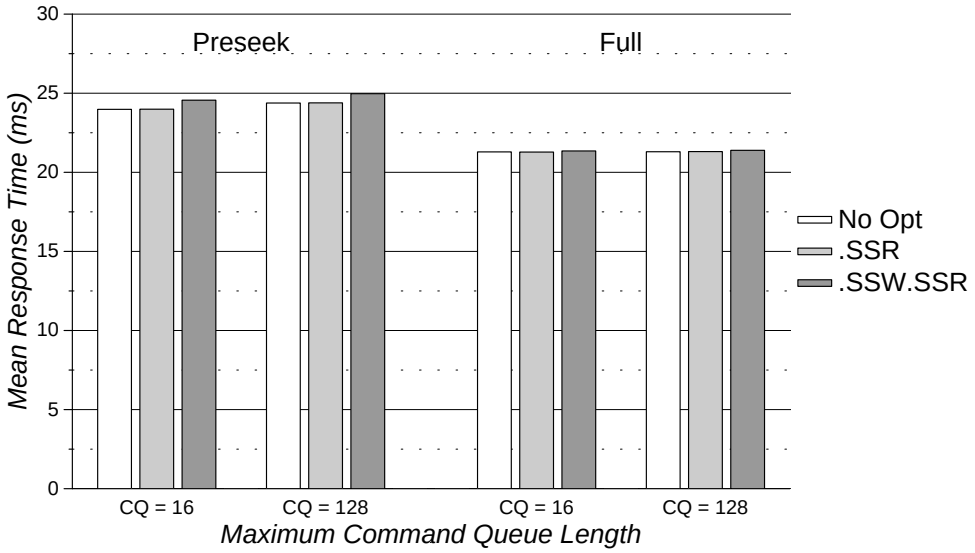


(b) Host-Based SSTF.CW.SSR

Figure 7.9: *Snake*, 1.25X: Disk-Based SPCTF Performance

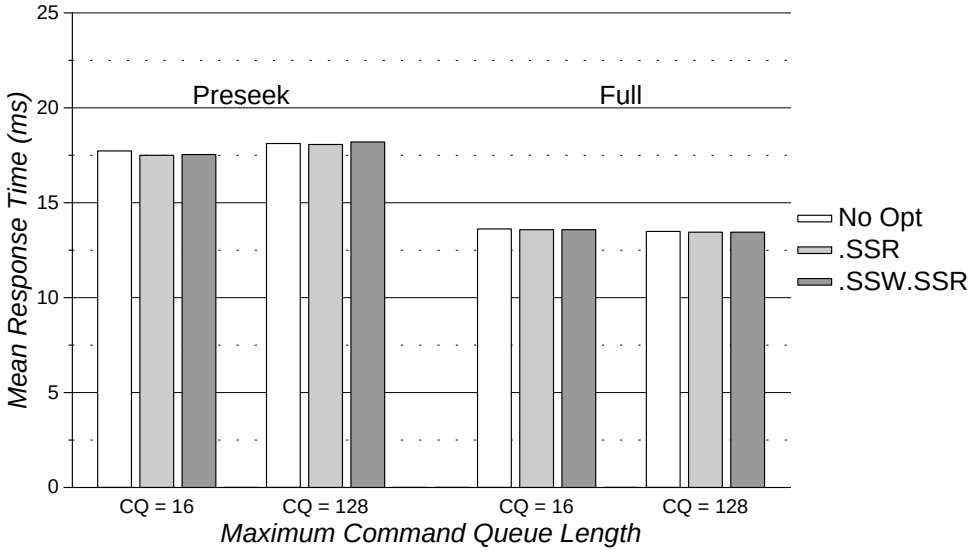


(a) Host-Based C-LOOK.CW.SSR

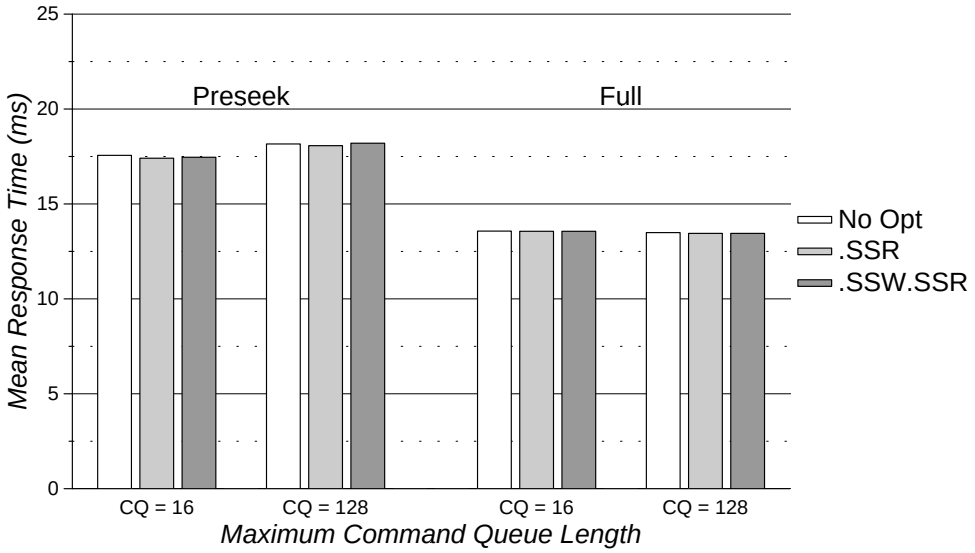


(b) Host-Based SSTF.CW.SSR

Figure 7.10: *Air-Rsv*, 2.5X: Disk-Based SPCTF Performance

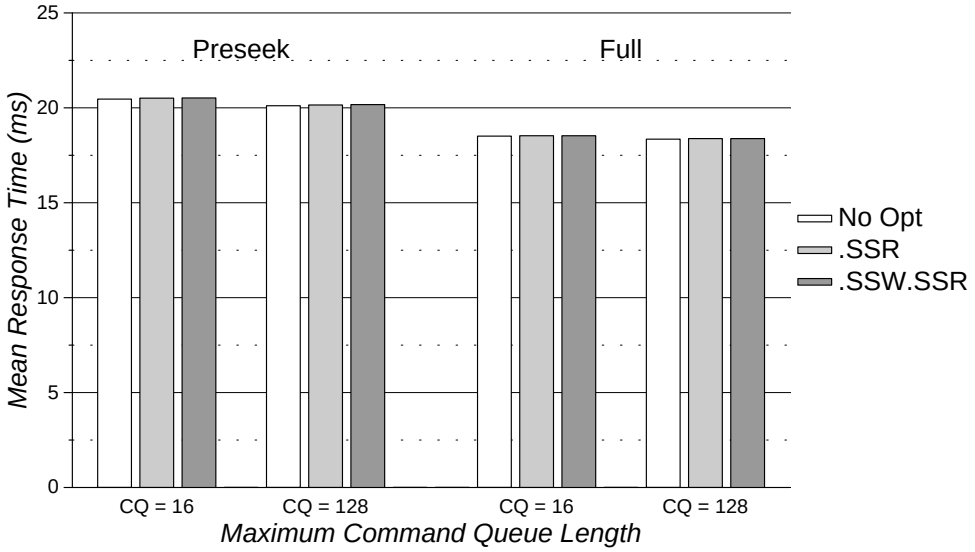


(a) Host-Based C-LOOK.CW.SSR

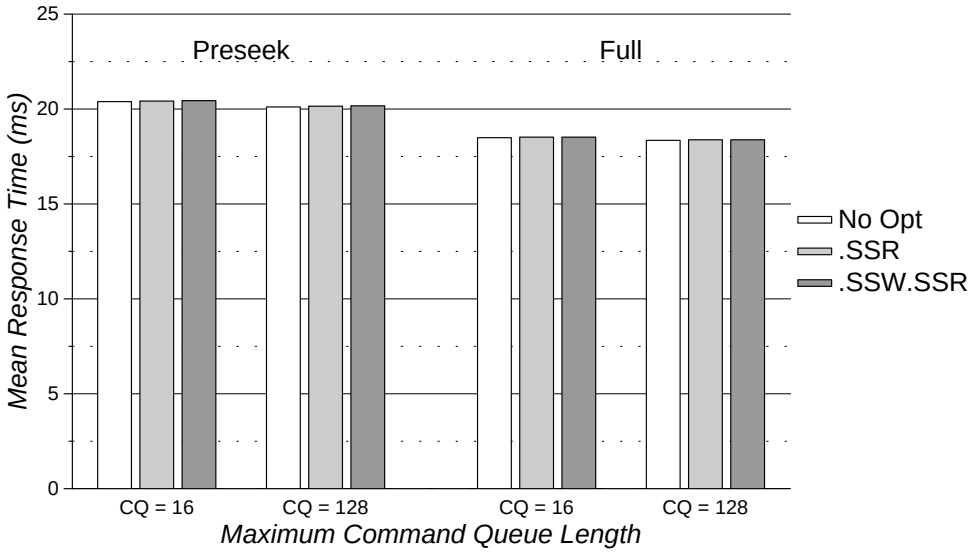


(b) Host-Based SSTF.CW.SSR

Figure 7.11: *Sci-TS*, 2.5X: Disk-Based SPCTF Performance

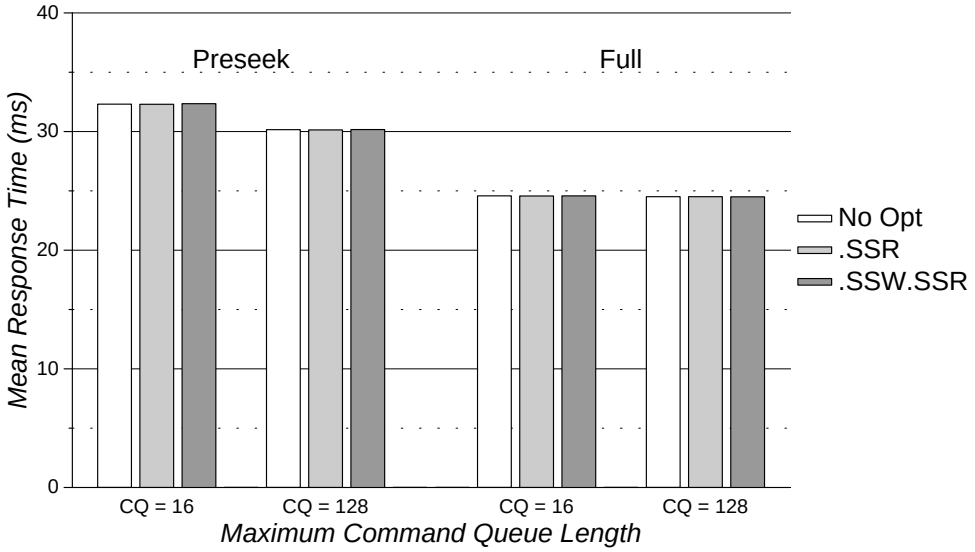


(a) Host-Based C-LOOK.CW.SSR

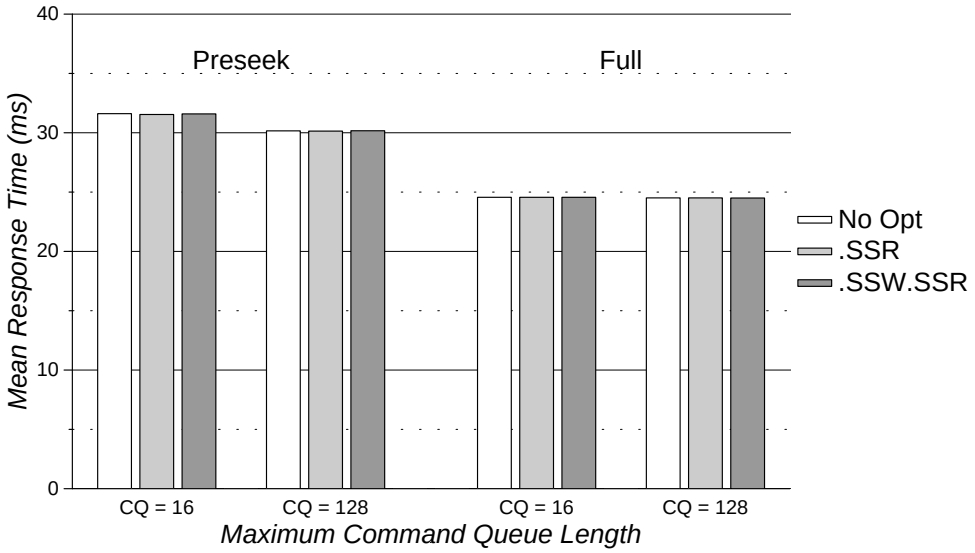


(b) Host-Based SSTF.CW.SSR

Figure 7.12: *Order*, 1.0X: Disk-Based SPCTF Performance

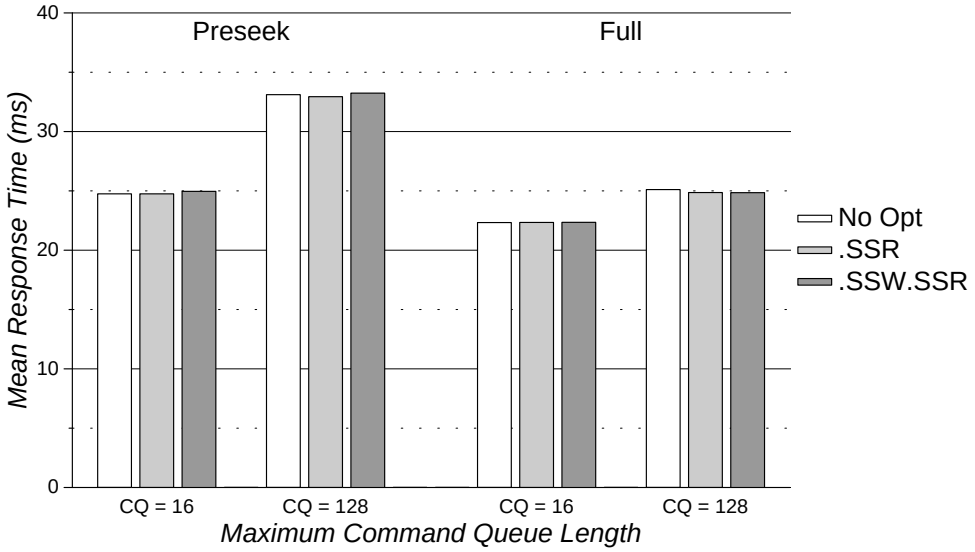


(a) Host-Based C-LOOK.CW.SSR

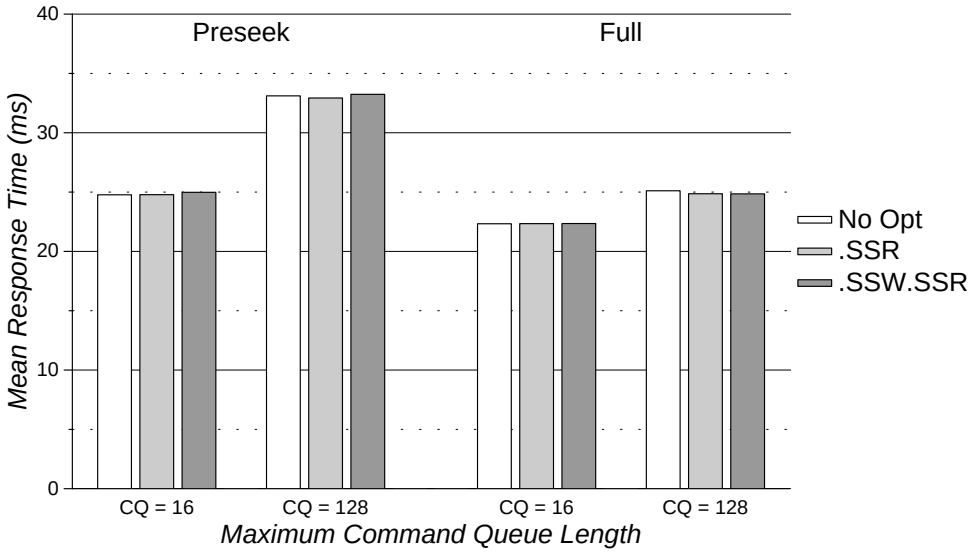


(b) Host-Based SSTF.CW.SSR

Figure 7.13: *Report*, 1.0X: Disk-Based SPCTF Performance



(a) Host-Based C-LOOK.CW.CR



(b) Host-Based SSTF.CW.CR

Figure 7.14: *Snake*, 1.25X: Disk-Based SPCTF Performance

of read requests with a disk-based SPTF scheduler impacts mean response times by 1% or less. Sequential scheduling of write requests degrades performance by up to 2.4% for disks with Preseek command queueing. Disks with Full command queueing experience less than a 1% change in performance with sequential scheduling of writes. Figures 7.9 and 7.14 show a 10% decrease in mean response times for *Snake* configurations with a maximum command queue length of 16 when the host-based scheduler changes from sequential scheduling of read requests to concatenation of sequential reads.

For both LBN-based and full-knowledge scheduling, it is beneficial to sequentially schedule read requests at the disk for distributed scheduling of the disk request traces. Furthermore, sequential scheduling of write requests sometimes provides a performance improvement when usings disks with Full command queueing. Therefore, all remaining experiments in this chapter using the HP and DEC traces study disk-based scheduling algorithms with sequential scheduling of reads. All experiments modeling disks with Full command queueing use sequential scheduling of writes as well.

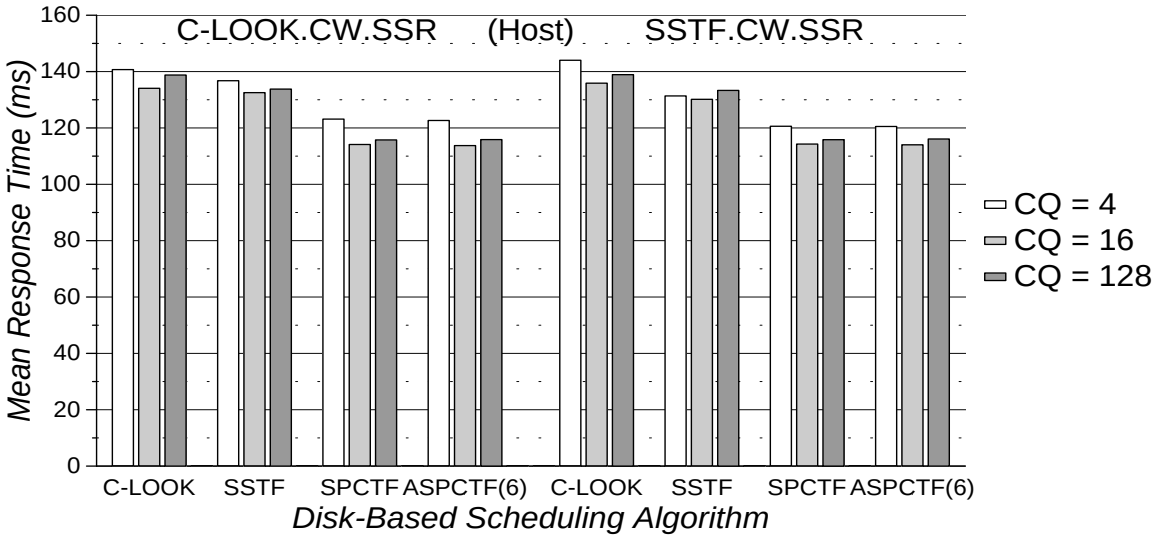
Maximum Command Queue Length

Figures 7.15–7.20 show mean response times for distributed scheduler implementations with maximum command queue lengths of 4, 16, and 128. More often than not, a maximum command queue length of 16 results in mean response times that are better or roughly equivalent to those for configurations with maximum queue lengths of 4 or 128. Most exceptions to this generalization are in three areas. First, a maximum command queue length of 4 provides 3–6% lower mean response times for scheduling *Cello* with Full command queueing (as compared to a maximum length of 16). Second, for *Sci-TS* and *Order*, a maximum command queue length of 128 provides up to 5.6% lower mean response times for some configurations of disk-based C-LOOK scheduling. Third and most important, a maximum command queue length of 16 is ill-advised for distributed scheduling of the *Report* workload using LBN-based scheduling on disks with Preseek command queueing. Maximum command queue lengths of either 4 or 128 provide much better performance for these configurations.

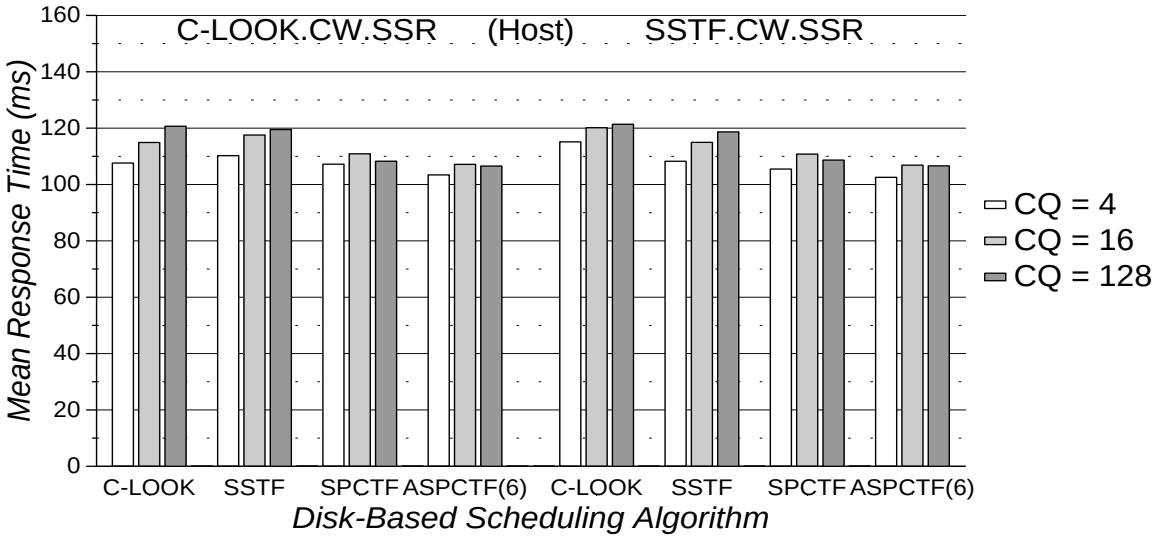
As *Snake* benefits more from concatenation of sequential read requests than from sequential scheduling of reads, figure 7.21 shows performance data for distributed implementations using C-LOOK.CW.CR and SSTF.CW.CR at the host. Compared with figure 7.16, the mean response times for configurations with maximum command queue lengths of 4 drop by approximately 30%. As a result, the mean response times for these configurations now outperform those using maximum queue lengths of 16 and 128. Thus, both *Cello* and *Snake* are best served by distributed scheduler implementations that allow the host-based scheduler to concatenate as many requests as possible, while allowing a small amount of inter-request concurrency at each disk. The data in table 6.1 supports this observation; section 6.1.1 shows that LBN-based scheduling at the host always outperforms LBN-based scheduling at the disk for *Cello* and *Snake*.

Comparison of Distributed Scheduling Algorithms

Across all six workloads, the difference in performance due to switching between C-LOOK and SSTF host-based schedulers decreases with the maximum command queue length. For a

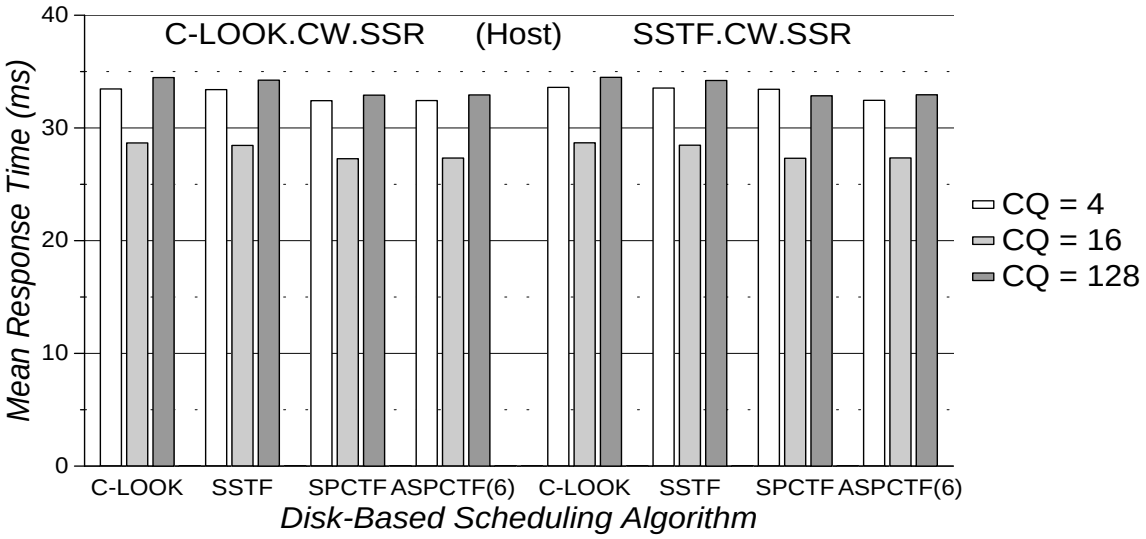


(a) Preseek Command Queueing

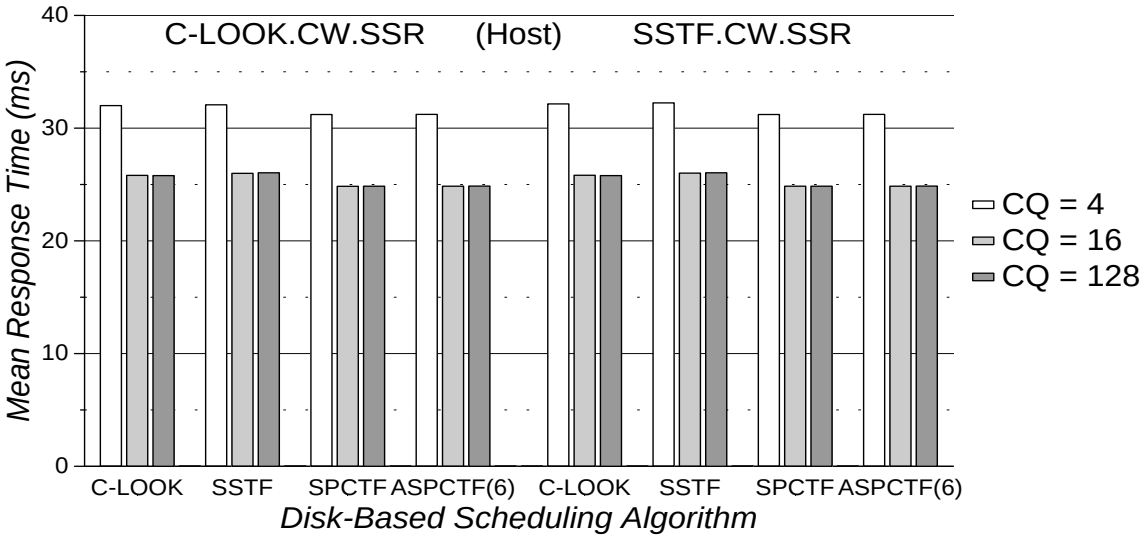


(b) Full Command Queueing

Figure 7.15: *Cello*, 1.75X: Distributed Scheduling Algorithm Performance

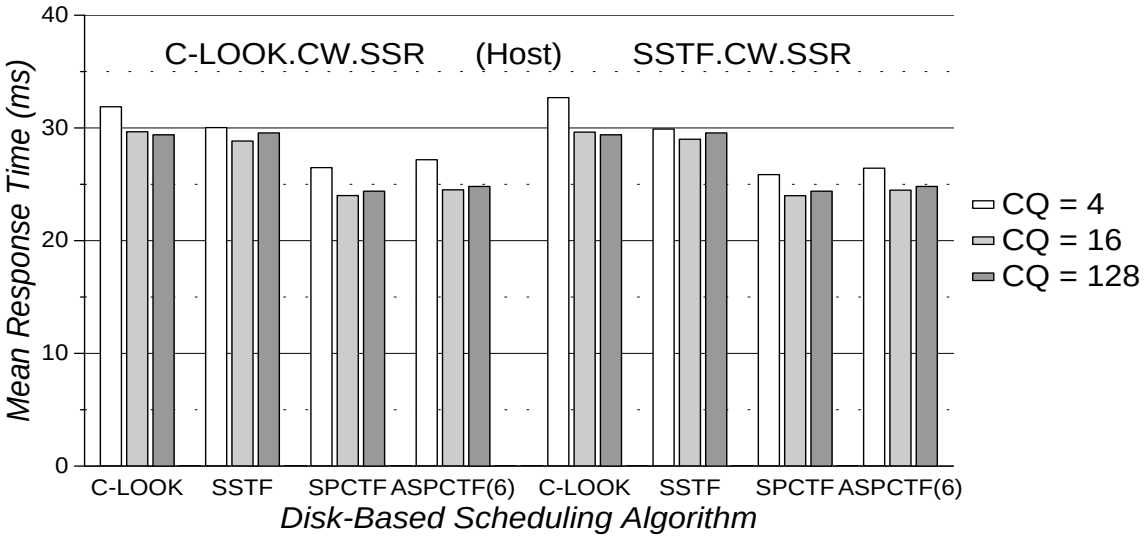


(a) Preseek Command Queueing

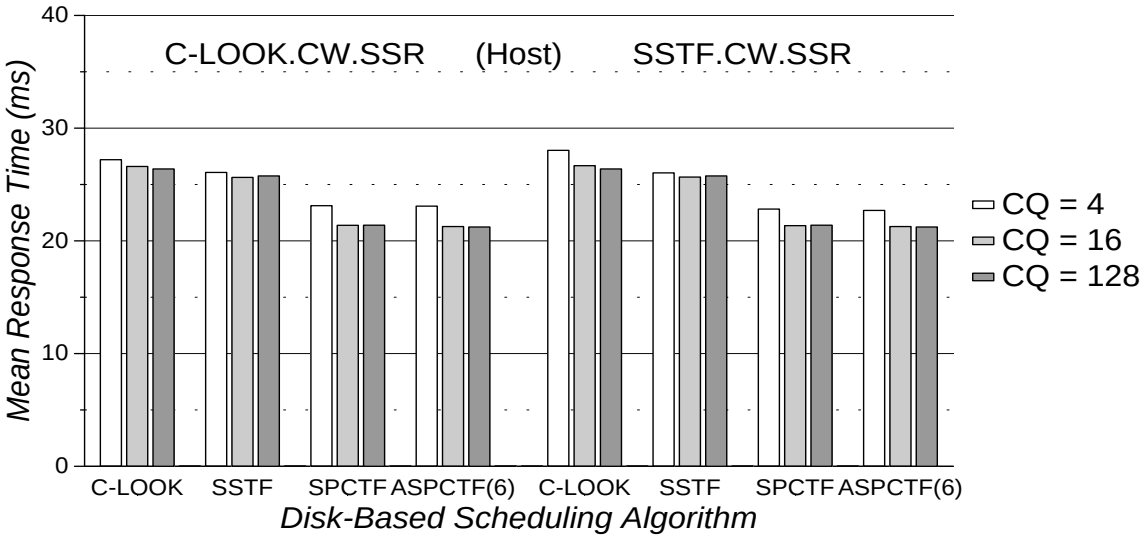


(b) Full Command Queueing

Figure 7.16: *Snake*, 1.25X: Distributed Scheduling Algorithm Performance

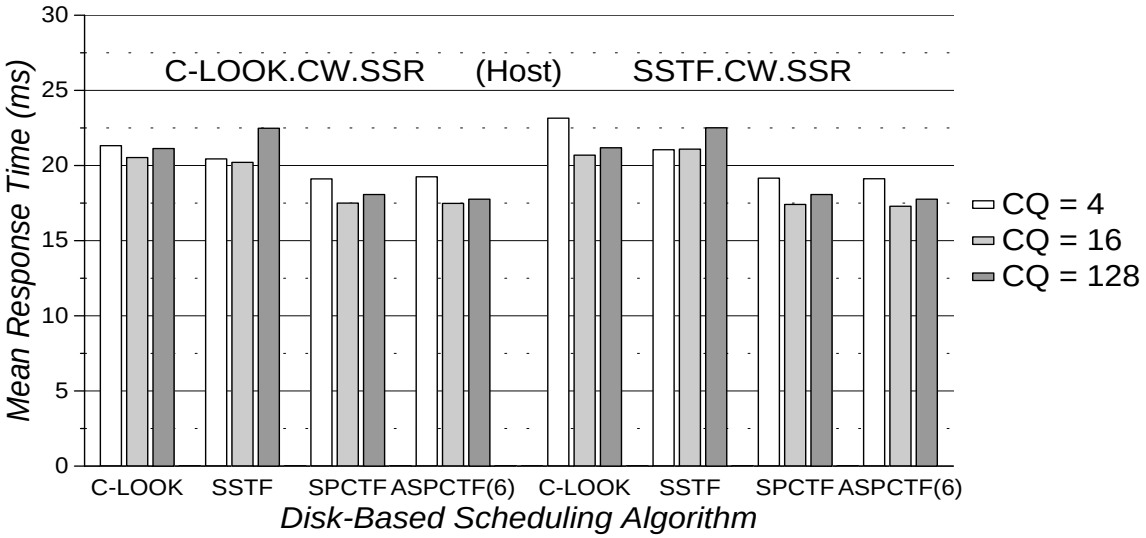


(a) Preseek Command Queueing

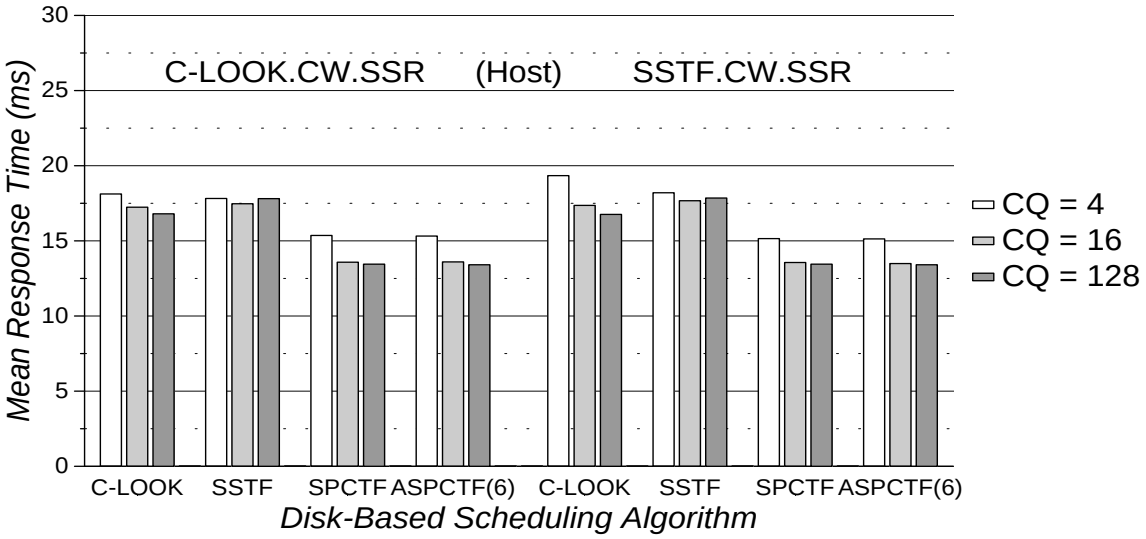


(b) Full Command Queueing

Figure 7.17: *Air-Rsv*, 2.5X: Distributed Scheduling Algorithm Performance

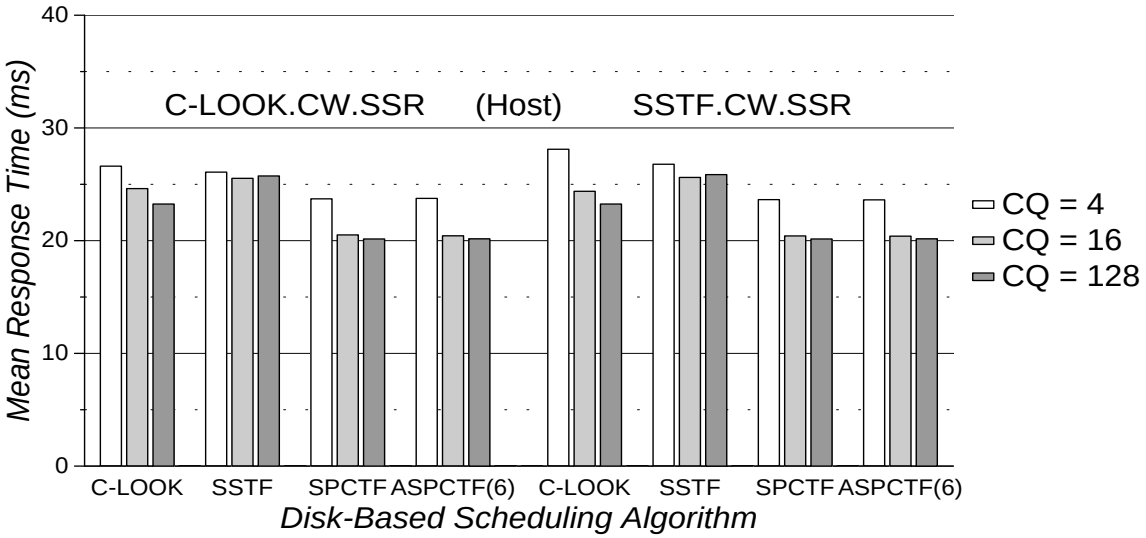


(a) Preseek Command Queueing

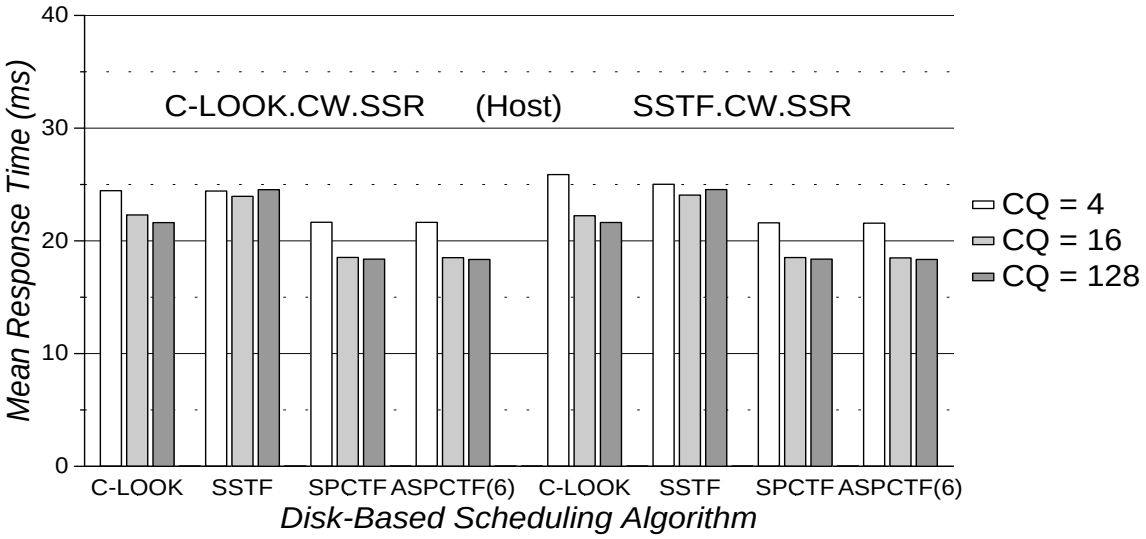


(b) Full Command Queueing

Figure 7.18: *Sci-TS*, 2.5X: Distributed Scheduling Algorithm Performance

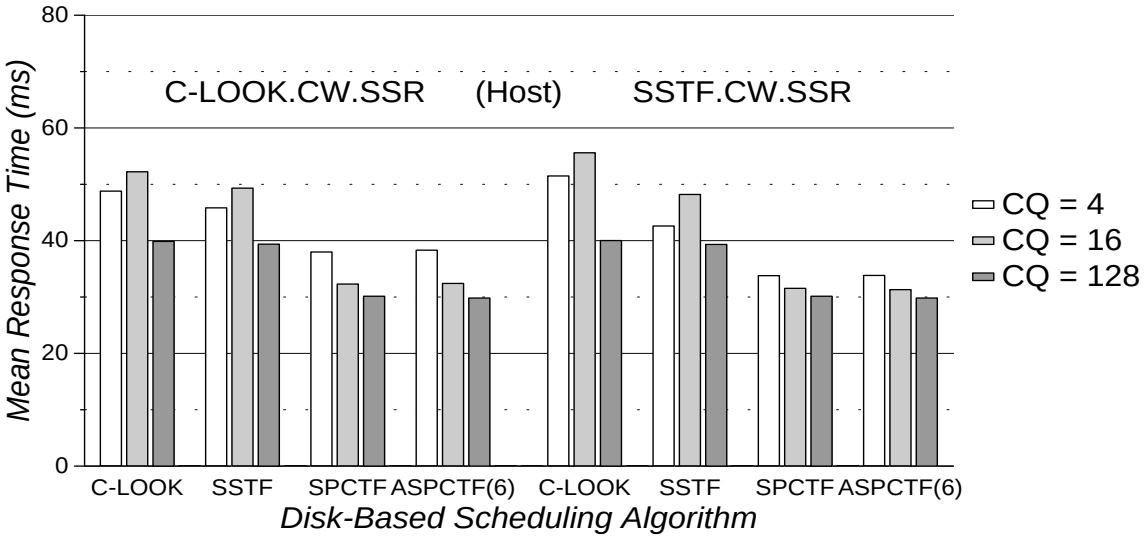


(a) Preseek Command Queueing

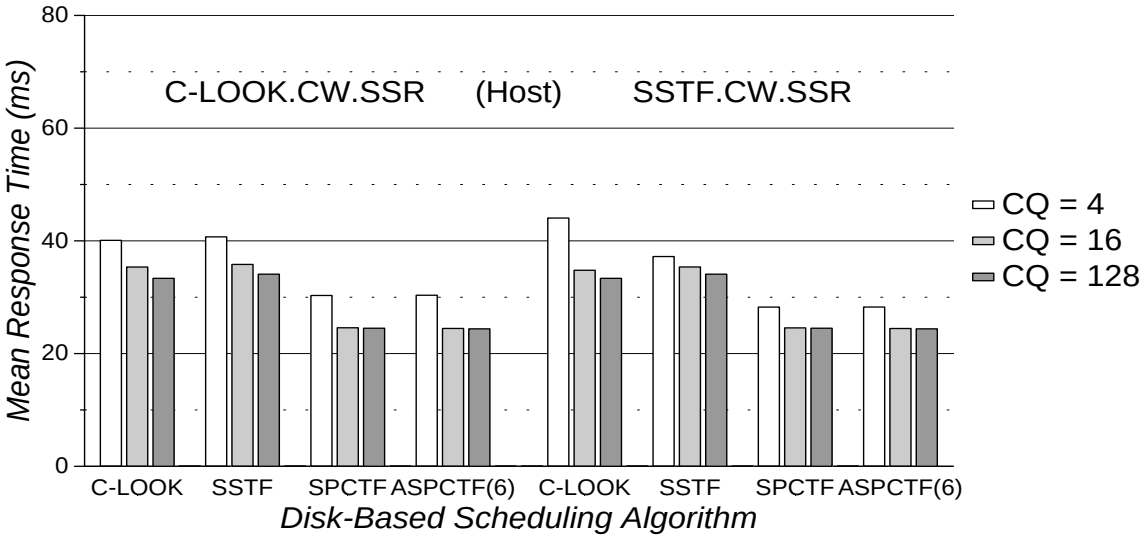


(b) Full Command Queueing

Figure 7.19: *Order*, 1.0X: Distributed Scheduling Algorithm Performance

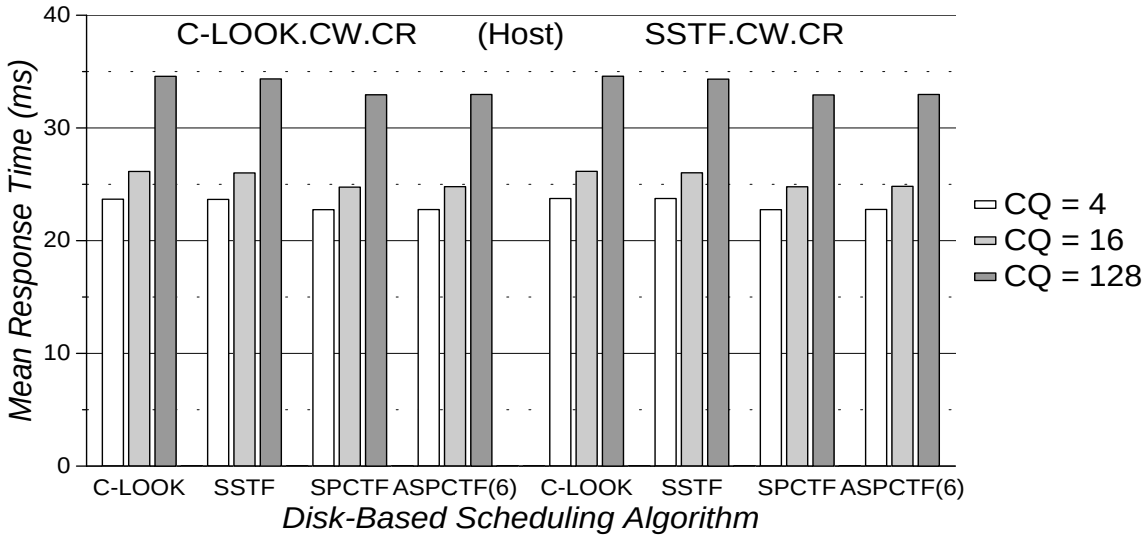


(a) Preseek Command Queueing

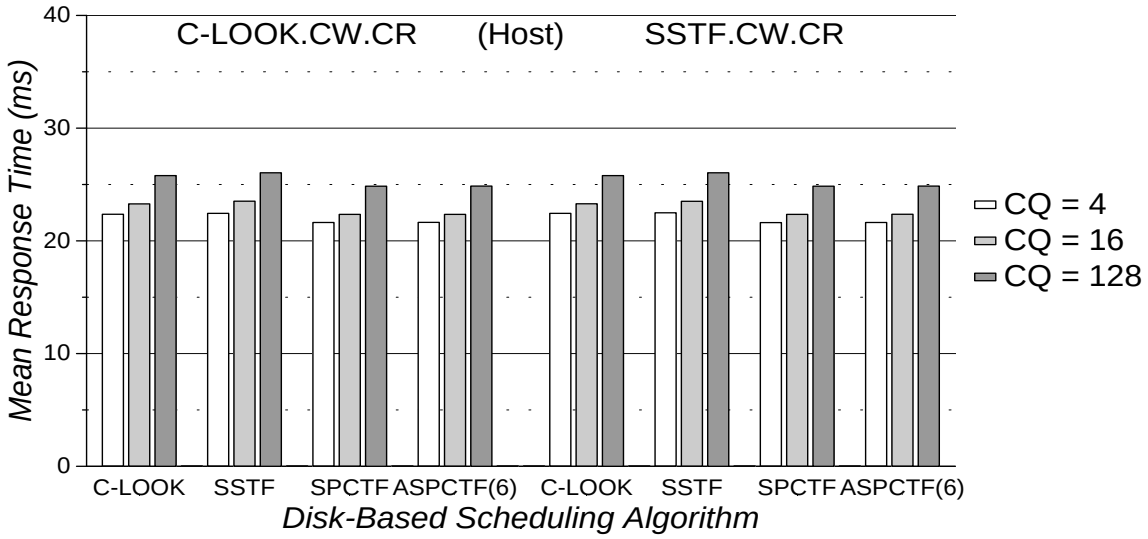


(b) Full Command Queueing

Figure 7.20: *Report*, 1.0X: Distributed Scheduling Algorithm Performance



(a) Preseek Command Queueing



(b) Full Command Queueing

Figure 7.21: *Snake*, 1.25X: Distributed Scheduling Algorithm Performance

maximum queue length of 16, the change in mean response time rarely exceeds 1% for workloads other than *Cello*. The most noticeable exception is the 6% performance advantage of host-based C-LOOK over host-based SSTF when scheduling *Report* using disks with Preseek command queueing. For a maximum queue length of 128, the choice of host-based scheduler affects mean response times by 1% or less. In terms of individual workloads, *Cello* is the most sensitive to the host-based scheduling algorithm. For most configurations, a dual SSTF scheduler implementation provides the lowest (or equivalent) mean response times among the LBN-based algorithms when scheduling the *Cello* trace.

The choice between C-LOOK and SSTF for disk-based scheduling is workload dependent. C-LOOK generally provides lower mean response times for *Sci-TS* and *Order*, whereas SSTF generally provides lower mean response times for *Cello* and *Air-Rsv*. SSTF also outperforms C-LOOK for *Snake* and *Report* when using Preseek command queueing. This behavior deviates from that of centralized scheduling, where C-LOOK outperforms SSTF for all workloads except *Cello*. Since the sequential stream optimizations improve SSTF more than C-LOOK, it is not surprising that the two algorithms perform equally with two levels of sequential stream optimizations in place. That is, with SSTF servicing sequential requests in a manner similar to that of C-LOOK, the shorter mean seek times incurred by SSTF frequently result in superior mean response times.

For distributed scheduling of the disk request traces, SPTF-based algorithms outperform LBN-based algorithms at the disk. As was true for centralized scheduling, adding age-sensitivity to SPCTF improves or degrades performance depending on the workload. However, the absolute change in mean response time is rarely greater than 1–2%. Section 7.2.1 compares the starvation resistance of the various distributed scheduler implementations (as measured by the squared coefficients of response time variation).

Comparison with Centralized Scheduling

Table 7.1 shows performance data for the "best" LBN-based centralized and distributed scheduler implementations from the experiments covered by this dissertation. For configurations with Preseek command queueing, distributed scheduling has up to 3% higher performance for all six HP and DEC traces. When Full command queueing is enabled, the distributed and centralized implementations provide roughly equivalent performance for all workloads except *Cello*, where the host-based centralized scheduler has a 5% lower mean response time. Therefore, the choice of a centralized versus a distributed implementation is not crucial to achieving good LBN-based scheduler performance.

Table 7.2 shows performance data for the "best" centralized and distributed scheduler implementations using the SPCTF full-knowledge scheduling algorithm. For Preseek and Full command queueing configurations, distributed scheduling provides equivalent or superior mean response times for all six disk request traces. For the *Snake* trace, distributed schedulers produce 14–18% lower mean response times. For configurations with Preseek command queueing, distributed scheduling of the HP traces also has a significant complexity advantage. The best centralized location for these traces is the host, and host-based SPCTF scheduling requires a significant effort; all disk-specific information related to data layout, mechanical and overhead delays, cache contents, and current rotational position must be extracted and transferred to the host-based scheduler. With distributed scheduling, the

Trace Scale		Host Sched (all .CW)	Disk Sched (all .SSR)	CQ	Mean Resp Time (ms)
Cello 1.75	Central	SSTF.SSR		2	133.19
	Distrib	SSTF.SSR	SSTF	16	**130.16**
Snake 1.25	Central	C-LOOK.CR		2	23.75
	Distrib	C-LOOK.CR	C-LOOK	4	23.68
Air-Rsv 2.5	Central		C-LOOK	128	29.52
	Distrib	C-LOOK.SSR	SSTF	16	**28.84**
Sci-TS 2.5	Central	C-LOOK.CR		2	20.63
	Distrib	C-LOOK.SSR	SSTF	16	**20.21**
Order 1.0	Central		C-LOOK	128	23.26
	Distrib	SSTF.SSR	C-LOOK	128	23.25
Report 1.0	Central	C-LOOK.CR		2	40.51
	Distrib	SSTF.SSR	SSTF	128	**39.33**

(a) Preseek Command Queueing

Trace Scale		Host Sched (all .CW)	Disk Sched (all .SSW.SSR)	CQ	Mean Resp Time (ms)
Cello 1.75	Central	SSTF.SSR		2	**101.96**
	Distrib	C-LOOK.SSR	C-LOOK	4	107.62
Snake 1.25	Central	C-LOOK.CR		2	22.68
	Distrib	C-LOOK.CR	C-LOOK	4	**22.36**
Air-Rsv 2.5	Central		SSTF	128	25.76
	Distrib	C-LOOK.SSR	SSTF	16	25.63
Sci-TS 2.5	Central		C-LOOK	128	16.79
	Distrib	SSTF.SSR	C-LOOK	128	16.76
Order 1.0	Central		C-LOOK	128	21.63
	Distrib	C-LOOK.SSR	C-LOOK	128	21.63
Report 1.0	Central		C-LOOK	128	33.35
	Distrib	C-LOOK.SSR	C-LOOK	128	33.35

(b) Full Command Queueing

Table 7.1: Mean Response Times for "Reasonable" LBN-Based Schedulers

Trace Scale		Host Sched (all .CW)	Disk Sched	CQ	Mean Resp Time (ms)
Cello 1.75	Central	SPCTF.CR		1	116.72
	Distrib	C-LOOK.SSR	SPCTF.SSR	16	**114.13**
Snake 1.25	Central	SPCTF.CR		1	26.94
	Distrib	C-LOOK.CR	SPCTF.SSR	4	**22.75**
Air-Rsv 2.5	Central		SPCTF	128	24.52
	Distrib	SSTF.SSR	SPCTF.SSR	16	**23.99**
Sci-TS 2.5	Central		SPCTF	128	18.79
	Distrib	SSTF.SSR	SPCTF.SSR	16	**17.40**
Order 1.0	Central		SPCTF	128	20.11
	Distrib	C-LOOK.SSR	SPCTF.SSR	128	20.15
Report 1.0	Central		SPCTF	128	30.16
	Distrib	C-LOOK.SSR	SPCTF.SSR	128	30.14

(a) Preseek Command Queueing

Trace Scale		Host Sched (all .CW)	Disk Sched	CQ	Mean Resp Time (ms)
Cello 1.75	Central		SPCTF	128	109.45
	Distrib	SSTF.SSR	SPCTF.SSW.SSR	4	**105.48**
Snake 1.25	Central		SPCTF	128	25.11
	Distrib	SSTF.CR	SPCTF.SSW.SSR	4	**21.62**
Air-Rsv 2.5	Central		SPCTF	128	21.30
	Distrib	SSTF.SSR	SPCTF.SSW.SSR	16	21.35
Sci-TS 2.5	Central		SPCTF	128	13.49
	Distrib	C-LOOK.SSR	SPCTF.SSW.SSR	128	13.45
Order 1.0	Central		SPCTF	128	18.35
	Distrib	C-LOOK.CW.SSR	SPCTF.SSW.SSR	128	18.40
Report 1.0	Central		SPCTF	128	24.51
	Distrib	C-LOOK.CW.SSR	SPCTF.SSW.SSR	128	24.50

(b) Full Command Queueing

Table 7.2: Mean Response Times for "Reasonable" SPCTF Schedulers

full-knowledge scheduler resides at the disk (with full access to the necessary configuration and state information).

7.1.2 Full System Traces

For light workloads, distributed scheduler implementations provide the same performance as disk-based centralized schedulers, since all pending requests typically fit within the on-board command queues. As the workload intensity increases, on-board queues occasionally reach their maximum lengths and the host-based schedulers begin to take an active role in ordering requests.

Figures 7.22 and 7.23 present mean response times and mean non-compute times for the 8-user *SynRGen* traces. The performance impact of the maximum command queue length varies from implementation to implementation, but the lowest mean non-compute times result when the disk-based scheduler uses an SSTF algorithm for scheduling short (i.e., 4 request) command queues. For disks with Preseek command queueing, a host-based C-LOOK scheduler has a marginal performance advantage. When Full command queueing is enabled, it has a marginal disadvantage.

Figures 7.24 and 7.25 show mean response times and application run times for the *Compress* workload. For configurations with Preseek command queueing, the maximum command queue length affects system-level performance by less than 1%. The lowest run times occur with a maximum command queue length of 16 and a disk-based scheduler using one of the cache-sensitive SPTF algorithms.

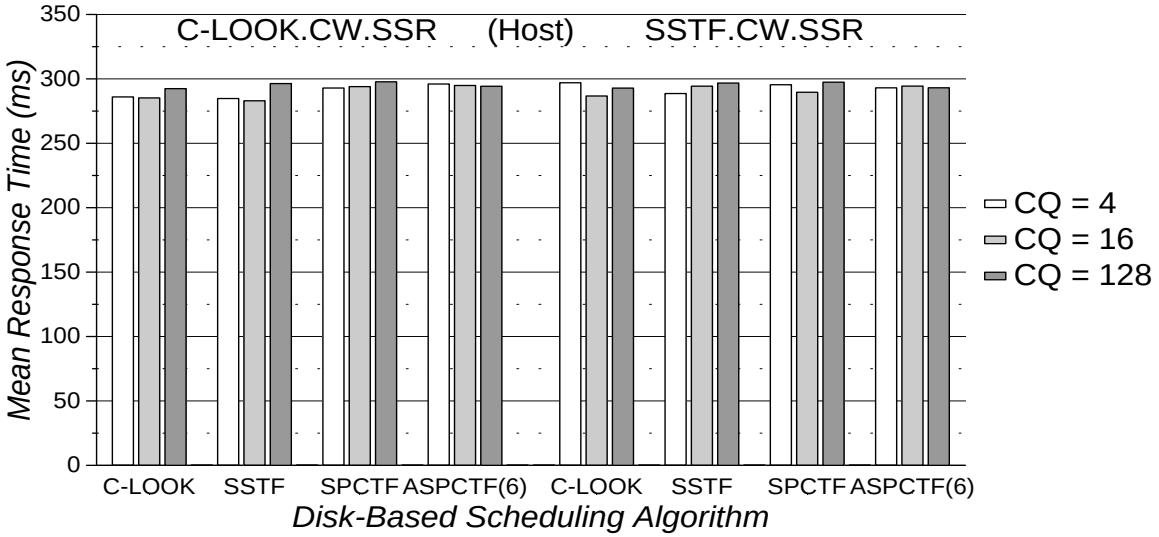
Comparison with Centralized Scheduling

Tables 7.3 and 7.4 allow comparison between the "best" centralized and distributed schedulers. The limited command queue lengths imposed on the disk-based schedulers make host-based centralized scheduling the best choice for scheduler implementations without system-level knowledge. Host-based schedulers produce up to 2% better performance over disk-based schedulers or distributed schedulers for the *SynRGen* and *Compress* workloads.

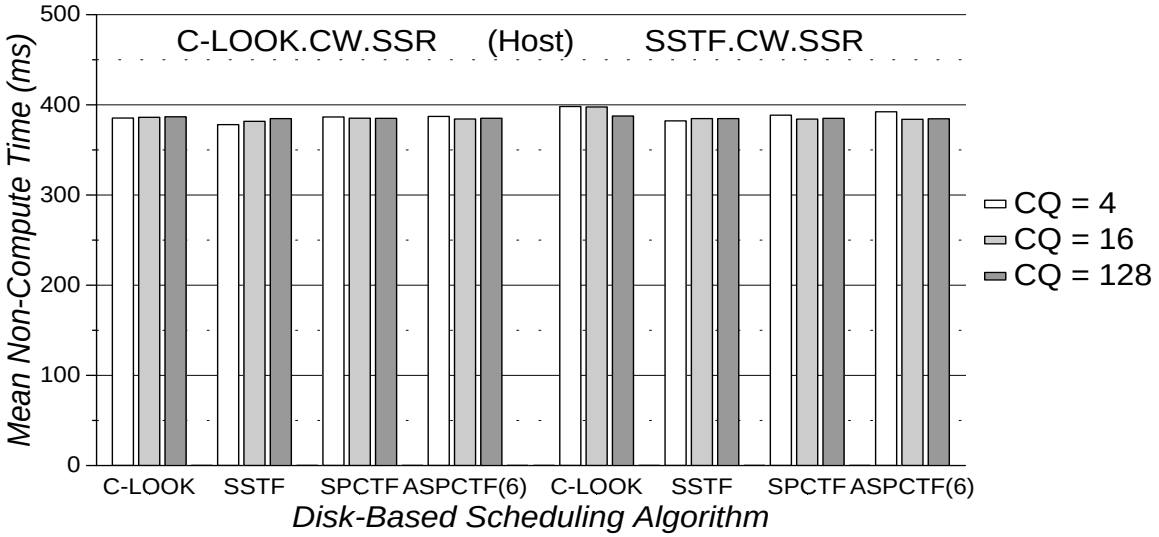
7.1.3 Summary of Conclusions

Distributed scheduler implementations gain the same performance benefits from sequential stream optimizations as their centralized counterparts. For implementations with maximum disk command queue lengths of 16 or higher, mean response times are generally unaffected by the choice between C-LOOK and SSTF host-based scheduling algorithms. The optimal disk-based scheduling algorithm is workload dependent, but distributed SSTF schedulers are more closely matched with distributed C-LOOK schedulers than is the case for centralized scheduling. Full-knowledge distributed scheduler implementations provide the lowest mean response times (when the on-board data caches have prefetching on read hits disabled).

For most configurations, a maximum command queue length of 16 provides the lowest mean response times. This lends additional support to the claim that distributed scheduling provides better overall performance. If this were not the case, most configurations would

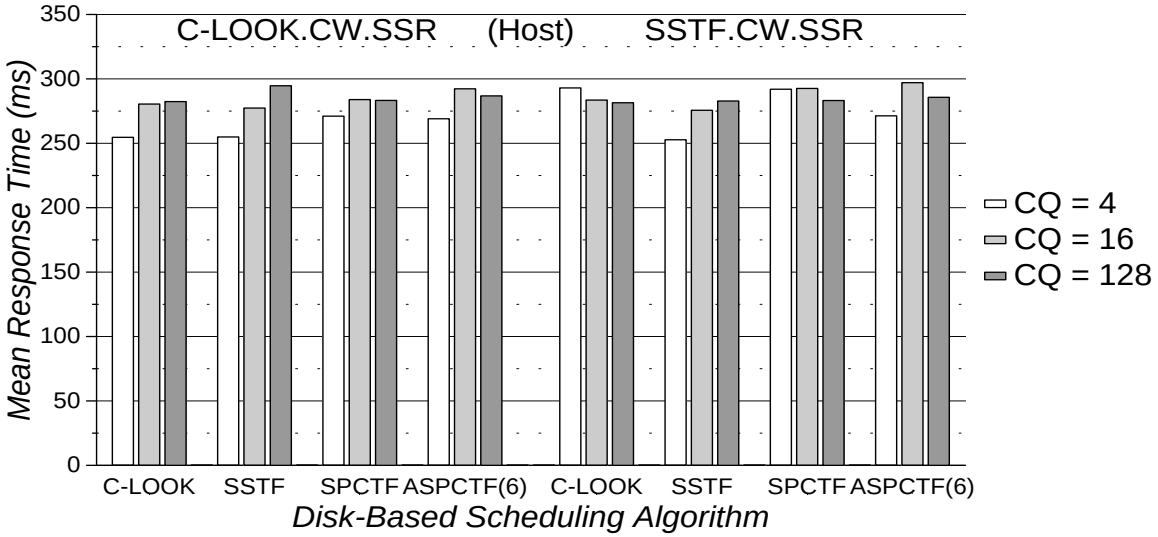


(a) Mean Response Time

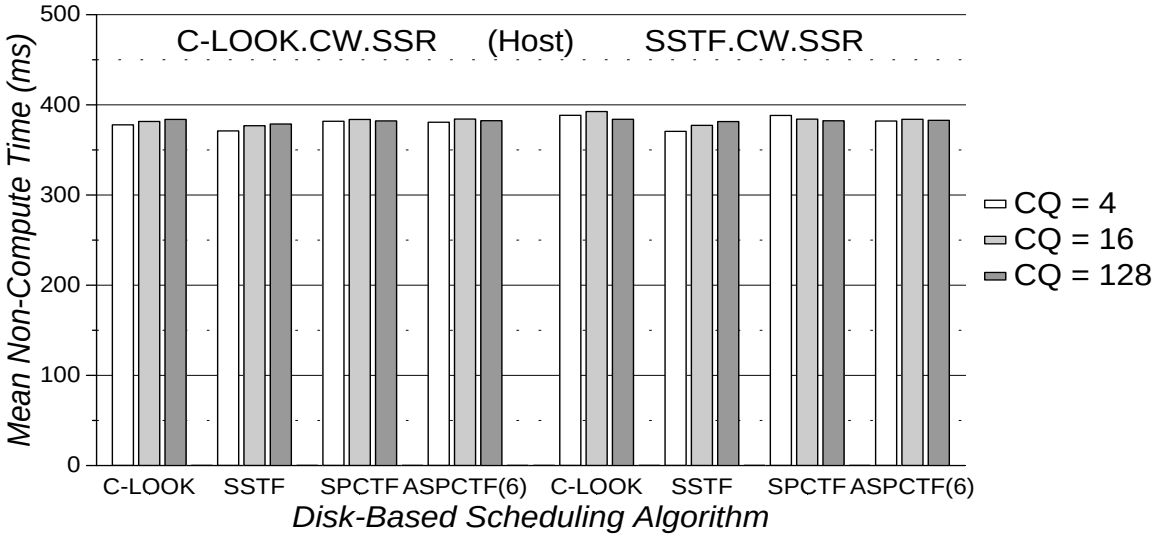


(b) Mean Non-Compute Time

Figure 7.22: *SynRGen*, 8 Users: Distributed Scheduling Algorithm Performance for Disks with Preseek Command Queueing

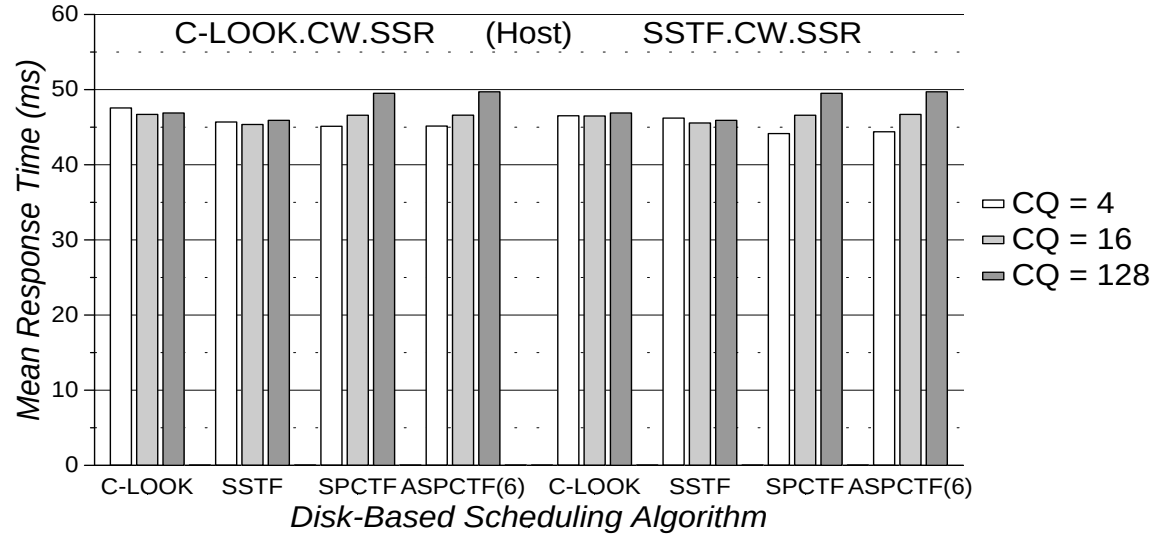


(a) Mean Response Time

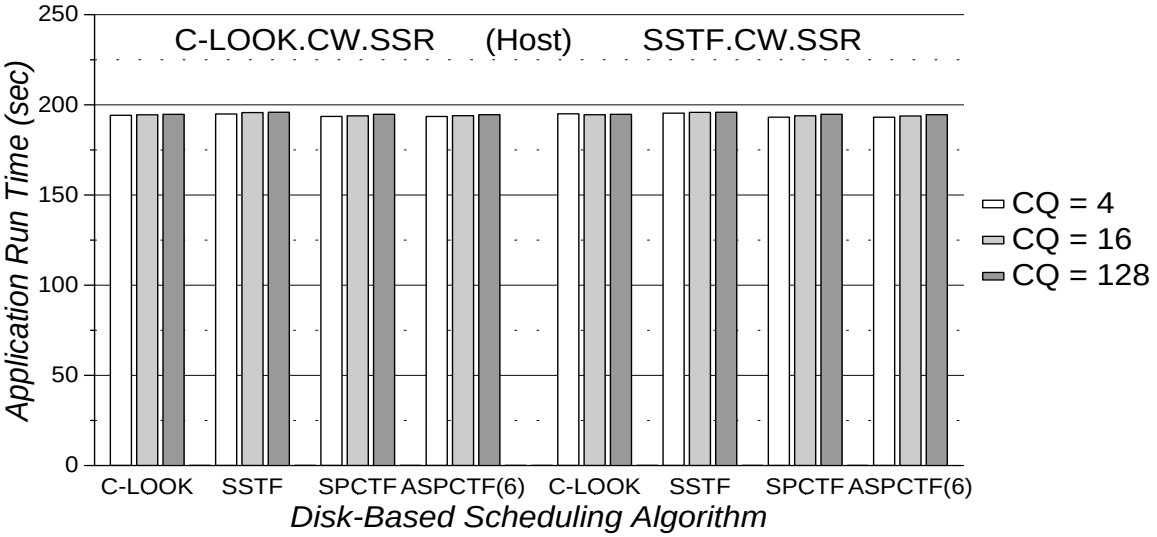


(b) Mean Non-Compute Time

Figure 7.23: *SynRGen*, 8 Users: Distributed Scheduling Algorithm Performance for Disks with Full Command Queueing

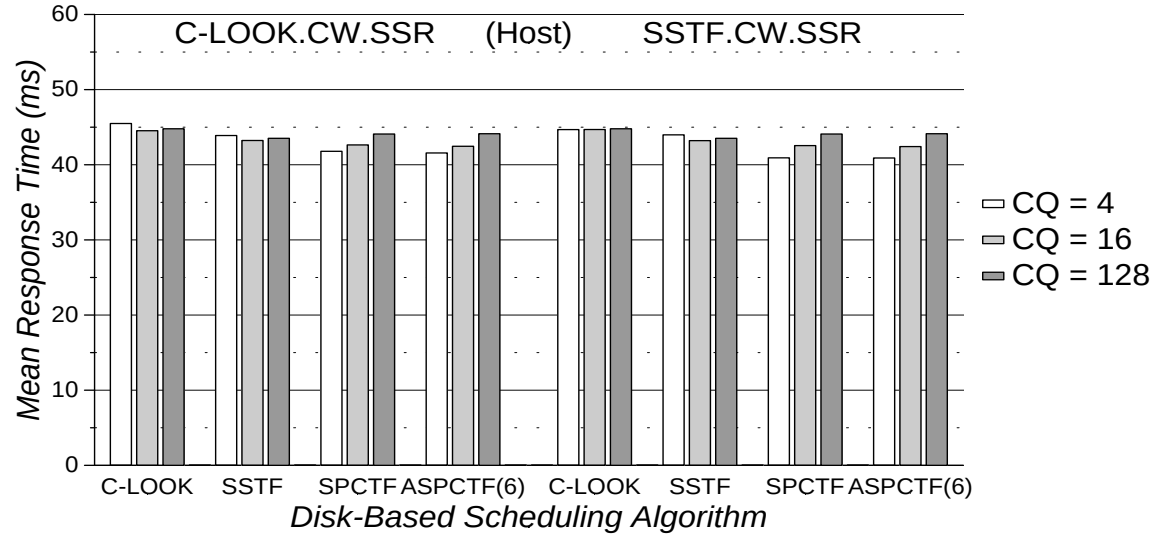


(a) Mean Response Time

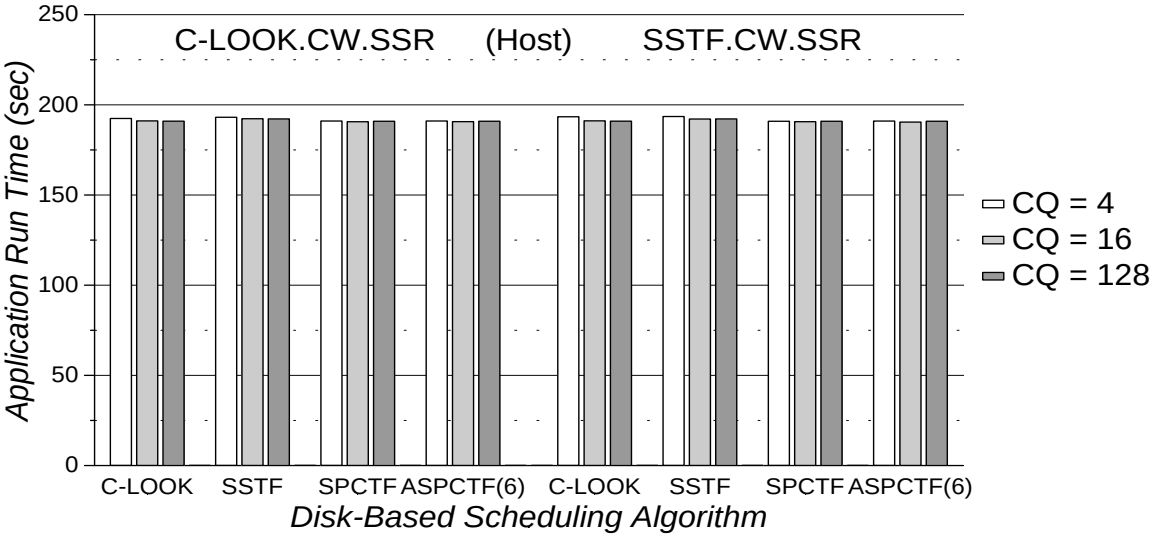


(b) Application Run Time

Figure 7.24: *Compress*: Distributed Scheduling Algorithm Performance for Disks with Preseek Command Queueing



(a) Mean Response Time



(b) Application Run Time

Figure 7.25: *Compress*: Distributed Scheduling Algorithm Performance for Disks with Full Command Queueing

Trace		Host Sched (all .CW.SSR)	Disk Sched	CQ	Metric Value
8-User SynRGen	Central	C-LOOK		4	**374.07**
	Distrib	C-LOOK	SSTF	4	378.06
Compress	Central	C-LOOK		2	**191.82**
	Distrib	C-LOOK	C-LOOK	4	194.21

(a) Preseek Command Queueing

Trace		Host Sched (all .CW.SSR)	Disk Sched (all .SSW.SSR)	CQ	Metric Value
8-User SynRGen	Central	C-LOOK		4	**370.71**
	Distrib	C-LOOK	SSTF	4	371.09
Compress	Central	C-LOOK		2	**190.05**
	Distrib	C-LOOK	C-LOOK	128	190.95

(b) Full Command Queueing

Table 7.3: Mean Non-Compute Times (ms) or Application Run Times (sec) for "Reasonable" LBN-Based Schedulers

Trace		Host Sched (all .CW.SSR)	Disk Sched	CQ	Metric Value
8-User SynRGen	Central	SPCTF		1	**377.74**
	Distrib	SSTF	SPCTF	16	384.20
Compress	Central	SPCTF		1	**191.86**
	Distrib	SSTF	SPCTF	4	193.15

(a) Preseek Command Queueing

Trace		Host Sched (all .CW.SSR)	Disk Sched (all .SSW.SSR)	CQ	Metric Value
8-User SynRGen	Central	SPCTF		1	**377.74**
	Distrib	C-LOOK	SPCTF	4	381.78
Compress	Central		SPCTF	128	190.90
	Distrib	C-LOOK	SPCTF	16	**190.64**

(b) Full Command Queueing

Table 7.4: Mean Non-Compute Times (ms) or Application Run Times (sec) for "Reasonable" SPCTF Schedulers

perform best with very long (i.e., centralized scheduling at the disk) or very short (i.e., centralized scheduling at the host) maximum disk command queue lengths. When the "best" centralized and distributed schedulers are compared, distributed scheduling provides equivalent or superior mean response times for most of the experiments using the HP and DEC disk request traces. The full system traces, however, are better served by centralized schedulers. This is due to the limited command queue length imposed on each disk component of the distributed scheduler implementations.

7.2 Scheduling with Information From "Above" and "Below"

7.2.1 Disk Request Traces

Section 7.1.1 shows that the ASPTF(6) and ASPCTF(6) algorithms have mean response times equivalent (within a few percent) to those of the SPTF and SPCTF algorithms, respectively. Figures 7.26–7.31 show that the age-sensitive SPTF algorithms also provide equivalent or superior starvation resistance (as measured by squared coefficients of variation). Adding age-sensitivity to SPTF algorithms in distributed scheduler implementations is therefore generally advisable. Furthermore, the age-sensitive SPTF algorithms usually provide better starvation resistance than the LBN-based algorithms for all workloads except *Cello*.

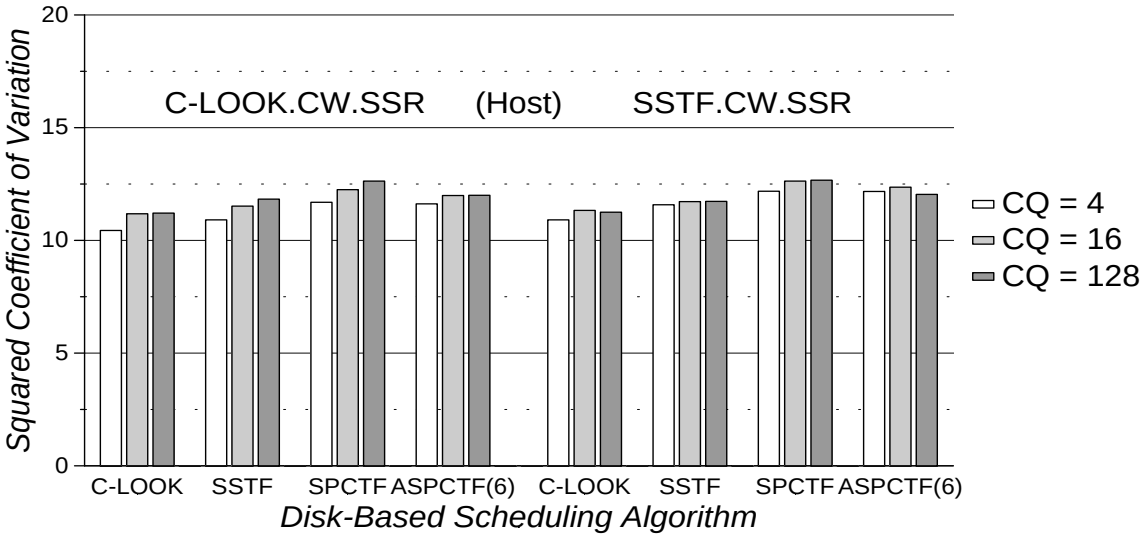
Comparison with Scheduling at the Host

Table 7.5 shows the mean response times for "reasonable" centralized and distributed scheduler implementations using the ASPCTF(6) algorithm. Automatic prefetching on full read hits was disabled for these simulations, as this type of prefetching degrades performance when scheduling the DEC traces. The centralized scheduler listed for each trace configuration represents the host-based or disk-based implementation that produces the lowest mean response time. Similarly, the chosen distributed schedulers provide the lowest mean response times among the distributed implementations studied. For all six traces, the "best" distributed scheduler provides an equivalent or superior mean response time to the "best" centralized scheduler. For the *Snake* trace, distributed scheduling drops mean response times by 32% and 13% for Preseek and Full command queueing configurations, respectively.

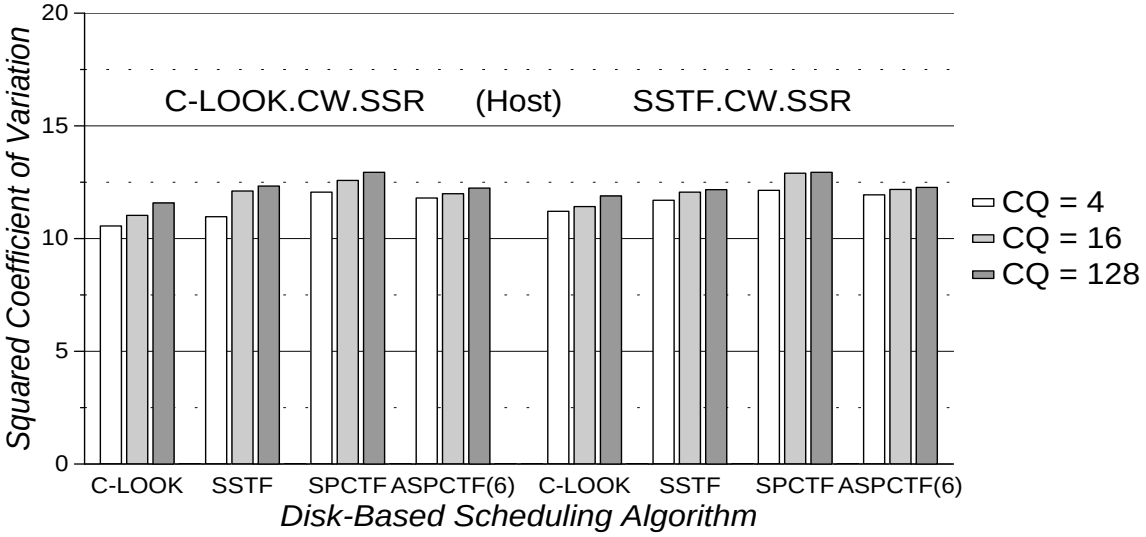
Table 7.6 shows the squared coefficients of variation for the same centralized and distributed scheduler implementations. With one exception, distributed scheduling provides equivalent or superior starvation resistance to centralized scheduling. When scheduling the *Cello* trace for disks with Preseek command queueing, a host-based ASPCTF(6) implementation has a 13% lower squared coefficient of variation.

7.2.2 Full System Traces

This section examines the performance of distributed scheduler implementations using 2Q algorithms. For convenience, the host-based and disk-based centralized schedulers use

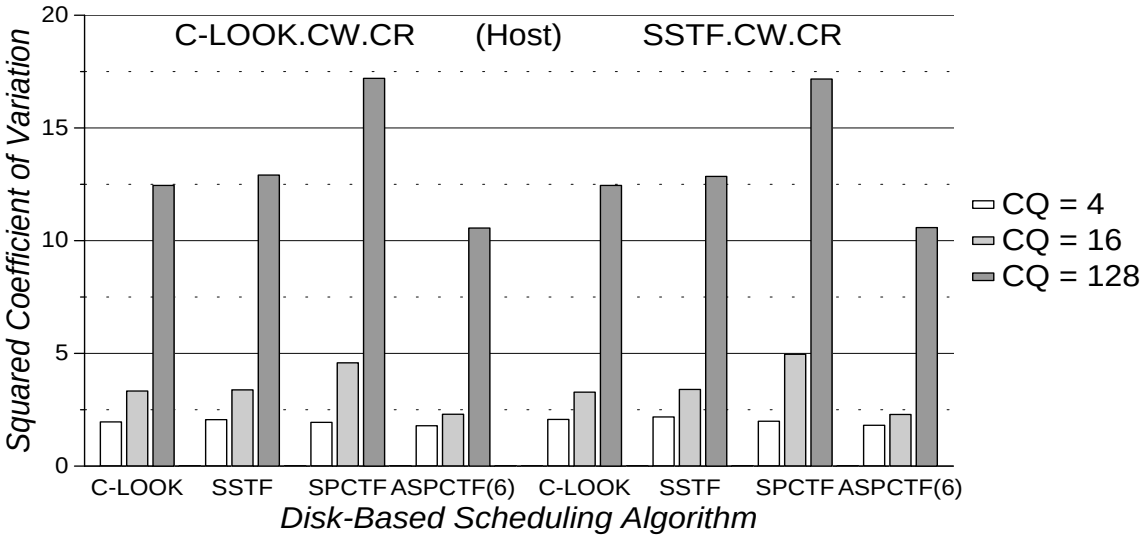


(a) Preseek Command Queueing

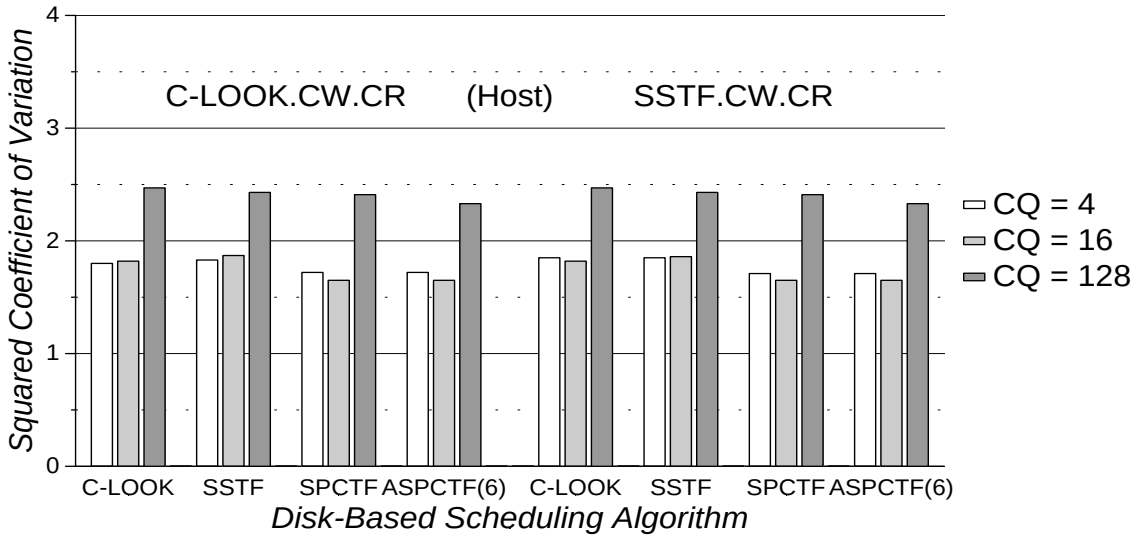


(b) Full Command Queueing

Figure 7.26: *Cello*, 1.75X: Distributed Scheduling Algorithm Performance

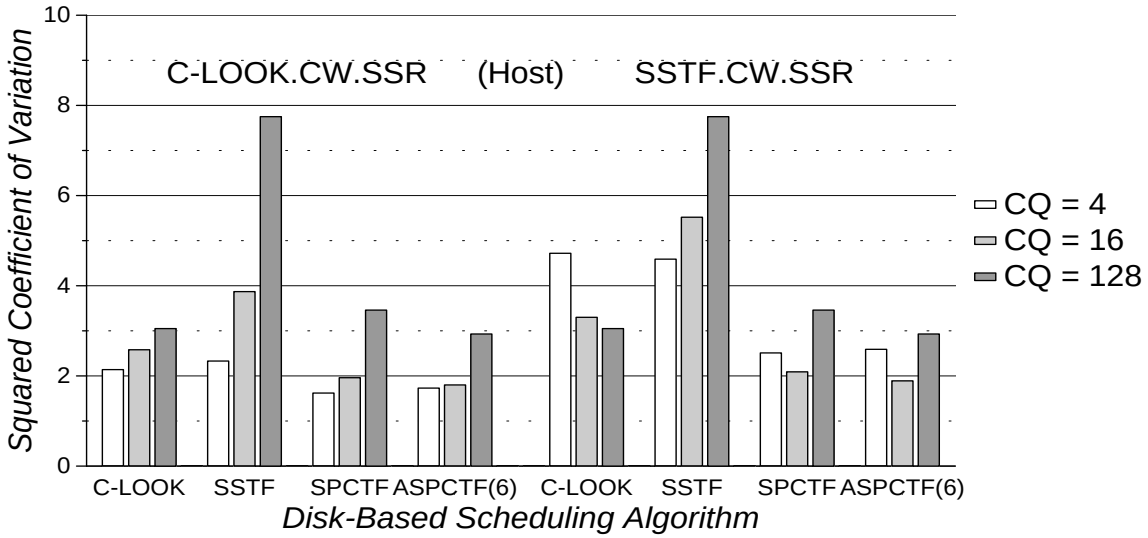


(a) Preseek Command Queueing

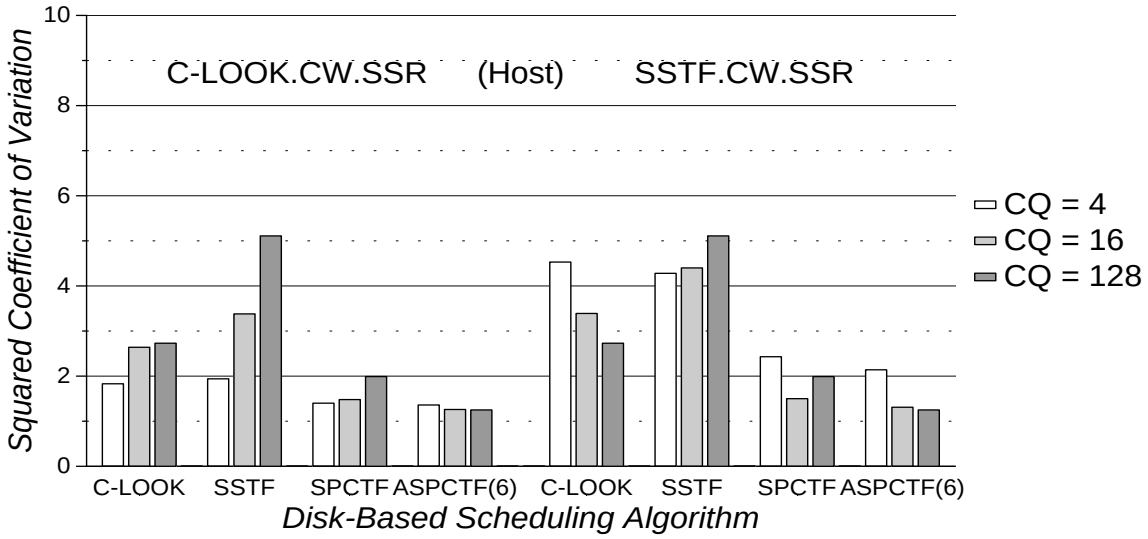


(b) Full Command Queueing

Figure 7.27: *Snake*, 1.25X: Distributed Scheduling Algorithm Performance

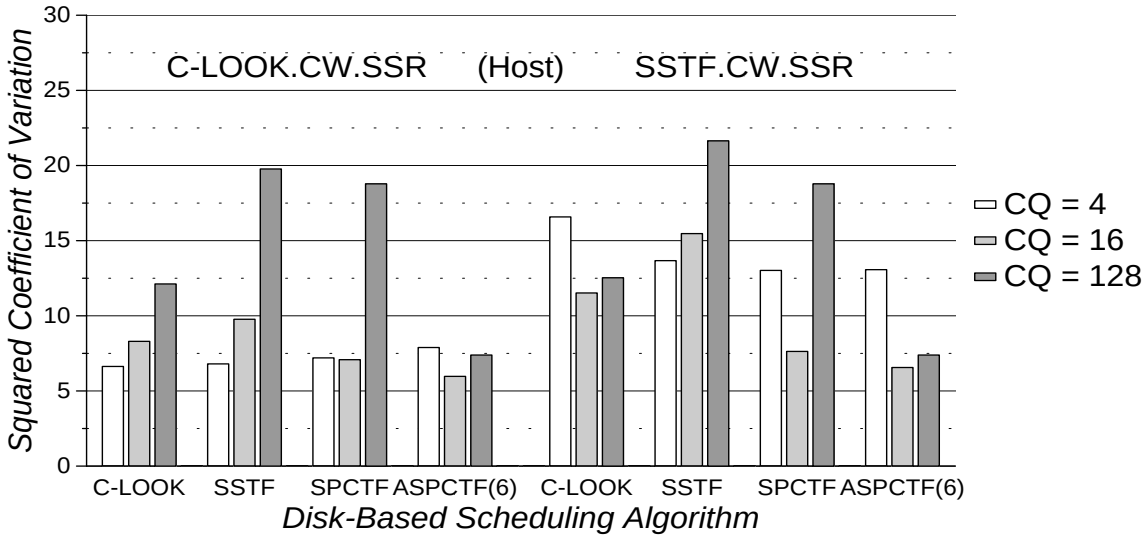


(a) Preseek Command Queueing

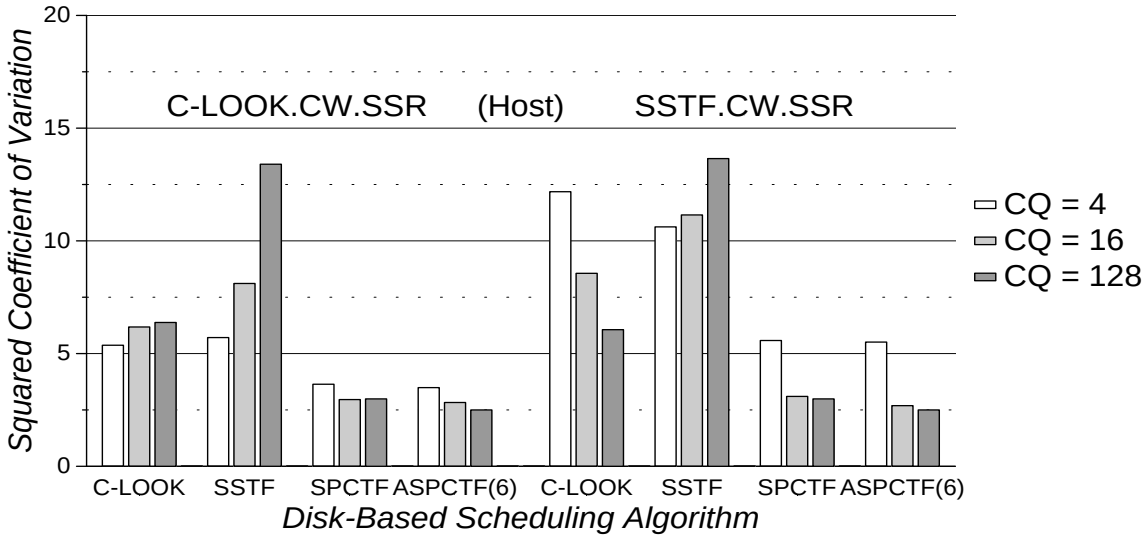


(b) Full Command Queueing

Figure 7.28: *Air-Rsv*, 2.5X: Distributed Scheduling Algorithm Performance

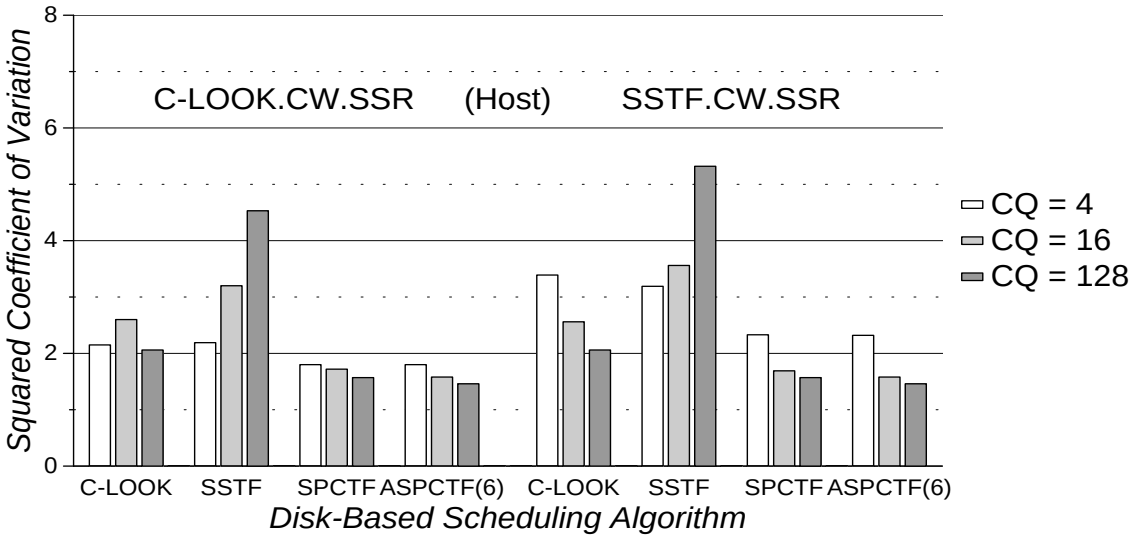


(a) Preseek Command Queueing

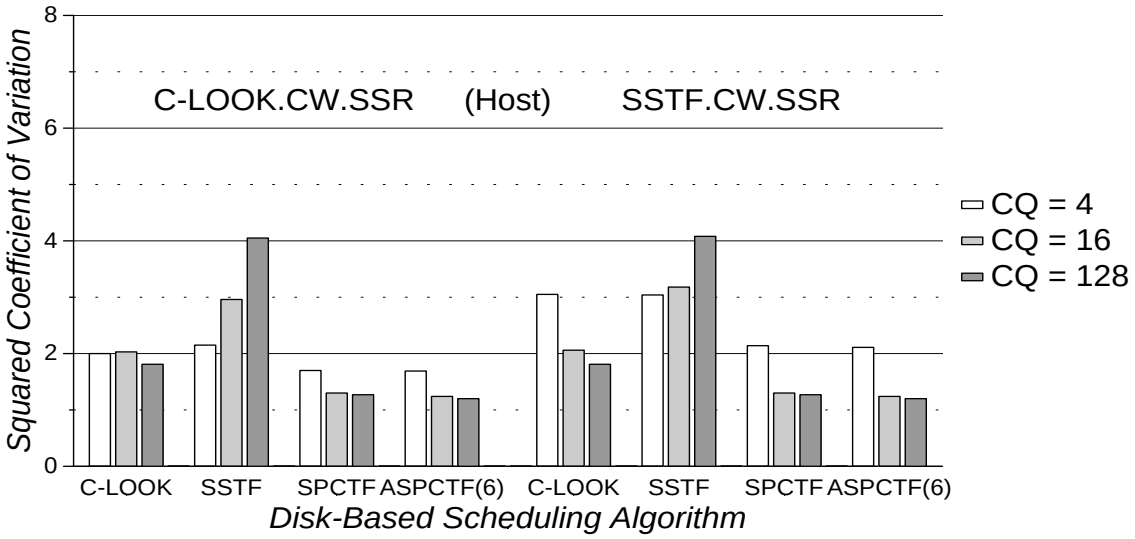


(b) Full Command Queueing

Figure 7.29: *Sci-TS*, 2.5X: Distributed Scheduling Algorithm Performance

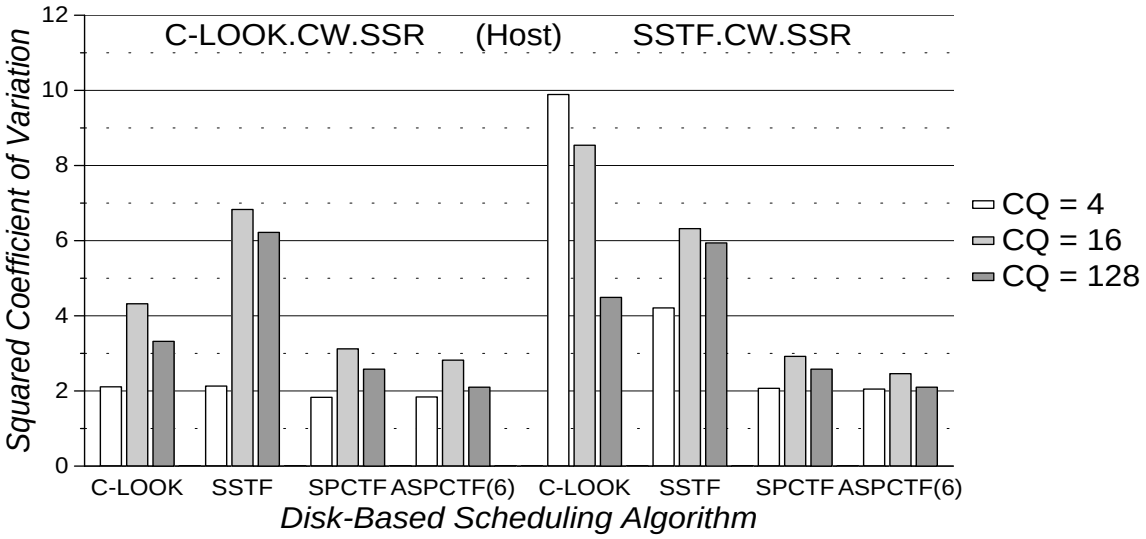


(a) Preseek Command Queueing

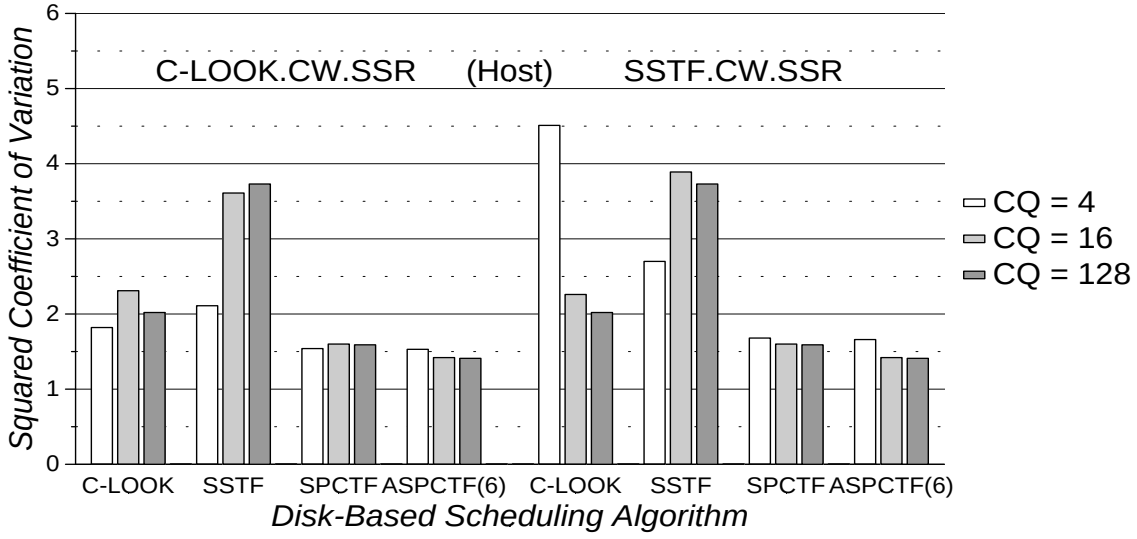


(b) Full Command Queueing

Figure 7.30: *Order*, 1.0X: Distributed Scheduling Algorithm Performance



(a) Preseek Command Queueing



(b) Full Command Queueing

Figure 7.31: *Report*, 1.0X: Distributed Scheduling Algorithm Performance

Trace Scale		Host Sched (all .CW)	Disk Sched (all .SSR)	CQ	Mean Resp Time (ms)
Cello	Central	ASPCTF(6).CR		1	115.38
1.75	Distrib	C-LOOK.SSR	SPCTF.SSR	16	**113.83**
Snake	Central	ASPCTF(6).CR		1	33.92
1.25	Distrib	C-LOOK.CR	SPCTF.SSR	4	**22.95**
Air-Rsv	Central		ASPCTF(6)	128	23.50
2.5	Distrib	SSTF.SSR	SPCTF.SSR	16	**22.85**
Sci-TS	Central		ASPCTF(6)	128	14.64
2.5	Distrib	SSTF.SSR	SPCTF.SSR	16	**13.91**
Order	Central		ASPCTF(6)	128	18.09
1.0	Distrib	SSTF.SSR	SPCTF.SSR	128	18.09
Report	Central		ASPCTF(6)	128	23.96
1.0	Distrib	C-LOOK.SSR	SPCTF.SSR	128	23.96

(a) Preseek Command Queueing

Trace Scale		Host Sched (all .CW)	Disk Sched (all .SSR)	CQ	Mean Resp Time (ms)
Cello	Central		ASPCTF(6)	128	108.66
1.75	Distrib	SSTF.SSR	SPCTF.SSW.SSR	4	**102.73**
Snake	Central		ASPCTF(6)	128	25.04
1.25	Distrib	SSTF.CR	SPCTF.SSW.SSR	4	**21.87**
Air-Rsv	Central		ASPCTF(6)	128	21.13
2.5	Distrib	C-LOOK.SSR	SPCTF.SSW.SSR	128	21.11
Sci-TS	Central		ASPCTF(6)	128	12.79
2.5	Distrib	C-LOOK.SSR	SPCTF.SSW.SSR	128	12.77
Order	Central		ASPCTF(6)	128	17.75
1.0	Distrib	C-LOOK.CW.SSR	SPCTF.SSW.SSR	128	17.75
Report	Central		ASPCTF(6)	128	23.45
1.0	Distrib	C-LOOK.CW.SSR	SPCTF.SSW.SSR	128	23.45

(b) Full Command Queueing

Table 7.5: Mean Response Times for "Reasonable" ASPCTF(6) Schedulers. Prefetching disabled for full read hits.

Trace Scale		Host Sched (all .CW)	Disk Sched (all .SSR)	CQ	Sqr Coeff of Var
Cello	Central	ASPCTF(6).CR		1	**10.38**
1.75	Distrib	C-LOOK.SSR	SPCTF.SSR	16	11.96
Snake	Central	ASPCTF(6).CR		1	14.12
1.25	Distrib	C-LOOK.CR	SPCTF.SSR	4	**1.74**
Air-Rsv	Central		ASPCTF(6)	128	3.69
2.5	Distrib	SSTF.SSR	SPCTF.SSR	16	**1.59**
Sci-TS	Central		ASPCTF(6)	128	6.14
2.5	Distrib	SSTF.SSR	SPCTF.SSR	16	**2.79**
Order	Central		ASPCTF(6)	128	1.12
1.0	Distrib	C-LOOK.SSR	SPCTF.SSR	128	1.12
Report	Central		ASPCTF(6)	128	1.25
1.0	Distrib	C-LOOK.SSR	SPCTF.SSR	128	1.25

(a) Preseek Command Queueing

Trace Scale		Host Sched (all .CW)	Disk Sched (all .SSR)	CQ	Sqr Coeff of Var
Cello	Central		ASPCTF(6)	128	12.71
1.75	Distrib	SSTF.SSR	SPCTF.SSW.SSR	4	**11.89**
Snake	Central		ASPCTF(6)	128	2.30
1.25	Distrib	SSTF.CR	SPCTF.SSW.SSR	4	**1.68**
Air-Rsv	Central		ASPCTF(6)	128	1.22
2.5	Distrib	SSTF.SSR	SPCTF.SSW.SSR	16	1.23
Sci-TS	Central		ASPCTF(6)	128	2.25
2.5	Distrib	C-LOOK.SSR	SPCTF.SSW.SSR	128	2.24
Order	Central		ASPCTF(6)	128	1.18
1.0	Distrib	C-LOOK.CW.SSR	SPCTF.SSW.SSR	128	1.18
Report	Central		ASPCTF(6)	128	1.28
1.0	Distrib	C-LOOK.CW.SSR	SPCTF.SSW.SSR	128	1.28

(b) Full Command Queueing

Table 7.6: Squared Coefficients of Variation for "Reasonable" ASPCTF(6) Schedulers. Prefetching disabled for full read hits.

the same 2Q algorithm, although the sequential stream optimizations differ. The host-based schedulers use concatenation of writes and sequential scheduling of reads. The disk-based schedulers use sequential scheduling of read and write requests when Full command queueing is enabled.

Host-based schedulers are assumed to have knowledge of each disk's maximum command queue length. In order to prevent high priority requests from being delayed by full on-board command queues, host-based schedulers only issue low priority requests to disks with at least 25% of their command queue empty. That is, once a disk's command queue reaches 75% of maximum, only high priority requests will be passed on by the host-based scheduler.[1]

For light workloads, distributed schedulers provide the same performance as disk-based centralized schedulers, since all pending requests typically fit within the on-board queues. Figures 7.32 and 7.33 present mean response times and mean non-compute times for the 8-user *SynRGen* traces. For configurations with Preseek command queueing, command queue lengths of 4 or 16 provide the lowest mean non-compute times. SSTF outperforms FCFS, C-LOOK, and the SPTF-based algorithms by 2%. The inferior performance of SPTF is due to an unfortunate on-board cache design choice (see section 5.1.3). For Full command queueing configurations, a command queue length of 4 provides the lowest mean non-compute times. The SPTF-based algorithms trail SSTF by less than 1% when Full command queueing is enabled.

Figures 7.34 and 7.35 show mean response times and application run times for the *Compress* workload. The maximum command queue length affects application run times by less than 1%. The SPTF-based algorithms outperform FCFS and the LBN-based algorithms by 2–3%.

Comparison with Centralized Scheduling

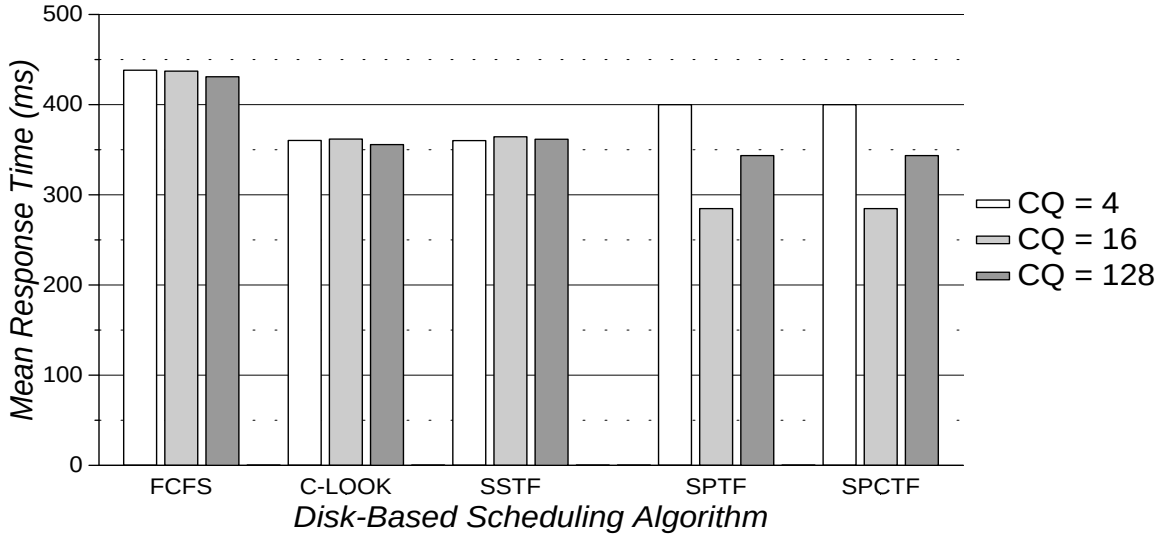
Tables 7.7 and 7.8 allow comparison between the "best" centralized and distributed 2Q schedulers. For configurations with either Preseek or Full command queueing, distributed implementations provide superior values for the application-specific performance metrics. Distributed schedulers using LBN-based algorithms provide 4% and 0.5% better performance for 8-user *SynRGen* and *Compress*, respectively. For schedulers using full-knowledge algorithms, distributed scheduling provides a 0.4–2.7% performance improvement for these workloads.

7.2.3 Summary of Conclusions

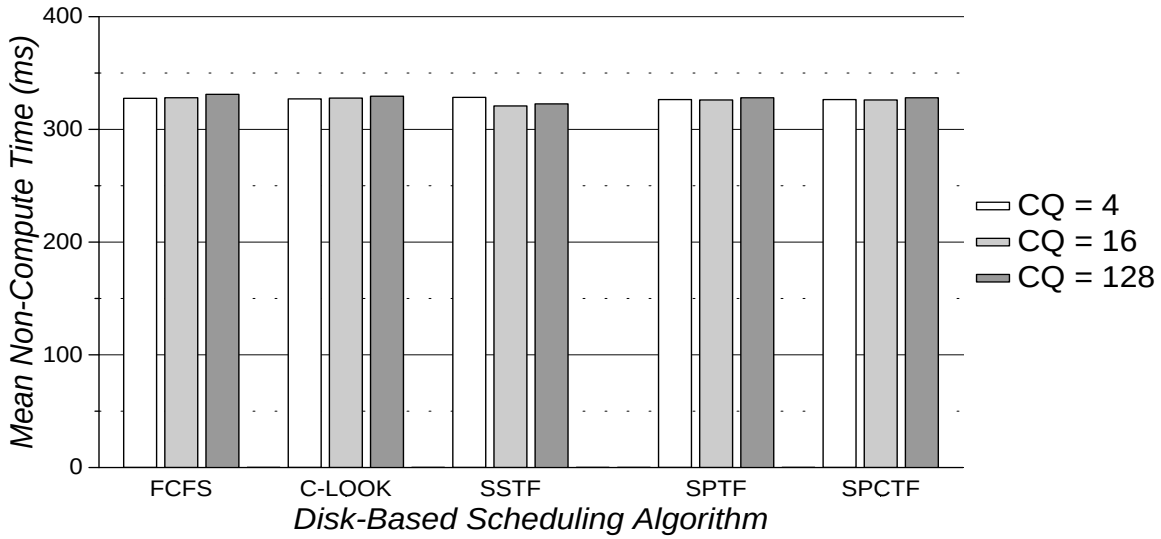
Age-sensitive versions of the full-knowledge algorithms provide good mean response times and superior starvation resistance for distributed scheduling of the disk request traces. When the "best" centralized and distributed ASPCTF(6) implementations are compared, distributed scheduling provides lower mean response times and better starvation resistance.

For light workloads, distributed schedulers provide the same performance as disk-based centralized schedulers, since all pending requests typically fit within the on-board disk queues. For the 8-user *SynRGen* and *Compress* traces, distributed 2Q scheduler implemen-

[1]Since this optimization requires that the host take an active role in selecting requests for service, it was not considered for disk-based centralized scheduling.

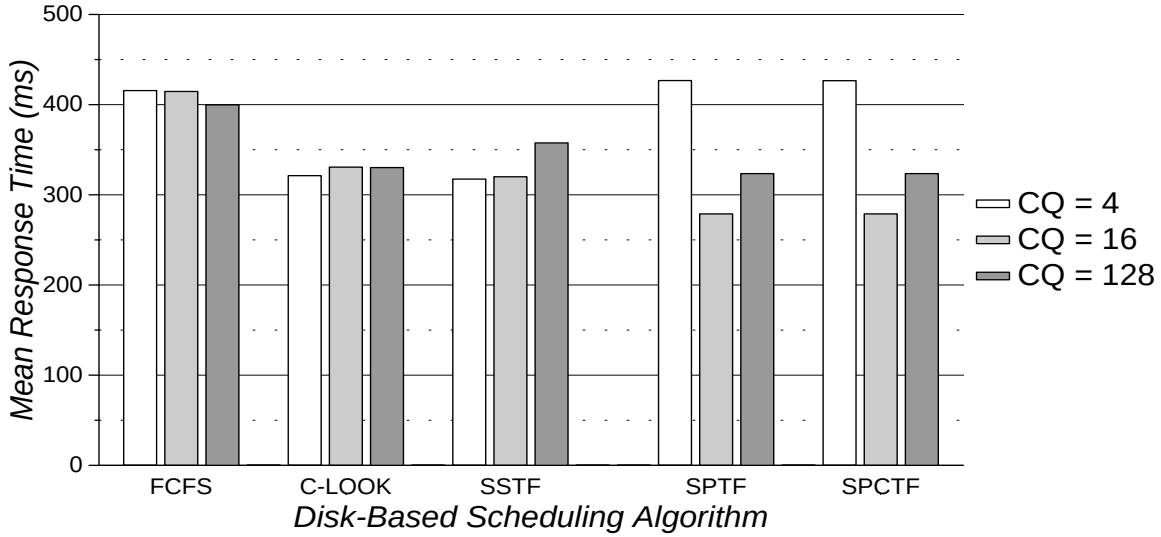


(a) Mean Response Time

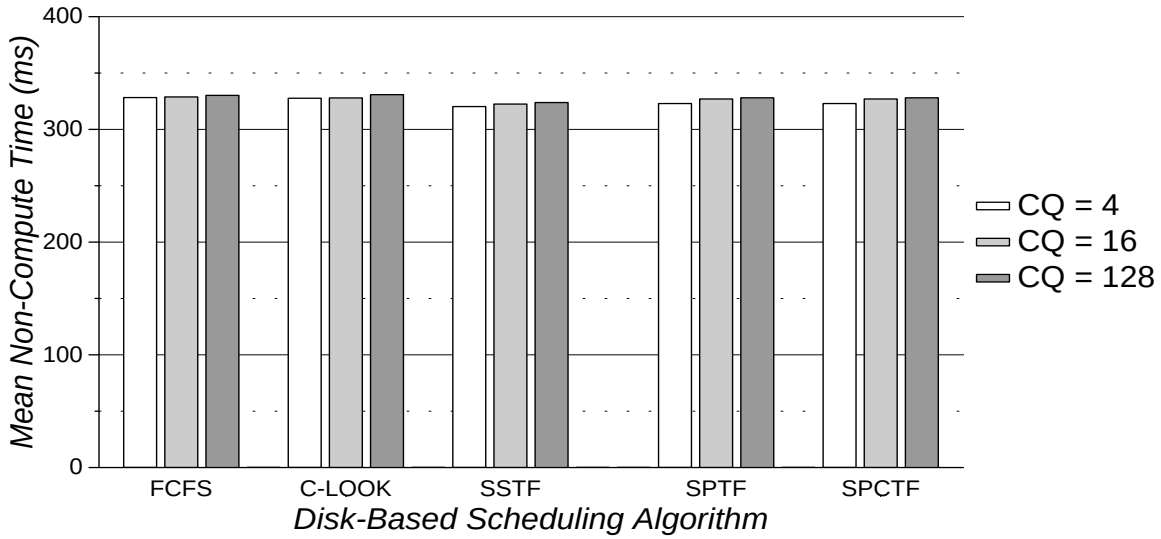


(b) Mean Non-Compute Time

Figure 7.32: *SynRGen*, 8 Users: Distributed 2Q Scheduling Algorithm Performance for Disks with Preseek Command Queueing

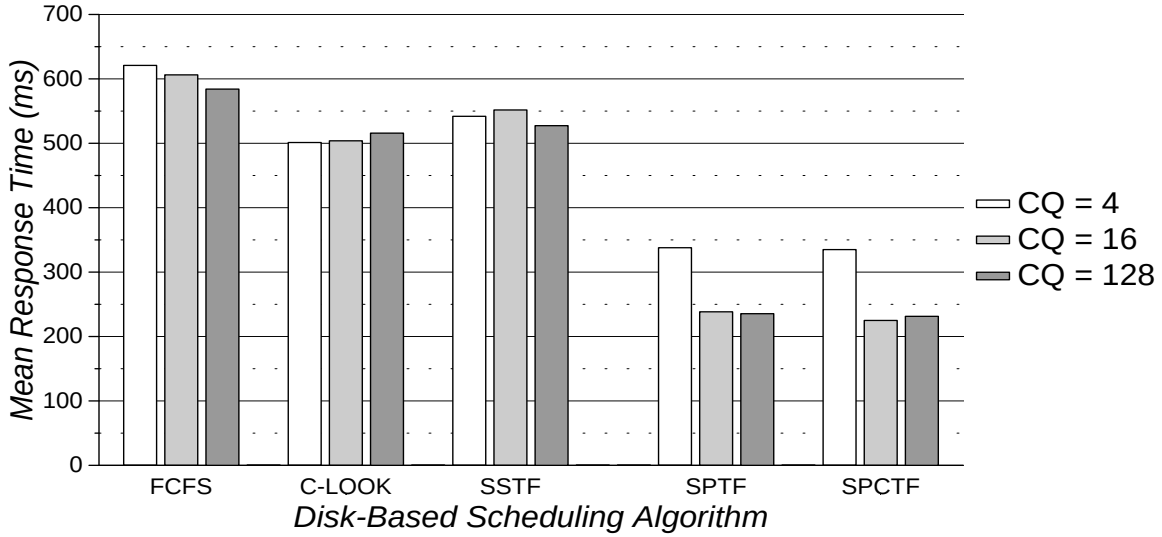


(a) Mean Response Time

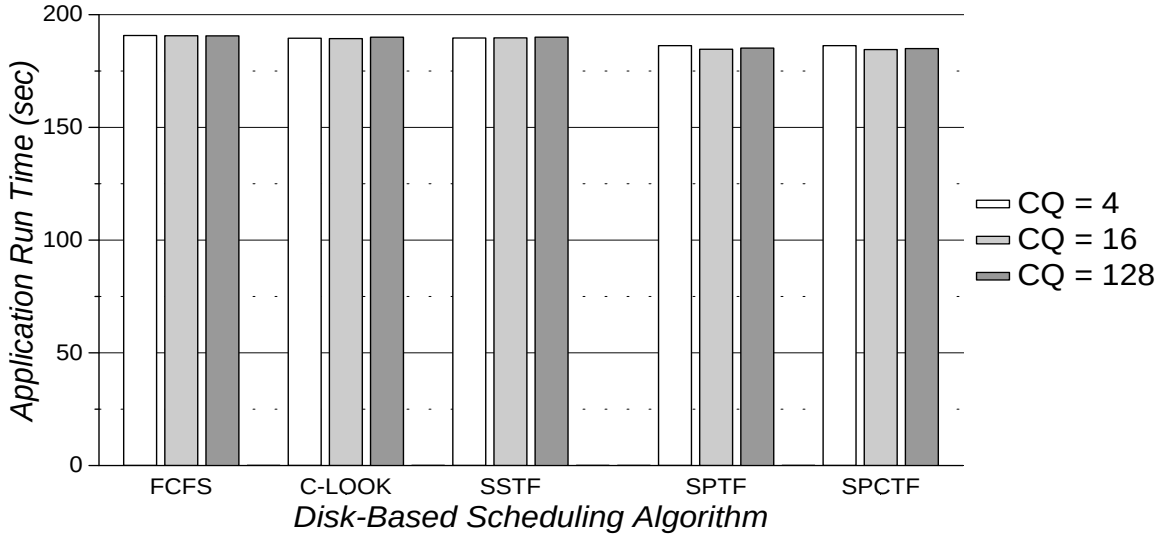


(b) Mean Non-Compute Time

Figure 7.33: *SynRGen*, 8 Users: Distributed 2Q Scheduling Algorithm Performance for Disks with Full Command Queueing

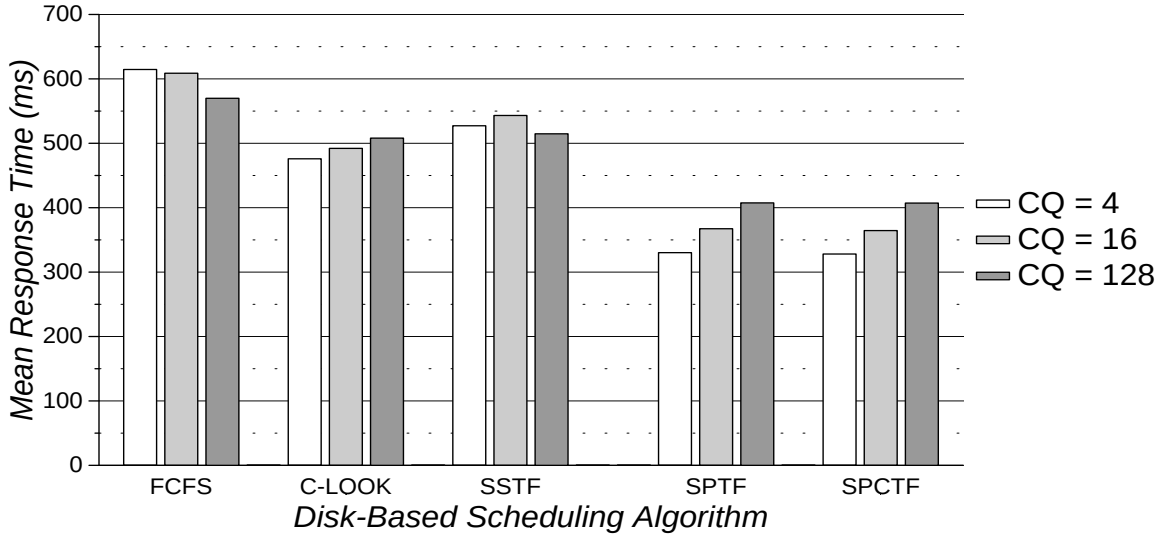


(a) Mean Response Time

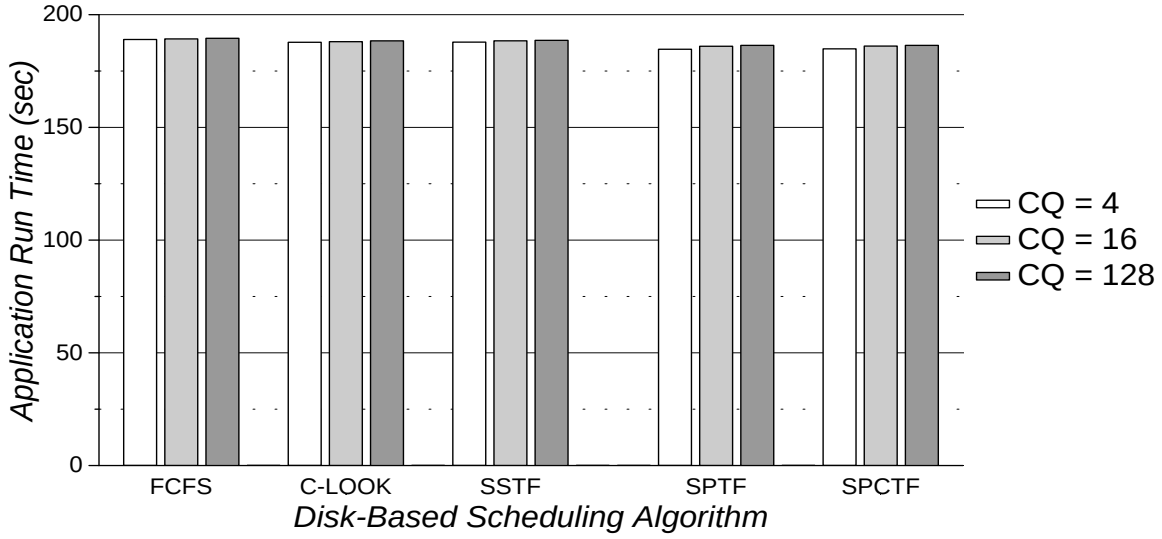


(b) Application Run Time

Figure 7.34: *Compress*: Distributed 2Q Scheduling Algorithm Performance for Disks with Preseek Command Queueing



(a) Mean Response Time



(b) Application Run Time

Figure 7.35: *Compress*: Distributed 2Q Scheduling Algorithm Performance for Disks with Full Command Queueing

Trace		Host Sched (all .CW.SSR)	Disk Sched	CQ	Metric Value
8-User SynRGen	Central	SSTF		1	333.56
	Distrib	SSTF	SSTF	16	**320.80**
Compress	Central		C-LOOK	128	190.38
	Distrib	C-LOOK	C-LOOK	16	**189.38**

(a) Preseek Command Queueing

Trace		Host Sched (all .CW.SSR)	Disk Sched (all .SSW.SSR)	CQ	Metric Value
8-User SynRGen	Central	SSTF		1	333.56
	Distrib	SSTF	SSTF	4	**320.25**
Compress	Central		C-LOOK	128	188.70
	Distrib	C-LOOK	C-LOOK	4	**187.75**

(b) Full Command Queueing

Table 7.7: Mean Non-Compute Times (ms) or Application Run Times (sec) for "Reasonable" LBN-Based 2Q Schedulers

Trace		Host Sched (all .CW.SSR)	Disk Sched	CQ	Metric Value
8-User SynRGen	Central	SPCTF		1	332.05
	Distrib	SPCTF	SPCTF	16	**326.11**
Compress	Central		SPCTF	128	185.31
	Distrib	SPCTF	SPCTF	16	**184.52**

(a) Preseek Command Queueing

Trace		Host Sched (all .CW.SSR)	Disk Sched (all .SSW.SSR)	CQ	Metric Value
8-User SynRGen	Central	SPCTF		1	332.05
	Distrib	SPCTF	SPCTF	4	**322.93**
Compress	Central		SPCTF	128	186.82
	Distrib	SPCTF	SPCTF	4	**184.84**

(b) Full Command Queueing

Table 7.8: Mean Non-Compute Times (ms) or Application Run Times (sec) for "Reasonable" 2Q SPCTF Schedulers

tations provide superior performance over equivalent centralized scheduler implementations (as measured by application-specific metrics).

The experiments in this dissertation show that distributed scheduling provides performance equal to or slightly superior to that of both host-based and disk-based centralized scheduling. With the additional cost and complexity benefits of distributed scheduler implementations, this new scheduling paradigm is recommended for future high-performance computer systems.

CHAPTER 8

Concluding Remarks and Future Directions

This dissertation demonstrates the major impact that disk request scheduling has on storage subsystem performance and overall system performance. Disk workloads contain intense bursts of activity, which result in long queues of pending disk requests. A disk request scheduler uses knowledge of the system and of the relationships between individual requests to produce a request ordering that optimizes specific criteria. As the amount of useful information increases, a scheduler achieves better performance, albeit at the cost of increased scheduling algorithm complexity and resource requirements (e.g., processing power and memory).

Useful scheduling information is partitioned based on whether it comes from "above" or "below" the scheduler. Information from "above" helps the scheduler to identify the importance of individual requests to the application processes and any request ordering restrictions. Experiments in this work use a simple priority scheme that classifies a request as high priority if a host process is (or will be) waiting for it to complete. Even this straightforward approach allows a scheduler to significantly improve overall system performance. For example, user "tasks" spend at least 16% less time waiting on disk I/O in an 8-user *SynRGen* edit/make/debug environment (as indicated by the mean non-compute time, a conservative system performance metric).

Without explicit knowledge of request priorities or the impact of individual requests on performance and/or reliability goals, a scheduler can only attempt to prevent the starvation of critical requests by minimizing response time variance (i.e., reducing starvation in general). Age-sensitive scheduling algorithms effectively reduce response time variance by giving priority to requests that have been pending for excessive periods of time. The two age-sensitive algorithms studied in this dissertation consistently and effectively resist starvation.

Information from "below" the scheduler provides hardware-specific knowledge of individual components along the I/O path. This dissertation focuses on schedulers that use disk drive configuration and state information. Mechanical positioning delays often dominate disk request service times, and scheduling algorithms that reduce these mechanical delays result in superior performance. Algorithms designed to minimize seek times require little disk-specific knowledge, relying on the fact that adjacent logical blocks are usually physically adjacent. Algorithms designed to reduce combined seek and rotational latencies usually outperform seek-reducing algorithms, but they require accurate logical-to-physical data map-

pings, seek curves, head switch times, command and completion overheads, rotation speeds, and actuator positions.

An aggressive scheduler can also exploit information related to the operation and state of the on-board data cache found in a modern disk drive. Cached data blocks are accessed far more quickly than blocks requiring media access. Scheduling algorithms designed to utilize on-board cache information typically produce superior performance. In particular, the C-LOOK scheduling algorithm, which always schedules pending requests in logically ascending order, interacts well with common prefetching mechanisms. Other scheduling algorithms, such as Shortest Seek Time First (SSTF), can be modified to take better advantage of on-board caches, but the simplicity of C-LOOK makes it a good choice for low-cost disk request scheduler implementations. More complex algorithms that exploit the cache while reducing combined seek and rotational latency delays (e.g., SPCTF) provide the highest performance. When scheduling the *Report* trace, for example, such algorithms provide lower mean response times and equivalent starvation resistance (compared to seek-reducing algorithms) for trace scaling factors up to 1.5 or more.

This dissertation compares a number of scheduling algorithms proposed in previous work as well as some new variations. For light workloads, all scheduling algorithms provide roughly equivalent performance. As the workload intensity increases, the performance differences between the algorithms grow. Schedulers using First Come First Serve (FCFS) typically saturate a disk subsystem well before those using algorithms designed to reduce mechanical delays. The C-LOOK scheduling algorithm maintains the lowest mean response time and response time variance among the seek-reducing algorithms studied. Variants of the Shortest Positioning Time First (SPTF) algorithm, which reduces combined seek and rotational latency delays, generally provide the best performance for medium-to-heavy workloads. However, SPTF-based algorithms degrade less gracefully than C-LOOK for the heaviest workloads (i.e., those close to saturation). Cache-sensitive versions of SPTF usually provide the highest performance. Age-sensitive versions provide increased resistance to starvation (i.e., lower response time variance). Combining cache-sensitive SPTF with age-sensitive SPTF produces a robust scheduling algorithm that performs well across a wide range of workloads.

Several scheduling algorithm optimizations exploit the presence of sequential disk requests in real-world workloads. When a host-based scheduler concatenates multiple sequential requests into fewer, larger requests, it reduces the total request processing overhead and better utilizes the target disk's on-board cache. For the *Snake* trace, which contains occasional bursts of highly sequential reads, concatenating read requests at the host decreases mean response times by 60–70% (for all algorithms except FCFS) at a trace scaling factor of 1.25. Alternately, a scheduler can improve performance by explicitly scheduling sequential requests in logically ascending order. This optimization also makes better use of the on-board cache, but it can degrade performance when scheduling sequential streams of write requests for disks without the ability to prebuffer write request data. For disks with write prebuffering, sequential scheduling of writes generally provides a marginal reduction in mean response times.

In a centralized scheduler implementation, only one entity along the I/O path actively reorders pending requests. Other entities participate in disk scheduling only by extracting and passing useful information to the active scheduler. The host and the disk are two

obvious locations for a centralized scheduler, as each possesses a significant quantity of information useful to scheduling activities. Host-based schedulers have the advantage of sequential request concatenation. They also have (essentially) unbounded queues for holding requests, allowing all pending requests to be considered for service. Disk-based schedulers have limited command queue capacity, but they maximize inter-request concurrency at the disk and have better access to disk configuration and state information. Also, disk-based schedulers are better suited to exploit the increasing computation and memory resources included with modern disk drive components.

Host-based schedulers can exploit inter-request concurrency by using FCFS command-queued disks. Short command queue lengths provide the best performance; longer on-board queues reduce a host-based scheduler's flexibility to dynamically reorder requests. On the other hand, experiments driven by the full system traces indicate that disk command queueing may not be advisable for host-based schedulers that use request priority information. If low priority requests are issued to a FCFS command-queued disk, a host-based scheduler cannot change the order of service when subsequent high priority requests arrive.

Although the optimal location for a centralized scheduler is workload-specific, scheduling algorithms using request priority information work best if implemented at the host (unless the workload is very light). Otherwise, when a disk's on-board command queue reaches its maximum length, subsequent requests are not considered for service (until space becomes available in the on-board queue). If high priority requests cannot reach the disk-based scheduler, overall system performance suffers.

In a distributed scheduler implementation, two or more entities along the I/O path cooperatively reorder disk requests. Each component scheduler exploits different information to achieve specific scheduling goals. This dissertation examines distributed scheduler implementations that use host-based schedulers in conjunction with disk-based schedulers. In most cases, a well-designed distributed scheduler outperforms a centralized scheduler with equivalent resources.

In addition to the performance advantage, distributed scheduling provides cost and complexity benefits. Aggressive host-based centralized schedulers require the extraction and communication of a large quantity of disk-specific configuration and state information. Distributed schedulers do not require this additional complexity, since the disk-based scheduling component has easy access to all disk-specific data. Disk-based centralized schedulers can be hampered by limited on-board memory and computation resources. In particular, high priority requests may be delayed at the host if on-board request queues are too small. Distributed scheduling components can cooperate to reduce the impact of finite on-board command queues. This cooperation can actually reduce the amount of on-board resources required for good performance. For example, several of the real-world traces used in this dissertation are best scheduled by distributed implementations using disks with relatively short on-board command queues.

As the demands on disk subsystems increase, disk request scheduling becomes a major factor in determining system performance. An improperly designed or implemented scheduler can easily be the bottleneck in a medium- or heavily-loaded disk subsystem. This dissertation provides guidelines for computer system engineers concerned with building aggressive disk request schedulers in high-performance systems using state-of-the-art storage components.

8.1 Future Work

The work in this dissertation can be extended in several areas:

- The effects of disk drive technology on scheduling activities must be continually re-evaluated. Rotation speeds and media densities are increasing; seek times are dropping; peripheral buses are getting wider and faster; and disk platter diameters are shrinking. As disk technology is expected to improve dramatically in the next decade, studies identifying additional scheduling issues for future high-performance disk subsystems will be needed.

- Full-knowledge scheduling algorithms are usually located at the disk drive, since they require extensive knowledge of disk drive configuration and state. However, [Wort95] describes how the necessary scheduling information can be extracted from modern SCSI disk drives for use in host-based schedulers. In order for a host-based full-knowledge scheduler to effectively utilize FCFS command-queued disks, it must also be able to predict request completion times and peripheral bus utilization. In essence, a host-based SPTF scheduler using FCFS command-queued disks needs to be able to "simulate" disk and bus activity for the immediate future. A scaled-down disk subsystem simulator could provide this functionality. The additional computation and memory resources consumed by the simulation effort should also be considered.

- Experiments in this dissertation show that on-board data cache configurations can significantly impact scheduling performance. In particular, the choice of when to initiate prefetch activity (e.g., after read hits or read misses or both) and the choice of cache segment size (or the number of segments) significantly impact the performance of the scheduler implementations studied. As on-board data caches become more complex, performance demands may necessitate an explicit interface between external request schedulers and on-board cache management algorithms. That is, schedulers and cache managers may need to cooperate in order to achieve high performance. The design and effective use of a scheduler/cache interface is an promising area for future research.

- The 2Q scheduling algorithms studied in this dissertation give priority to time-critical and time-limited requests, thereby achieving better overall system performance. These algorithms could be modified to only give priority to time-limited requests that are in danger of exceeding their time limits. This could improve response times for time-critical requests, albeit at the risk of having more time-limited requests exceed their limits. In addition, this could reduce the starvation of time-noncritical requests (e.g., write requests generated by a cache flush daemon). More sophisticated algorithms could be developed to utilize additional system-level information or make better use of request priorities; the design space for scheduling algorithms that use system-level information remains relatively unexplored.

- The host and the disk have been examined as potential participants in scheduling activities. Intermediate I/O controllers and disk array controllers could also participate in disk request scheduling. Controller-level schedulers could incorporate knowledge of

resource (e.g., disk and bus) utilization and the state of any additional data caches along the I/O path.

- Request scheduling for disk arrays involves new issues, such as reducing combined service times for multi-disk accesses, finding an appropriate priority level for redundant data updates, and selecting between multiple data sources. A number of data layout schemes and redundancy mechanisms have been proposed and implemented. Each presents unique opportunities for aggressive scheduling optimizations.

APPENDIX A

HP C2247 Disk Drive Parameters

The following parameters were used to configure the simulator to model the HP C2247 disk drive. They were obtained both from manufacturer specifications and from direct observation of SCSI bus activity.

Table A.1 contains several fixed and variable parameters used in configuring the HP C2247 disk model. The host system used during the extraction was an NCR 3550 multiprocessor. Some of the values given contain combinations of disk and host delays. For example, the host system (not the HP C2247) limits the data transfer rates given in Table A.1.

Table A.2 specifies the layout of the HP C2247's eight zones. Most of this information is available in the disk drive specifications. All skew values are given in sectors. The Send/Receive Diagnostic SCSI commands were used to obtain full logical-to-physical mappings for eight different disks. This provided data on the first logical sector in each zone and allowed verification of other data layout parameters. The first 17 tracks are reserved for internal use. All spare tracks are placed at the end of each zone. The HP C2247 uses track-based sparing for both slipped and dynamically reallocated defects. The default disk configuration in the experiments has 38 slipped tracks (matching the largest defect list found in the set of eight experimental HP C2247 disk drives).

Table A.3 presents the head switch, seek, and settling delays observed for the HP C2247. A full seek curve is given in figure A.1. These values were obtained via extensive repetition of predetermined seek distances and appropriate monitoring of SCSI activity using a logic analyzer. Seek distances greater than ten cylinders can be approximated by a two-piece curve. Although the exact overlap between command overhead and seek initiation cannot be extracted empirically, separate command overhead and seek values have been determined which together account for the observed behavior.

Table A.4 lists various measured control and communication overheads. In most cases, these values are sensitive to the characteristics of both the current request and the immediately previous request. Also, the cache contents affect the observed delays. In the case of consecutive non-sequential write requests, the second write incurs an additional disconnect (immediately after receiving the command) and has different overhead delays than a write following a read request or a sequential write. Most of the "preparation" delays are caused by the host system rather than the disk drive.

Using the above parameter values for the HP C2247, the model still deviates by a small amount from the observed behavior. By adding additional write overhead delays, this devi-

Constant	Value
Rotational Speed	5400 RPM
Cache Size	256 KB
Read Data Transfer	3.01 MB/s
Write Data Transfer	2.74 MB/s
Read Transfer Disconnect	No

Variable	Value
Cache Segments	2
Read Watermark	75 %
Write Watermark	75 %
Prefetch Minimum	0 KB
Prefetch Maximum	64 KB

Table A.1: Parameters for Modeling the HP C2247 on the NCR 3550 system

Zone	First Cyl	Last Cyl	Sectors /Track	1st Logical Sector	Track Skew	Cyl Skew	Rsrvd Tracks	Spare Tracks
1	0	558	96	0	14	32	17	325
2	559	760	92	14	13	30	0	130
3	761	901	88	73	13	29	0	65
4	902	1051	84	80	12	28	0	78
5	1052	1193	80	52	12	27	0	52
6	1194	1519	72	0	11	24	0	104
7	1520	1793	64	32	10	22	0	78
8	1794	2050	56	30	9	19	0	65

Table A.2: Zone Specifications for the HP C2247

ation can be reduced significantly. Although these delays cannot be attributed to any documented phenomena, the final model closely matches the observed activity of the HP C2247 disk drive. Table A.5 lists the additional overhead delays added to the model. In the last two table entries, the delay depends on the request size (S sectors).

Figures A.2–A.5 compare the response time distributions for four typical validation runs comparing the model and the actual system. The workloads consist of 95% random reads, 95% random writes, 95% sequential reads and 95% sequential writes, respectively. The demerit figures (defined by [Ruem94] as the root mean square horizontal distance between the two distribution curves) are 0.43%, 0.52%, 1.58%, and 0.99% of the mean response times for each run, respectively. The largest demerit figure results from the workload of 95% sequential reads, which are usually satisfied by the prefetching cache and thus exhibit very low mean response times. (The root mean square difference between the two distributions is only 0.078 ms, the lowest value of all four validation runs.) The demerit figure has never exceeded 1.9% of the corresponding mean response time for any validation run.

Seek Distance (cylinders)	Delay (ms)
0 (Head Switch)	0.89
1	2.69
2	3.48
3	3.61
4	3.78
5	4.07
6	4.45
7	4.32
8	4.45
9	4.62
10	4.79
$10 < D \leq 300$	$3.81 + 0.33\sqrt{D}$
$300 < D$	$7.75 + 0.0059D$
Write Settling	0.65

Table A.3: Seek Parameters for Modeling the HP C2247

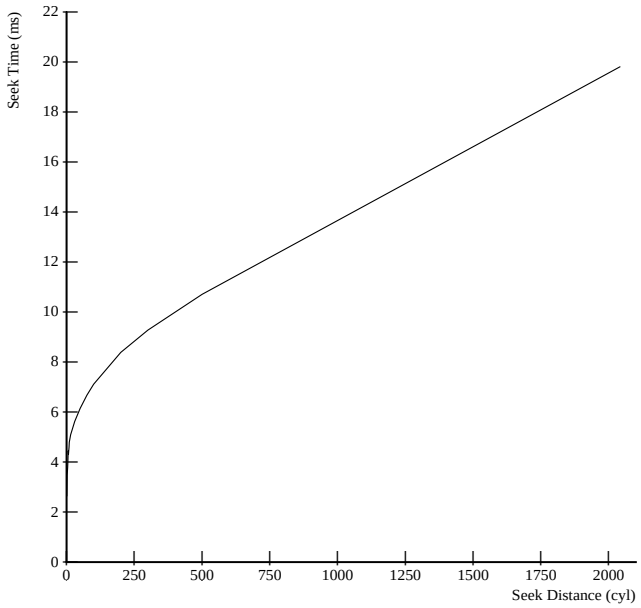


Figure A.1: HP C2247 Disk Drive Seek Curve

Overhead Description	Delay (ms)
Read Command (Hit)	0.953 ms
Read Command (Miss)	0.558 ms
Write Command (Seq or After Read)	0.824 ms
Write Command (Non-Seq Write)	0.642 ms
Read Disconnect Preparation (After Read)	0.023 ms
Read Disconnect Preparation (After Write)	0.046 ms
Write Disconnect Preparation (Seq or After Read)	0.166 ms
Write Disconnect Preparation (Non-Seq Write)	0.046 ms
Write Disconnect Duration (Non-Seq Write)	0.429 ms
Data Phase Preparation	0.025 ms
Read Completion	0.057 ms
Write Completion	0.050 ms
First Reselect	0.162 ms
Second Reselect (Non-Seq Writes Only)	0.129 ms

Table A.4: Overhead Delays for Modeling the HP C2247 on the NCR 3550 system

Overhead Description	Value
Reconnect Preparation After Media Write Complete	0.540 ms
Minimum Time Before Media Write (After Command)	0.600 ms
Minimum Sectors Received Before Media Write ($S < 45$)	Min(10,S)
Minimum Sectors Received Before Media Write ($S \geq 45$)	15

Table A.5: Additional Write Overhead Delays for Modeling the HP C2247 (S is the number of sectors in the request)

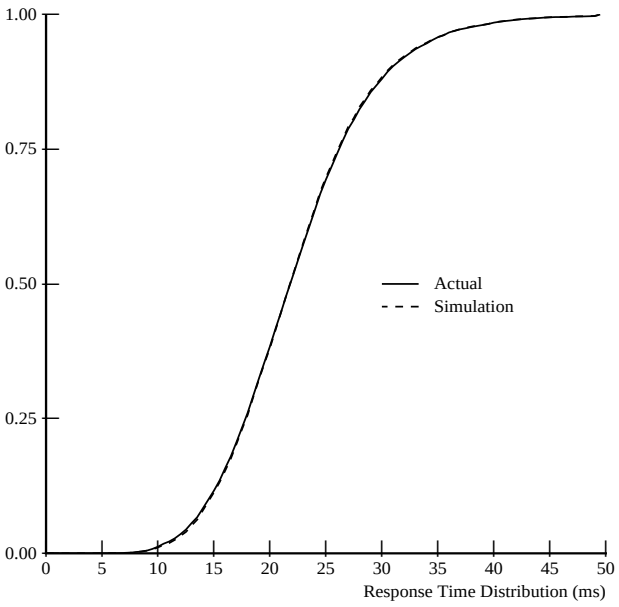


Figure A.2: Validation Workload Response Time Distributions (95% non-sequential reads, 8KB mean request size [exponential], 0-22 ms request interarrival time [uniform])

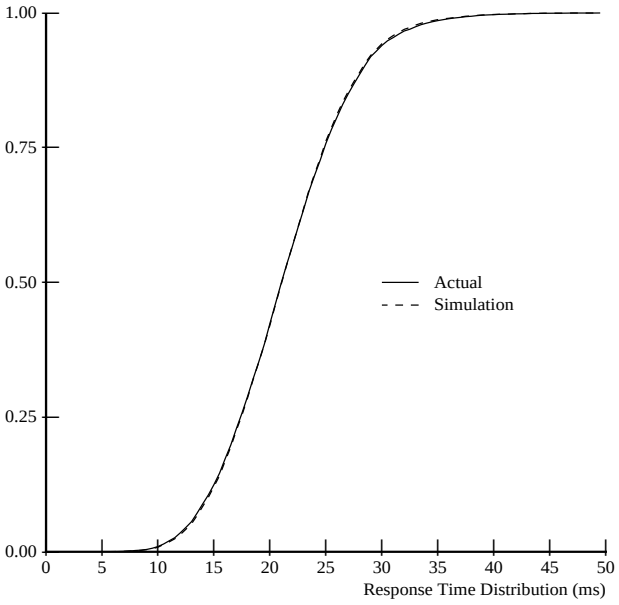


Figure A.3: Validation Workload Response Time Distributions (95% non-sequential writes, 8KB mean request size [exponential], 0-22 ms request interarrival time [uniform])

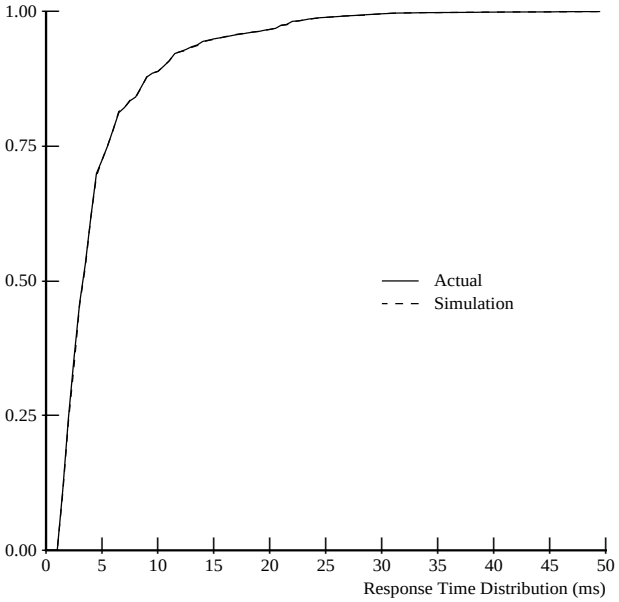


Figure A.4: Validation Workload Response Time Distributions (95% sequential reads, 8KB mean request size [exponential], 0-22 ms request interarrival time [uniform])

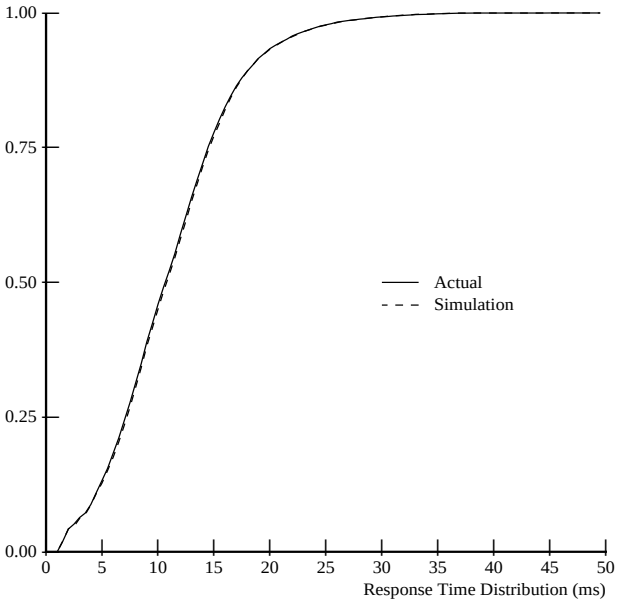


Figure A.5: Validation Workload Response Time Distributions (95% sequential writes, 8KB mean request size [exponential], 0-22 ms request interarrival time [uniform])

BIBLIOGRAPHY

[Abbo89] R. Abbott, H. Garcia-Molina, "Scheduling Real-Time Transactions with Disk Resident Data", *Proceedings of the 15th International Conference on Very Large Data Bases*, Amsterdam, August 1989, pp. 385-396.

[Abbo90] R. Abbott, H. Garcia-Molina, "Scheduling I/O Requests with Deadlines: a Performance Evaluation", *Proceedings of the IEEE Real-Time Systems Symposium*, Lake Buena Vista, Florida, December 1990, pp. 113-124.

[Bate91] K. Bates, *VAX I/O Subsystems: Optimizing Performance*, Professional Press Books, 1991.

[Benn94] S. Bennett, D. Melski, "A Class-Based Disk Scheduling Algorithm: Implementation and Performance Study", Class Project, University of Wisconsin.

[Bitt88] D. Bitton, J. Gray, "Disk Shadowing", *Proceedings of the 14th International Conference on Very Large Data Bases*, Long Beach, California, September 1988, pp. 331-338.

[Bitt89] D. Bitton, "Arm Scheduling in Shadowed Disks", *COMPCON*, March 1989, pp. 132-136.

[Care89] M. Carey, R. Jauhari, M. Livny, "Priority in DMBS Resource Scheduling", *Proceedings of the 15th International Conference on Very Large Data Bases*, Amsterdam, August 1989, pp. 397-410.

[Chen90] P. Chen, G. Gibson, R. Katz, D. Patterson, "An Evaluation of Redundant Arrays of Disks using an Amdahl 5890", *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Boulder, Colorado, May 1990, pp. 74-85.

[Chen91] S. Chen, J. Stankovic, J. Kurose, D. Towsley, "Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems", *Real-Time Systems Journal*, Vol. 3, No. 3, September 1991, pp. 307-336.

[Chen94] S. Chen, D. Towsley, "Scheduling Customers in a Non-Removal Real-Time System with an Application to Disk Scheduling", *Real-Time Systems Journal*, Vol. 6, No. 1, January 1994, pp. 55-72.

[Coff72] E. Coffman, L. Klimko, B. Ryan, "Analysis of Scanning Policies for Reducing Disk Seek Times", *SIAM Journal of Computing*, Vol. 1, No. 3, September 1972, pp. 269-279.

[Dani83] S. Daniel, R. Geist, "V-SCAN: An Adaptive Disk Scheduling Algorithm", *Proceedings of the IEEE International Workshop On Computer Systems Organization*, New Orleans, Louisiana, March 1983, pp. 96-103.

[Denn67] P. Denning, "Effects of Scheduling on File Memory Operations", *Proceedings of the AFIPS Spring Joint Computer Conference*, Atlantic City, New Jersey, April 1967, pp. 9-21.

[Ebli94] M. Ebling, M. Satyanarayanan, "SynRGen: An Extensible File Reference Generator", *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Nashville, Tennessee, May 1994, pp. 108-117.

[Fran69] H. Frank, "Analysis and Optimization of Disk Storage Devices for Time-Sharing Systems", *Journal of the Association for Computing Machinery*, Vol. 16, No. 4, October 1969, pp. 602-620.

[Fuji91] Fujitsu Limited, "M2622Sx/M2623Sx/M2624Sx Intelligent Disk Drives CE Manual", Specification Number 41FH6868E-01, July 1991.

[Fuji91a] Fujitsu Limited, "M2622Sx/M2623Sx/M2624Sx Intelligent Disk Drives OEM Manual - SCSI Logical Specification", Specification Number 41FH5057E-01, January 1991.

[Fuji91b] Fujitsu Limited, "M2622Sx/M2623Sx/M2624Sx Intelligent Disk Drives OEM Manual - Specifications & Installation", Specification Number 41FH5055E-01A, June 1991.

[Gang93a] G. Ganger, B. Worthington, R. Hou, Y. Patt, "Disk Subsystem Load Balancing: Disk Striping vs. Conventional Data Placement", *Proceedings of the 26th Hawaii International Conference on System Sciences*, Wailea, Hawaii, January 1993, Vol. 1, pp. 40-49.

[Gang93] G. Ganger, Y. Patt, "The Process-Flow Model: Examining I/O Performance from the System's Point of View", *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Santa Clara, California, May 1993, pp. 86-97.

[Gang94] G. Ganger, B. Worthington, R. Hou, Y. Patt, "Disk Arrays: High-Performance, High-Reliability Storage Subsystems", *IEEE Computer*, Vol. 27, No. 3, March 1994, pp. 30-36.

[Geis87] R. Geist, S. Daniel, "A Continuum of Disk Scheduling Algorithms", *ACM Transactions on Computer Systems*, Vol. 5, No. 1, February 1987, pp. 77-92.

[Geis87a] R. Geist, R. Reynolds, E. Pittard, "Disk Scheduling in System V", *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Banff, Alberta, May 1987, pp. 59-68.

[Geis95] R. Geist, J. Westall, "Disk Scheduling in Linux", *Proceedings of the Computer Measurement Group (CMG) Conference*, Orlando, Florida, December 1994, pp. 739-746.

[Gotl73] C. Gotlieb, G. MacEwen, "Performance of Movable-Head Disk Storage Devices", *Journal of the Association for Computing Machinery*, Vol. 20, No. 4, October 1973, pp. 604-623.

[Gray90] J. Gray, B. Horst, M. Walker, "Parity Striping of Disk Arrays: Low-Cost Reliable Storage with Acceptable Throughput", *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Australia, August 1990, pp. 148-161.

[Henl89] M. Henley, B. McNutt, "DASD I/O Characteristics: A Comparison of MVS to VM", *Proceedings of the Computer Measurement Group (CMG) Conference*, Reno, Nevada, December 1989, pp. 566-578.

[Holl92] M. Holland, G. Gibson, "Parity Declustering for Continuous Operation in Redundant Disk Arrays", *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, Massachusetts, October 1992, pp. 23-35.

[Hou93] R. Hou, Y. Patt, "Trading Disk Capacity for Performance", *Proceedings of the 2nd International Symposium on High-Performance Distributed Computing*, Spokane, Washington, July 1993, pp. 263-270.

[HP92] Hewlett-Packard Company, "HP C2240 Series 3.5-inch SCSI-2 Disk Drive, Technical Reference Manual", Part Number 5960-8346, Edition 2, April 1992.

[HP92a] Hewlett-Packard Company, "HP C2244/45/46/47 3.5-inch SCSI-2 Disk Drive, Technical Reference Manual", Part Number 5960-8346, Edition 3, September 1992.

[HP93] Hewlett-Packard Company, "HP C2490A 3.5-inch SCSI-2 Disk Drives, Technical Reference Manual", Part Number 5961-4359, Edition 3, September 1993.

[HP94] Hewlett-Packard Company, "HP C3323A 3.5-inch SCSI-2 Disk Drives, Technical Reference Manual", Part Number 5962-6452, Edition 2, April 1994.

[Jaco91] D. Jacobson, J. Wilkes, "Disk Scheduling Algorithms Based on Rotational Position", Hewlett-Packard Technical Report, HPL-CSP-91-7, February 1991.

[Katz89] R.H. Katz, G.A. Gibson, D.A. Patterson, "Disk System Architectures for High Performance Computing", *Proceedings of the IEEE*, Vol. 77, No. 12, December 1989, pp. 1842-1858.

[Kim86] M. Kim, "Synchronized Disk Interleaving", *IEEE Transactions on Computers*, Vol. C-35, No. 11, November 1986, pp. 978-988.

[Kim91] W. Kim, J. Srivastava, "Enhancing Real-Time DBMS Performance with Multiversion Data and Priority Based Disk Scheduling", *Proceedings of the IEEE Real-Time Systems Symposium*, San Antonio, Texas, December 1991, pp. 222-231.

[Lary95] R. Lary, Storage Architect, Digital Equipment Corporation, Personal Communication, May 1995.

[Lee90] E. Lee, "Software and Performance Issues in the Implementation of a RAID Prototype", Report No. UCB/CSD 90/573, University of California, Berkeley, May 1990.

[Lync72] W. Lynch, "Do Disk Arms Move?", *Performance Evaluation Review*, Vol. 1, No. 4, December 1972, pp. 3-16.

[Maxt92] Maxtor Corporation, "MXT-1240S Product Specification and OEM Technical Manual", Document 1028044, Revision A, November 1992.

[McNu86] B. McNutt, "An Empirical Study of Variations in DASD Volume Activity", *Proceedings of the Computer Measurement Group (CMG) Conference*, 1986, pp. 274-283.

[Mert70] A. Merten, "Some Quantitative Techniques for File Organization", Ph.D. Thesis, Technical Report No. 15, University of Wisconsin Computing Center, 1970.

[Ng91] S. Ng, "Improving Disk Performance Via Latency Reduction", *IEEE Transactions on Computers*, Vol. 40, No. 1, January 1991, pp. 22-30.

[Patt88] D. Patterson, G. Gibson, R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois, May 1988, pp. 109-116.

[Poly93] C. Polyzois, A. Bhide, D. Dias, "Disk Mirroring with Alternating Deferred Updates", *Proceedings of the 19th International Conference on Very Large Data Bases*, Dublin, Ireland, 1993, pp. 604-617.

[Quan93] Quantum Corporation, "ProDrive 700/1050/1225S Product Manual", Publication Number 81-102480-02, March 1993.

[Rama92] K. Ramakrishnan, P. Biswas, R. Karedla, "Analysis of File I/O Traces in Commercial Computing Environments", *Proceedings of the 1992 ACM SIGMETRICS and PERFORMANCE '92 International Conference on Measurement and Modeling of Computer Systems*, Newport, Rhode Island, June 1992, pp. 78-90.

[Reyn88] R. Reynolds, "Sector-Based Disk Scheduling in the UNIX^{TM} System V Environment", *Proceedings of the ACM Southeastern Regional Conference*, Mobile, Alabama, April 1987, pp. 648-652.

[Ruem93] C. Ruemmler, J. Wilkes, "UNIX^{TM} Disk Access Patterns", *Proceedings of the Winter USENIX Conference*, San Diego, California, January 1993, pp. 405-420.

[Ruem94] C. Ruemmler, J. Wilkes, "An Introduction to Disk Drive Modeling", *IEEE Computer*, Vol. 27, No. 3, March 1994, pp. 17-28.

[Scra83] R. Scranton, D. Thompson, D. Hunter, "The Access Time Myth", IBM Research Report, RC 10197, September 1983.

[SCSI93] "Small Computer System Interface-2", ANSI X3T9.2, Draft Revision 10k, March 1993.

[Seag92] Seagate Technology, Inc., "SCSI Interface Specification, Small Computer System Interface (SCSI), Elite Product Family", Document Number 64721702, Revision D, March 1992.

[Seag92a] Seagate Technology, Inc., "Seagate Product Specification, ST41600N and ST41601N Elite Disc Drive, SCSI Interface", Document Number 64403103, Revision G, October 1992.

[Seag93] Seagate Technology, Inc., "Seagate Product Specification, ST11750/1 N/ND and ST12550/1 N/ND Barracuda Disc Drive, SCSI Interface", Document Number 64403700, Revision A, January 1993.

[Seam66] P. Seaman, R. Lind, T. Wilson "An Analysis of Auxiliary-Storage Activity", *IBM System Journal*, Vol. 5, No. 3, 1966, pp. 158-170.

[Selt90] M. Seltzer, P. Chen, J. Ousterhout, "Disk Scheduling Revisited", *Proceedings of the Winter USENIX Conference*, Washington, D.C., January 1990, pp. 313-324.

[Sugg90] D. Suggs, "The Use of Future Knowledge in the Design of a Disk Scheduling Algorithm", Master's Thesis, Clemson University, South Carolina, 1990.

[Sugg93] D. Suggs, "Disk Subsystem Performance and Reliability Enhancements Through the Use of Carnival Mirrors", Ph.D. Thesis, Clemson University, South Carolina, 1993.

[Teor72] T. Teorey, T. Pinkerton, "A Comparative Analysis of Disk Scheduling Policies", *Communications of the ACM*, Vol. 15, No. 3, March 1972, pp. 177-184.

[Wilh76] N. Wilhelm, "An Anomaly in Disk Scheduling: A Comparison of FCFS and SSTF Seek Scheduling Using an Empirical Model for Disk Accesses", *Communications of the ACM*, Vol. 19, No. 1, January 1976, pp. 13-17.

[Wort94] B. Worthington, G. Ganger, Y. Patt, "Scheduling Algorithms for Modern Disk Drives", *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Nashville, Tennessee, May 1994, pp. 241-251.

[Wort94a] B. Worthington, G. Ganger, Y. Patt, "Scheduling for Modern Disk Drives and Non-Random Workloads", University of Michigan, Technical Report CSE-TR-194-94, March 1994.

[Wort95] B. Worthington, G. Ganger, Y. Patt, J. Wilkes, "On-line Extraction of SCSI Disk Drive Parameters", *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, Ottawa, Ontario, May 1995, pp. 146-156.