Fair Schedulers for Disk Drives

A Project Report
Submitted in partial fulfilment of the
requirements for the Degree of
Master of Engineering
in
Computer Science and Engineering

by

Tulika Pradhan

Department of Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012

June 1995

Contents

Acknowledgements

I would like to thank my project guide, *Dr. Jayant R. Haritsa* for his excellent guidance throughout the course of this project work. Without his ideas and persistent encouragement, this project would not have materialized. I also thank *Dr. Y. Narahari* for his valuable suggestions.

I express my thanks to *Raj Kumar* for helping me in the initial stages of the project. I thank my friends, in particular *Vidya*, *Vardhini*, *Joyjeet*, *Surekha* and *Rupa* for making my stay in IISc., pleasant and memorable.

Finally, I thank my parents for their constant love and encouragement.

Abstract

Time-shared systems are usually expected to provide fair service to their users. In this report, we examine the problem of scheduling **disk** requests in a fair manner. A new fairness metric is defined and a detailed simulation study is made of the fairness characteristics of a variety of classical disk scheduling algorithms over a wide range of system loads and scheduling granularities. The experimental results show that these algorithms fail to provide adequate fairness. To address this problem, new algorithms that consistently provide fair service are presented.

A drawback of these new algorithms is that fairness is achieved at the cost of a significant increase in the mean response time for all users. Therefore, improved algorithms are constructed which reduce the mean response time, keeping the fairness value satisfactory. More fundamentally, it is found that, in disk systems, it is not possible to achieve fairness by redistributing the cylinder mean response times. An approximate lower bound on the mean response time for fair algorithms is determined, and it is found that the mean response times of the improved algorithms come close to this bound.

Chapter 1

Introduction

There is an increasing demand on computer systems to provide service guarantees to end-users with regard to a variety of metrics. Real-time applications, for example, may require meeting of task deadlines to be guaranteed. Time-shared systems, on the other hand, are usually expected to provide **fair service**. While processor scheduling has been extensively investigated in this context, the issue of providing fairness to disk users, although considered, has not been analyzed to a similar extent. In this report, we present a detailed study of the fairness characteristics of a variety of disk scheduling algorithms.

1.1 Fairness

The term "fairness" is used in several different senses in the performance evaluation literature. In particular, a common interpretation of fairness is "starvation prevention", that is, no request will starve indefinitely, or equivalently, every request will eventually complete. Another interpretation of fairness is that a resource divides its processing capacity *equally* among all its current users, that is, it provides "service fairness". We use this interpretation of fairness in our study of disk schedulers. Note that service fairness implies starvation prevention.

Service fairness may be achieved in two different ways: "instantaneous" and "statistical". In instantaneous fairness, a resource provides fair service at every instant of operation. This type of fairness is exhibited by processor scheduling algorithms such as the idealized Processor-Sharing scheduler, or its practical version, the Round-Robin scheduler [Klei75]. Statistical fairness, on the other hand, is different in that the resource provides fair service to a group of users when each of them makes a large number of service requests, and the fairness is evaluated over the entire set of requests.

1.2 Disk Scheduling

Before considering the type of fairness provided by disk systems, we need to know the basic features of a disk system and why it is so important.

The performance of a computing system depends significantly on techniques used for handling secondary storage devices. A commonly used device for secondary storage is the *movable head disk*. A disk system consists of a stack of disks mounted together with read/write heads attached to the end of a movable arm. Each surface is divided by concentric circles into *tracks* which are further divided into *sectors*. The set of tracks equidistant from the centre forms a *cylinder*. Accessing a record on disk involves three sequential operations. First is the *seek operation* which involves movement of the disk head to the specified cylinder. Then follows the rotation of the disk so that the required record comes under the disk head. The time taken for this operation is the *latency time*. Lastly, the required record is actually transferred to/from the disk. In most disk drives, the seek time is the most significant part of the total access time. The seek distances traveled by the disk head depend on the order in which the requests from the disk queue are served. Hence, the scheduling policy largely determines the performance characteristics including the fairness charateristics, of the disk system.

In disk systems, it is inherently impossible to achieve instantaneous fairness. This limitation arises due to two factors: First, the context-switch time in disks, that is, the time taken between the completion of one request and the start of service of the next request, is not constant. Instead, it is a function of the seek distance between the physical disk locations accessed by the requests. Second, the service process in disks is non-preemptive, that is, once the transfer of data has been initiated for a request, other requests cannot be served until the data transfer is completed. Due to these reasons, the disk capacity as perceived by two concurrent requests may be radically different. Consequently, we can only hope to achieve statistical service fairness in disk drives; that is, the average response time for each concurrent user should be approximately the same when evaluated over a large number of disk accesses.

The scheduling algorithms that are typically used in commercial disk drives do not achieve statistical fairness. Consider the case where user A happens to have all her files stored in the innermost cylinders whereas user B happens to have her files stored in the neighborhood of the central cylinders, and that they make several file access requests in a time-shared environment. In such a situation, several studies (e.g. [Oney75]) have shown that classical disk scheduling algorithms such as SCAN, SSTF, etc. are unfair in that they provide better average response time to User B as compared to User A.

1.3 Performance Metric

There is a large body of literature related to performance of disk schedulers (e.g. [Fran69, Teor72, Oney75, Swar80, Geis87b, Wort94, IEEE94]). Many of these studies have also considered the fairness issue (e.g. [Teor72, Geis87a, Wort94]). The metric used to measure fairness in these studies has typically been either the variance or the coefficient of variation of user response times. However, as discussed above, an inherent feature of disk service is that there is significant variation in response times. Therefore, evaluating fairness through the overall variance is perhaps an unsuitable metric. Instead, given that the requests of each user are typically bunched around a few cylinders, fairness should be evaluated by measuring the variance of the *cylinder* mean response times. This means that we consider a disk system to be fair if users have approximately the same average response time *independent* of where on the disk their files happen to be clustered. This concept is shown pictorially in Figure 1.1 for a disk with N cylinders, and in this study, we use the coefficient of variation of the cylinder mean response times as the fairness metric.

Optimization of performance with respect to one metric usually results in degraded performance for some other metric. In disk scheduling, fairness is usually achieved at

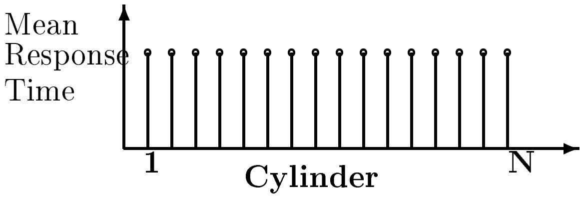


Figure 1: Fair Disk Scheduler

the expense of increased response times. Our study characterizes this tradeoff between fairness and mean response time.

1.4 Goals

In summary, we address the following issues in this report:

- Evaluating the fairness performance of classical disk schedulers.
- Designing new disk scheduling algorithms which are completely fair.
- Characterizing the tradeoff between fairness and mean response time.
- Optimizing the new algorithms with respect to mean response time.
- Determining a lower bound on the mean response time for fair algorithms.

A feature of our study is that a range of "scheduling granularities" are considered. The scheduling granularity limits the set of time instants at which scheduling decisions can be made, and is determined by the disk controller implementation. We have looked at three scheduling granularities, STOP, GATE and DYNAMIC. In the STOP scheduling granularity, scheduling decisions can be made only when the disk head is stopped. In GATE, scheduling is done on a group basis, and in DYNAMIC, scheduling decisions can be made at any time.

Another feature of our study is that the implementation of the classical schedulers is specially directed towards maximizing their fairness. For example, a "single service" policy is followed by all schedulers except those of the GATE scheduling granularity where the algorithms require that "exhaustive service" policy be followed. In single service policy, whenever a cylinder chosen for service has multiple outstanding requests, only one request for the cylinder is served before the disk head moves on to serve other cylinders whereas in exhaustive service policy, all requests of a particular cylinder are served before the head moves on to serve other cylinders.

1.5 Results

Our simulation results show that even the above fair versions of classical disk schedulers generally exhibit poor fairness characteristics. Most of these scheduling algorithms have [illegible]

To address this problem, improved algorithms are constructed for each granularity, which reduce the mean response time *keeping* the fairness value satisfactory. These algorithms have cylinder response time distributions similar to curve C of Figure 1.2. But this is still not good enough. What we would like to have is an *ideal fair algorithm* which is fair and has the same mean response time as the unfair classical algorithm. This ideal fair algorithm will have the cylinder response time distribution curve like curve D of Figure 1.2. Surprisingly, it is found that such an ideal fair algorithm is **not feasible** due to the inherent drawbacks of a disk system. A *lower bound* on the mean response time for fair algorithms is determined. A fair algorithm with this value of mean response time is the *optimal fair algorithm*. It is observed that the mean response time of the improved algorithms come close to the lower bound determined and hence these algorithms are *nearly* optimal.

In most applications, an optimal fair algorithm would not be preferable because, when compared to the classical one, the response times are either increased, for central cylinders, or remain the same, for extreme cylinders. However, in some applications, this might be useful. This will be the case when it is required that several jobs complete simulateously because it is more advantageous for requests to wait in the input queue rather than the output queue.

1.6 Organization

The rest of this report is organized as follows: We present, in Chapter 2, an overview of related work on disk scheduling. The performance model is described in Chapter 3. The fair algorithms and the results of the simulation experiments are presented in Chapter 4. Chapter 5 highlights the drawbacks of the fair algorithms and presents improved algorithms. The simulation study of the improved algorithms is also described here. The inherent difficulties in achieving fairness in a disk system are discussed in Chapter 6. An approximate lower bound on the mean response time for fair algorithms is also determined here. In Chapter 7, the robustness of the improved algorithms with respect to variations in the model, workload and parameters is examined. We conclude in Chapter 8 with a summary of the results presented.

Chapter 2

Overview of Related Work

Several scheduling policies have been proposed in literature for handling a queue of requests for moving head disks. These include **FCFS**, **SSTF** [Denn67], **SCAN** [Denn67], standard **LOOK** [Mert70] and **SVS(N)** algorithms [Geis87a]. The modified versions of these basic algorithms such as **CSCAN** [Seam66], **CLOOK** [Teor72], **VSCAN(R)** [Dani83] and **CVSCAN(N, R)** [Geis87a] have also been proposed and studied. The name SCAN is often loosely used to also refer to the LOOK algorithm. To avoid confusion, we use the term POLL [Coff73] in this report to refer to the original SCAN algorithm.

The performance metric used to study the behavior of the above algorithms have typically been expected waiting time, expected queue length, variance and coefficient of variation of waiting times. A linear combination of the mean and the variance of the waiting times has also been used as a *combined goodness measure* [Teor72] to compare various algorithms. With respect to one or more of these metrics, all the above algorithms improve the performance of the disk system over FCFS. We now describe the various classical algorithms and their performance below.

2.1 Classical Algorithms

The **FCFS** (First Come First Serve) scheduler is a simple algorithm which serves requests in the sequence of their arrival to the disk system. It takes no advantage of the positional relationships between the requests. Hence it has a random seek pattern and saturates at lighter loads [Teor72]. In contrast, the **SSTF** (Shortest Seek Time First) scheduler serves the request that is nearest to the current disk head position. Its goal is to reduce the mean response time by minimizing the seek distance traversed in servicing requests. At high arrival rates, SSTF tends to become localized in a small set of cylinders in an attempt to exhaust all requests to that region, thereby greatly increasing the response time for requests at other cylinders. The **POLL** scheduler sweeps the disk head, back and forth, from one end of the disk to the other, serving requests as it passes by each cylinder. It usually exhibits higher response times for extreme cylinders as compared to the centrally-located cylinders [Oney75]. This is because the extreme cylinders see a sequence of alternate short and long gaps in the inter-visit times of the disk head as compared to the inter-visit times of the central cylinders where it is almost periodic. The **LOOK** scheduler implements a slight modification of the POLL algorithm. Instead of reversing the direction of the head movement at the disk extremities, it reverses the direction of the head movement whenever no further requests exist in the current sweep direction.

The **CPOLL** (Circular POLL) scheduler is a "unidirectional" variant of POLL. The basic feature here is that the disk head serves requests in only one sweep direction, say, inside-out. In the opposite direction, CPOLL strokes directly to the innermost cylinder before commencing the sweep again. The **CLOOK** (Circular LOOK) scheduler is a similar variant of LOOK wherein the disk head, on the return path, strokes directly to the innermost cylinder having a request. For heavier loads CLOOK performs better than other policies with respect to the combined goodness measure of [Teor72].

The **VSCAN(R)** (where R is a real number between 0, and 1) scheduler maintains a current sweep direction and services next, that request which is the smallest effective distance away from the current position. The effective distance to a request in the current sweep direction is the physical distance in cylinders from the head position. The effective distance to a request in the opposite direction is its physical distance $+R\times$ (total cylinders). Thus VSCAN(0.0) = SSTF and VSCAN(1.0) = SCAN. VSCAN(R) provides a performance continuum with respect to both mean and variance of request waiting times. Since these two measures tend to reach optimal values near opposite endpoints of the continuum, for objective functions that are a linear combination of the two [Teor72], intermediate points on the continuum provide better performance than both SCAN and SSTF [Geis87a].

The **CVSCAN(N, R)** (where R is a real number between 0 and 1 and N is an integer) scheduler computes the effective distance by the method of VSCAN(R) but uses the average effective distance of N requests in each direction as the basis for deciding which of the two requests is serviced next. If there are fewer than N requests in either direction then the minimum of the number of requests in the two directions is used to compute the average.

The **SVS(N)** scheduler makes batches of size N and within each batch, the following algorithm is used. The first arrival to an empty queue becomes the request in service, but retains its position at the head of the queue until its service is completed. For each request arriving thereafter, the algorithm proceeds as follows. Starting with $n = 1$ in the queue, it compares two distances: The first is the distance from the arriving requests to entry n and the second is the distance from entry n to entry $n + 1$ (taken to be $+\infty$ if entry $n+1$ does not exist). If the first distance is smaller than the second, the new request is inserted immediately after entry n. Otherwise n is incremented and the procedure is repeated. Requests from this queue are serviced in order. This algorithm shows sharp improvement over FCFS in mean response time with little sacrifice in variance [Geis87a].

Many studies have considered the fairness issue (e.g. [Teor72, Geis87a, Wort94]). The metric used to characterize fairness have either been the variance or the coefficient of variation of the waiting times. This is not a very suitable metric for fairness amongst users because, in a disk system, the variation in waiting times is large. Hence, what we need is a more suitable fairness metric using which the fairness of various algorithms can be studied.

Chapter 3

Performance Model and Methodology

In our study of the fairness characteristics of the various disk scheduling algorithms, a simulation model of a disk system was used. A disk module with a single movable arm is modeled. Table 3.1 summarizes the key parameters of the model. The model consists of a disk which has a fixed number of cylinders on which user data is stored. The requests arrive to the system, wait in a single queue and after they are served, leave the system immediately. This corresponds to the *open model* for the disk system. The *closed model* is described and its performance results discussed in Chapter 7.

The disk head seek times are computed using the common linear approximation formula [Teor72]

$$T_s = \alpha + \beta \times n_c \quad (n_c \geq 1) \tag{3.1}$$

where n_c is the seek distance measured in number of cylinders; α represents the startup time for the disk head to start seeking; β is the seek-distance multiplier. The values of α and β are constants for a specific disk. The number of cylinders and the values of the seek-time constants are set by the $NumCyl$, $Alpha$ and $Beta$ parameters, respectively. The time taken for the disk controller to make a scheduling decision is assumed to be negligible.

User requests for disk access arrive in a Poisson stream, whose rate is specified by the *ArrRate* parameter. Other arrival patterns are also considered and discussed in Chapter 7 Each request asks for data from a specific disk cylinder. The amount of data to be transferred is the same for all requests and is equal to the basic unit of disk transfer (i.e., a block). The position of the requested data within the cylinder is not modeled explicitly. Instead, the rotational latency and the data transfer time are assumed to be constant and are lumped together into the *Access* parameter. This is because in most disk drives the seek time largely contributes to the total time required to access a disk block. The data requests are *uniformly* distributed over the entire disk. Other distributions for data requests are also considered and discussed in Chapter 7.

Parameter	Meaning
$NumCyl$	Number of disk cylinders
Alpha	Seek startup time
Beta	Seek-distance multiplier
Access	Rotation time plus transfer time
ArrRate	Request arrival rate

Table 3.1: Open Model Parameters

3.1 Statistics

The simulator used to obtain the results is implemented in the Simscript II.5 simulation language and experiments were conducted on a SUN Sparc machine running SunOS 4.1. The simulator was instrumented to generate a large number of statistics. These included the first three moments (i.e., mean, variance, and skew) of the response time distribution at each cylinder and the overall response time distribution. Secondary measures such as the average seek distance between requests, the average queue length and the utilization, were used to verify the correctness of the implementation and to lend insight into the observed performance behavior.

The fairness metric was computed as the coefficient of variation of the cylinder mean response times, that is,

$$Fairness = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(RT_i - ORT)^2}}{ORT} \tag{3.2}$$

where N is the number of cylinders, RT_i is the average response time for cylinder i and ORT is the overall average response time. In this formulation, a scheduler with a fairness value of 0.0 is completely fair. As a practical matter, we consider fairness values that are *less than* 0.05 to be satisfactorily fair and those *greater than* 0.2 to be very unfair.

All the cylinder response time distribution graphs in this report show mean values that have relative half-widths about the mean of less than 5 percent at the 90 percent confidence level, with each experiment having been run until at least 1,000,000 requests were processed by the disk system. This was determined using the sequential procedure based on the method of batch means ([Law77]). The initial transient effect is eliminated by ignoring the first 1000 requests. Only statistically significant differences are discussed here.

3.2 Parameter Settings

The parameter settings which were kept constant in all the experiments are listed in Table 3.2. The values for the disk constants are similar to the ones in [Teor72] and are therefore somewhat outdated. However, since these values have been used in several followup studies, we too have done so to aid in comparison of results. Experiments using parameters of current disks were also made and the results of those experiments were found to be qualitatively similar (Chapter 7).

3.3 Scheduling Granularities

Different scheduling granularities have been considered and simulated. The scheduling granularity limits the set of time instants at which scheduling decisions can be made, and is determined by the sophistication of the disk controller. Three scheduling granularities

Parameter	Meaning
NumCyl	1000
Alpha	30.0 msec.
Beta	0.50 msec./cylinder
Access	8.0 msec.

Table 3.2: Parameter Settings

are mentioned in the literature in connection with disk scheduling: GATE, STOP and DYNAMIC.

The basic concept of **GATE** is that requests are served in *groups*. Each new group consists of the entire set of requests that arrived while the previous group was being served. This means that once a group is formed, the server's "gate" is closed for new arrivals until the current group is completely served. After an idle period, the group formation process is re-initiated by making the first arrival form the new group.

In **STOP**, scheduling choices are made only when the disk head is stopped. Once a decision has been made and the disk head has started seeking towards the chosen request, that request has to be completely served before the next scheduling choice can be made. Therefore, requests which arrive when the disk head is busy are not considered for scheduling until the current request has been served completely.

In **DYNAMIC**, scheduling decisions can be made at any time. That is, even during the process of moving towards a selected request, the scheduler can change its decision about which request to serve next based on the arrival of new requests.

3.4 Fairness Modification

We have implemented most of the schedulers to obtain maximum fairness. This is done by maintaining the "single service" policy. In this policy, whenever there are multiple outstanding requests for a cylinder that has been chosen for service, only a single request among this set is served before the disk head moves on to serve other cylinders. The request that is served is the one that arrived earliest to the disk system among this set.

This is done for STOP and DYNAMIC scheduling granularities to avoid starvation ([Shoh92]). For GATE, since the requests are served in groups, it is preferable to follow the "exhaustive service" policy where all requests for the cylinder chosen for service are served before the head moves on to the next cylinder to be served. This is done because, anyway, *all* requests in the current group are served before requests in the next group are considered. Hence, exhaustive service policy reduces the mean waiting times and does not lead to starvation.

Chapter 4

Simulation Results

In this chapter, we present the results from our experiments on the fairness characteristics of the various disk scheduling algorithms. The results are presented separately for the three different scheduling granularities: GATE, STOP and DYNAMIC. For each granularity, a new algorithm is presented which exhibits consistent fairness over the entire range of system loading. Twelve graphs are shown for each of these – the first four graphs show the cylinder response time behavior for user request arrival rates ranging from light load to heavy load; the next three graphs show the global mean response time, the global coefficient of variation, and the cylinder fairness, respectively, as a function of arrival rate; the next graph characterizes the tradeoff between fairness and mean response time. The last four graphs show the mean seek distance between requests, overall mean queue length, utilization and skewness of mean cylinder response times as a function of the arrival rate. The arrival rates that were simulated are between 1 and 20 requests per second; for higher arrival rates, the algorithms go into saturation. As discussed in Chapter 3, fairness values that are less than 0.05 are considered to be satisfactorily fair.

To ensure graph clarity, the performance of only a representative subset of the algorithms for which simulation was done is shown for each granularity. In addition, although the mean response time was measured for all 1000 disk cylinders, the cylinders are bunched together in groups of 50 and the average response time of each group is plotted in the cylinder response time versus cylinder number graphs, again for graph clarity. Finally, to serve as a basis for comparison, the performance of FCFS is shown in all the experiments.

The implementation of most of the schedulers is dependent on the scheduling granularity. For each granularity, the implementation details are described first and then the results are presented. Since the FCFS implementation is the same for all granularities, its performance remains the same for all the experiments.

4.1 GATE Scheduling

We first consider the GATE scheduling granularity, where requests are served in groups.

4.1.1 Classical Algorithms

The classical algorithms which are considered are LOOK, CLOOK and CPOLL, apart from FCFS. Their implementations are as follows:

LOOK: The requests in the current group are served as the head sweeps the disk surface. After serving the current group, the LOOK scheduler forms a group of the new arrivals. If all the requests in the new group are ahead of the current head position, then the sweep direction of the head is not reversed. However, if there are requests behind the head, the sweep direction is reversed and the entire group is then served in the normal manner. When there are no requests in the system the head remains idle at the cylinder of the previous request served.

CLOOK: The requests in the current group are served as the head sweeps the disk surface moving in the direction inside–out. After serving the current group, the CLOOK scheduler forms a group of the new arrivals. The disk head then moves to the lowest cylinder with a request in the new group and then serves the entire group in the normal manner. When there are no requests in the system the head remains idle at the cylinder of the previous request served.

CPOLL: The requests in the current group are served as the head sweeps the disk surface from the inner end to the outer end. After serving the current group, the CPOLL scheduler forms a group of the new arrivals. The disk head then moves to the outer extremity before rewinding to the inner one. If all requests in the new group were in front of the disk head, then they get served as the head moved towards the outer edge. When there are no requests in the system the head remains idle at the cylinder of the previous request served.

4.1.2 New Algorithm : Park Poll (PPOLL)

A new algorithm, PPOLL, has been developed which shows consistent fairness over the entire range of system loading. It is described as follows:

PPOLL: This is a variant of POLL algorithm except that whenever the system finishes serving a group of requests, the disk head moves to the last cylinder in the current sweep direction and *parks* there before serving the next group.

It is implemented as follows : After serving the current group, the PPOLL scheduler moves to the extreme disk cylinder in the current sweep direction. It then forms a group of the new arrivals, reverses the sweep direction and serves the entire group in the normal manner.

4.1.3 Results

The results of the experiments evaluating the performance of the above algorithms are shown in Figures 4.1a through 4.1*l*. From these figures, we observe first that the performance of FCFS saturates much earlier than that of the other algorithms (Figure 4.1e, 4.1k). Also, it is quite unfair under light loads, favoring the central cylinders over the extreme cylinders (Figures 4.1a, 4.1b). This is because at light loads, new requests typically arrive to an empty system. The disk head will be found idle at the cylinder of the previously served request. Since the requests are uniformly distributed, the head position is also uniformly distributed. Therefore, new requests arriving at the central cylinders are, on the average, only $N/4$ cylinders away from the current disk head position, whereas requests at the extreme cylinders are, on the average, $N/2$ cylinders away from the current disk head position (where N is the total number of disk cylinders). As expected, for FCFS, the average seek distance between requests is $N/3$, irrespective of the arrival rate (Figure 4.1i).

Moving on to LOOK, we observe that under light loading (Figures 4.1a, 4.1b) it favors the central cylinders over the extreme cylinders. This is because at low arrival rates, the algorithm behaves essentially like FCFS as most of the groups consist of a single request. However, at higher arrival rates, it becomes quite fair. In contrast, the circular algorithm CLOOK is unfair throughout the entire loading range, favoring the inner cylinders at the expense of the outer cylinders at high arrival rates. This is because requests which arrive during a sweep are only served in the next sweep and the direction of the head while servicing requests is *inside–out*. Hence the outer cylinders have to wait longer for service than the inner cylinders. At low arrival rates, this also performs like FCFS.

The CPOLL scheduler shows an interesting behavior. At very low arrival rates (Figure 4.1a), new requests typically arrive to an empty system. Since the position of the disk head is uniformly distributed, requests for the inner cylinders typically have to wait while the disk head moves to the outermost cylinder and then strokes back to the innermost cylinder before recommencing the sweep. However, requests for the outer cylinders will usually be served directly since the disk head will tend to be located behind them. Therefore, the outer cylinders have a better response time than the inner cylinders. For moderate to heavy loads (Figures 4.1b - 4.1d), however, the inner cylinders are favoured, and this is for the same reasons as described above for CLOOK.

Finally, turning our attention to PPOLL, we find that its cylinder response time distribution is nearly flat over the entire loading range (Figure 4.1a - 4.1d). This can be explained as follows: At low arrival rates, requests tend to arrive to an empty system and find the disk head parked at either of the ends. This parking feature of PPOLL ensures that the average distance of the new request from the current head position is always $N/2$ (Figure 4.1i), unlike FCFS (or classical POLL), where the average distance ranges from $N/4$ to $N/2$ as described above. Therefore, there is no discrimination against any cylinder

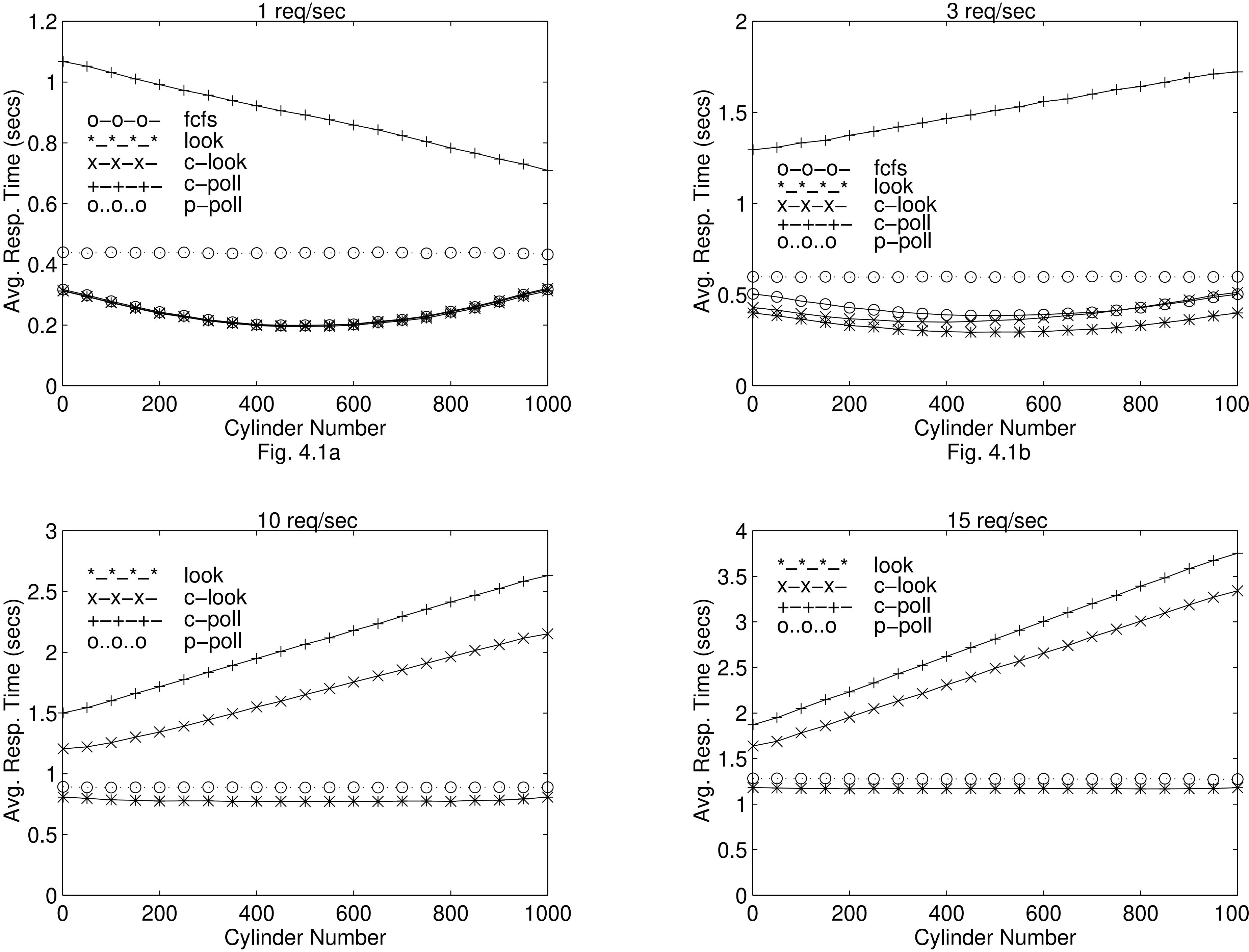


Fig. 4.1a

Fig. 4.1b

Fig. 4.1c

Fig. 4.1d

GATE Algorithms

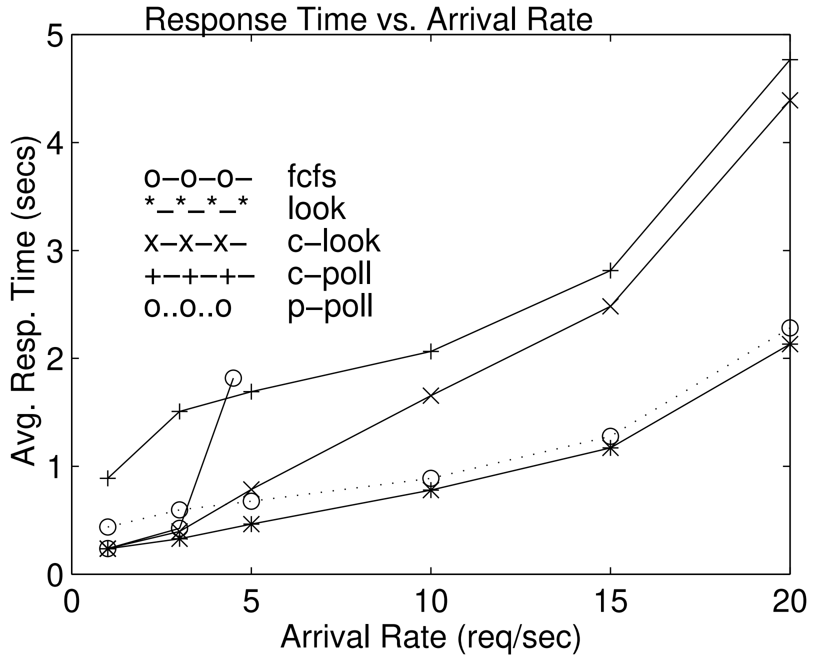


Fig. 4.1e

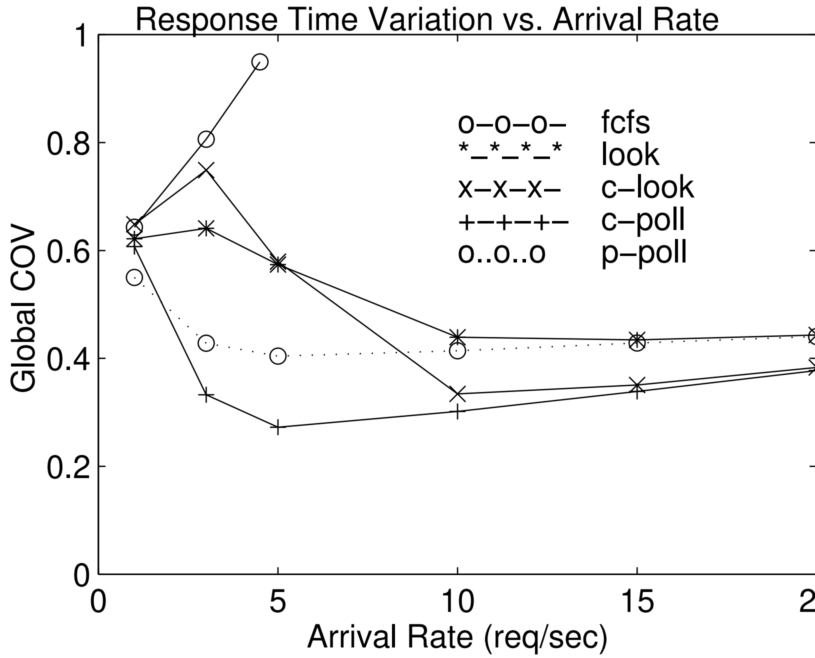


Fig. 4.1f

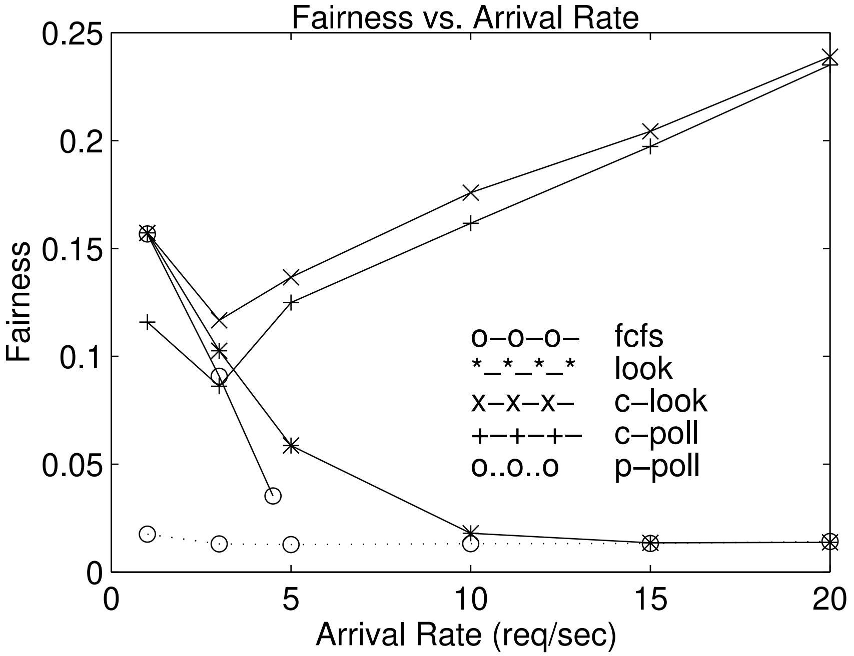
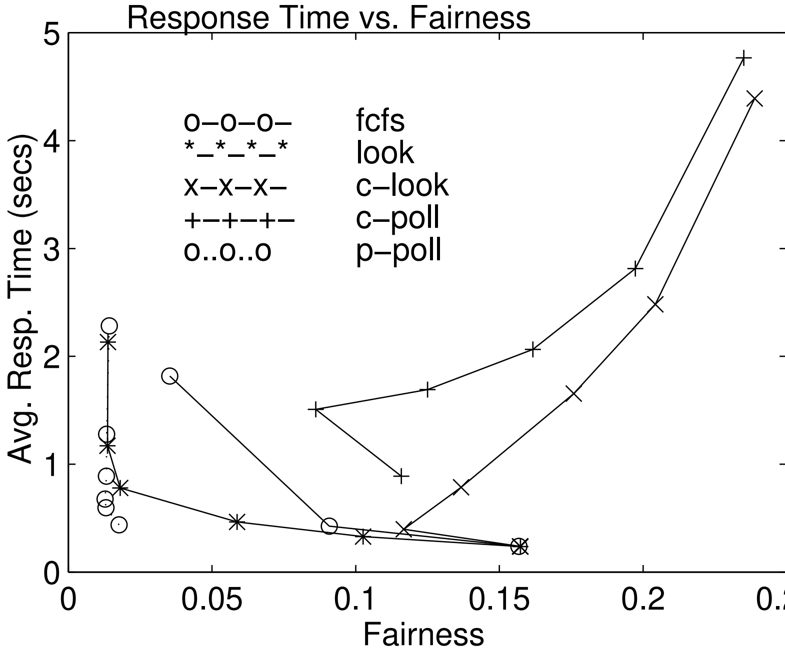


Fig. 4.1g



GATE Algorithms

Fig. 4.1h

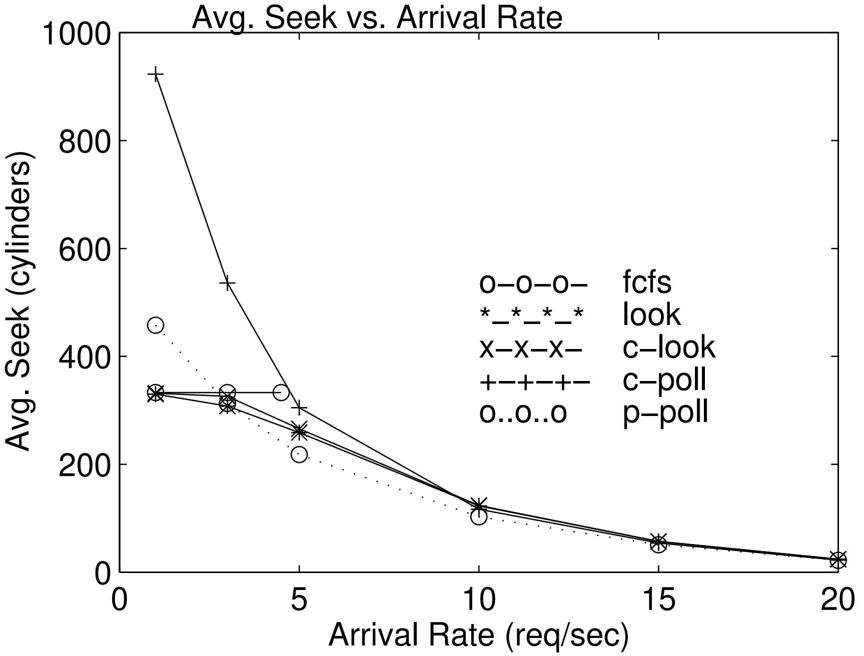


Fig. 4.1i

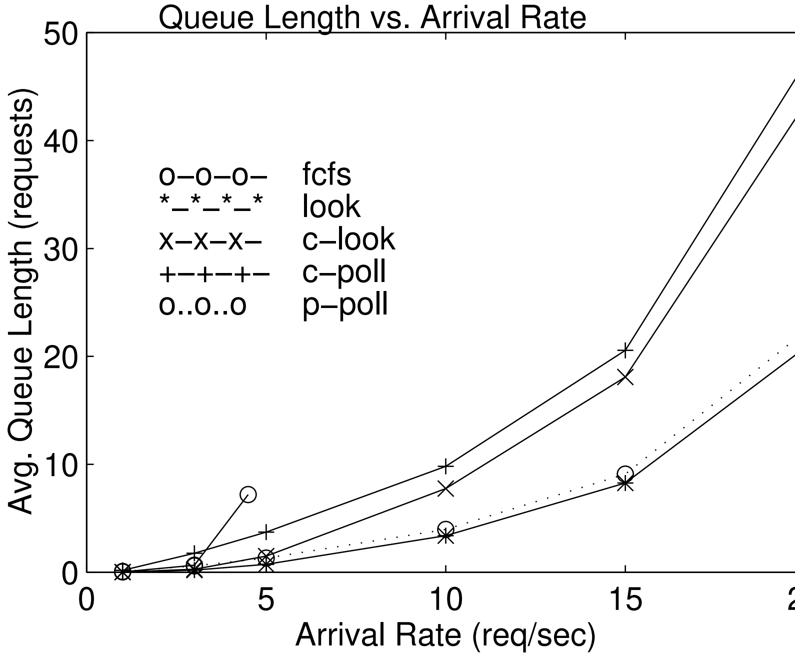


Fig. 4.1j

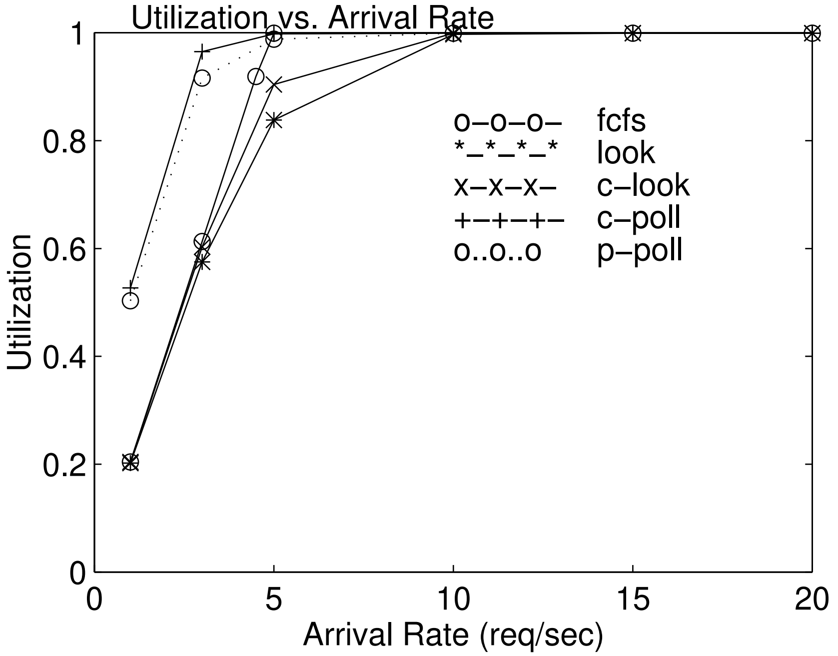


Fig. 4.1k

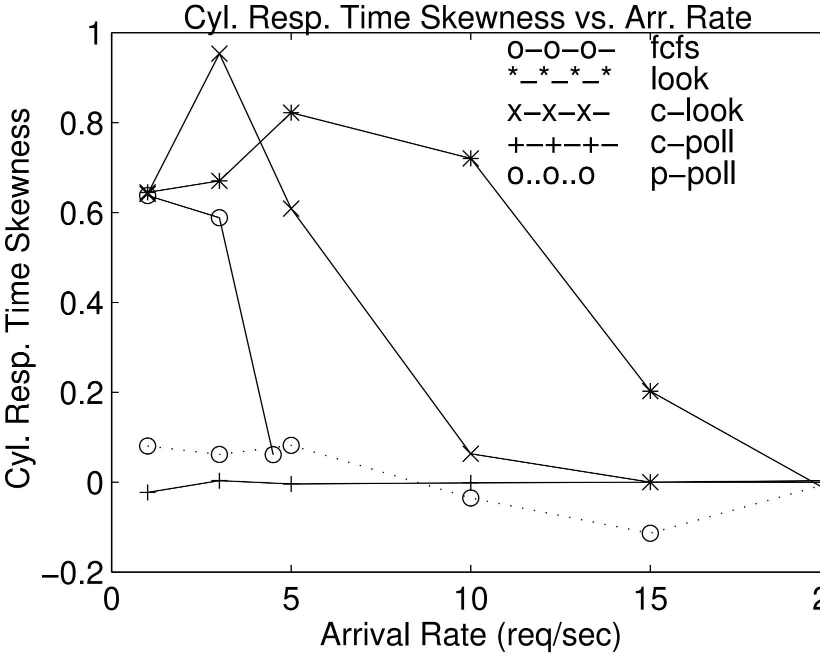


Fig. 4.1l

GATE Algorithms

group. At higher loads, when the disk head is moving inside–out the inner cylinders get a better response, whereas when the disk head is moving outside–in the outer cylinders are benefited. Since each group on the average, contains similarly distributed set of requests and each inside–out sweep is followed by an outside–in sweep, the average response time is the same for all cylinders. At high arrival rates, LOOK is almost identical to PPOLL (as should be expected) since the requests in each group span most of the disk surface and disk idle periods are rare.

In Figure 4.1f, the overall coefficient of variation of response time is plotted against the arrival rate. Note that all the algorithms exhibit significant variation. More interestingly, the variation of PPOLL is greater than that of the circular algorithms for most of the loading range. This clearly shows that PPOLL is achieving fairness, not by minimizing the variation inherent in disk scheduling, but instead by "evenly distributing" the variation.

In Figure 4.1h, the tradeoff between mean response time and fairness is plotted. The points here are joined in ascending order of the arrival rates. This clearly shows that PPOLL is satisfactorily fair in that its fairness level is always below 0.05 at all arrival rates. In addition, for medium to heavy loading, its response time is better than that of most of the other algorithms. Under light loading, however, its fairness is purchased at the expense of a significant increase in mean response time.

In Figure 4.1i, the average seek between requests is plotted against the arrival rate. For all algorithms, the average seek between requests decreases with increase in the arrival rate. Under heavy loadings, the mean seek becomes nearly the same for all algorithms. This is because, for all algorithms, the disk head is hardly ever idle.

In Figure 4.1j, the overall queue length is plotted against the arrival rate. It is observed that, for PPOLL, the queue length is smaller than that for the circular algorithms. In Figure 4.1k, the utilization of the disk head is plotted against the arrival rate. Although, due to the scale used for plotting utilization, it seems that the algorithms saturate beyond 10 requests/second, it is not so. The utilizations for arrival rates greater than 10 requests/second, are slightly less than 1.0. They saturate only when the arrival rate exceeds 20 requests/second.

In Figure 4.1*l*, the skewness of the mean cylinder response time is plotted against the arrival rate. It is observed that for PPOLL, the skewness value is small for all arrival rates, unlike that for most of the classical algorithms.

An analytical derivation of an expression for the mean waiting times for PPOLL is given in Appendix A. It assumes the workload of Chapter 3. It is shown that for this workload, the cylinder mean waiting times are independent of the cylinder number. Hence the algorithm is fair under these conditions.

4.2 STOP Scheduling

We now move on to considering the STOP scheduling granularity, where scheduling decisions are made only when the disk head is stopped.

4.2.1 Classical Algorithms

The classical algorithms considered in this category are SSTF, LOOK and CLOOK, apart from FCFS. Their implementations are as follows:

SSTF: After the service of a request, the request queue is searched for the request that is closest to the current head position, and this request is then served.

LOOK: After the service of a request, the request queue is searched for the request closest to the current head position in the current sweep direction, and this request is then served. If no requests remain in the current sweep direction, the disk head reverses direction and searches again. When there are no requests in the system, the head remains idle at the previous request position.

CLOOK: After the service of a request, the request queue is searched for the request closest to the current head position in the current sweep direction, and this request is then served. If no requests remain in the current sweep direction, the disk head is stroked back to the innermost cylinder with a pending request. When there are no requests in the system, the head remains idle at the previous request position.

4.2.2 New Algorithm : Park Circular Look Poll (PCPL)

A new algorithm, PCPL, has been developed which shows consistent fairness over the entire range of system loading. It is described as follows:

PCPL: This is based on both CLOOK and CPOLL with two modifications: First, whenever the system becomes emptied of requests the disk head moves to the nearest disk extremity and parks there. If the head is parked at the inner end, the service direction when new requests arrive is inside–out. However, if the head is parked at the outer end, the service direction for further requests is outside–in. Therefore, although the service is always unidirectional as in CLOOK or CPOLL, the direction of the sweep may change dynamically. Second, when no further requests exist in the current sweep direction, the disk head is stroked back to the *extreme* disk cylinder. Therefore, PCPL is similar to CLOOK in that it terminates its sweep if no further requests exist in front of the head and similar to CPOLL in that it stokes back to the extreme cylinder to make the next scheduling decision.

It is implemented as follows : After the service of a request, the request queue is searched for the request closest to the current head position in the current sweep direction, and this request is then served. If no requests remain in the current sweep direction, the disk head is moved back to the extreme disk cylinder. Whenever the disk becomes idle, the disk head moves to the closer disk extremity. If the head moved to the inner end, the subsequent sweep direction is inside–out. On the other hand, if the head moved to the outer end, the subsequent sweep direction is outside–in.

4.2.3 Results

The results of the experiments evaluating the performance of the above algorithms are shown in Figures 4.2a through 4.2*l*. From these figures, we observe that, as in GATE scheduling, FCFS saturates much earlier than the other algorithms and is unfair for light loads. All classical algorithms here perform essentially similar to FCFS at very low arrival rates (Figure 4.2a). This is because for all these algorithms, under such loads, new requests typically arrive to an empty system, find the head idle at the previously served cylinder and move straight to the requested cylinder. Moving on to SSTF, we observe that it is quite unfair, especially at higher arrival rates where the central cylinders receive significantly better treatment than the extreme cylinders. This is because the disk head tends to get localized at the central cylinders thereby increasing the response time for the extreme cylinders. The LOOK scheduler is also unfair over the entire loading range. This is because the extreme cylinders see a sequence of alternate short and long gaps in the inter-visit times of the disk head as compared to the inter-visit times of the central cylinders where it is almost periodic. This results in long queues developing at the extreme cylinders and relatively smaller queues at the central cylinders.

Turning our attention to CLOOK, we observe that at heavy loads it becomes increasingly fair, but there is a distinct bias against the innermost set of cylinders (Figures 4.2c and 4.2d). This bias is because of the following reason: A scheduling decision is made at the end of service of every request. Hence, when there are no requests in front of the disk head, the decision is made *before* the head begins to rewind to the chosen request cylinder. Due to this, requests for cylinders which are lower than the chosen request cylinder, which arrive during the return stroke, will be served only in the next sweep. Whereas, requests arriving during the return stroke, for cylinders higher than the chosen request cylinder will be served in the current sweep anyway. As the arrival rate increases, this bias is reduced to *fewer* inner cylinders as the mean cylinder to which the disk head rewinds to, becomes lower.

Finally, turning our attention to PCPL, we observe that it is satisfactorily fair throughout the entire loading range, at no stage exceeding 0.05 (Figure 4.2g), although under light loading, there is some bias against the central cylinders (Figure 4.2a, 4.2b). The reason for its good performance is the following: Since the decision as to which request is served next is made after the head rewinds to an extreme cylinder, the response time for inner cylinders improves when the sweep direction is inside–out. Similarly the response time for the outer cylinders improves when the sweep direction is outside–in. The middle

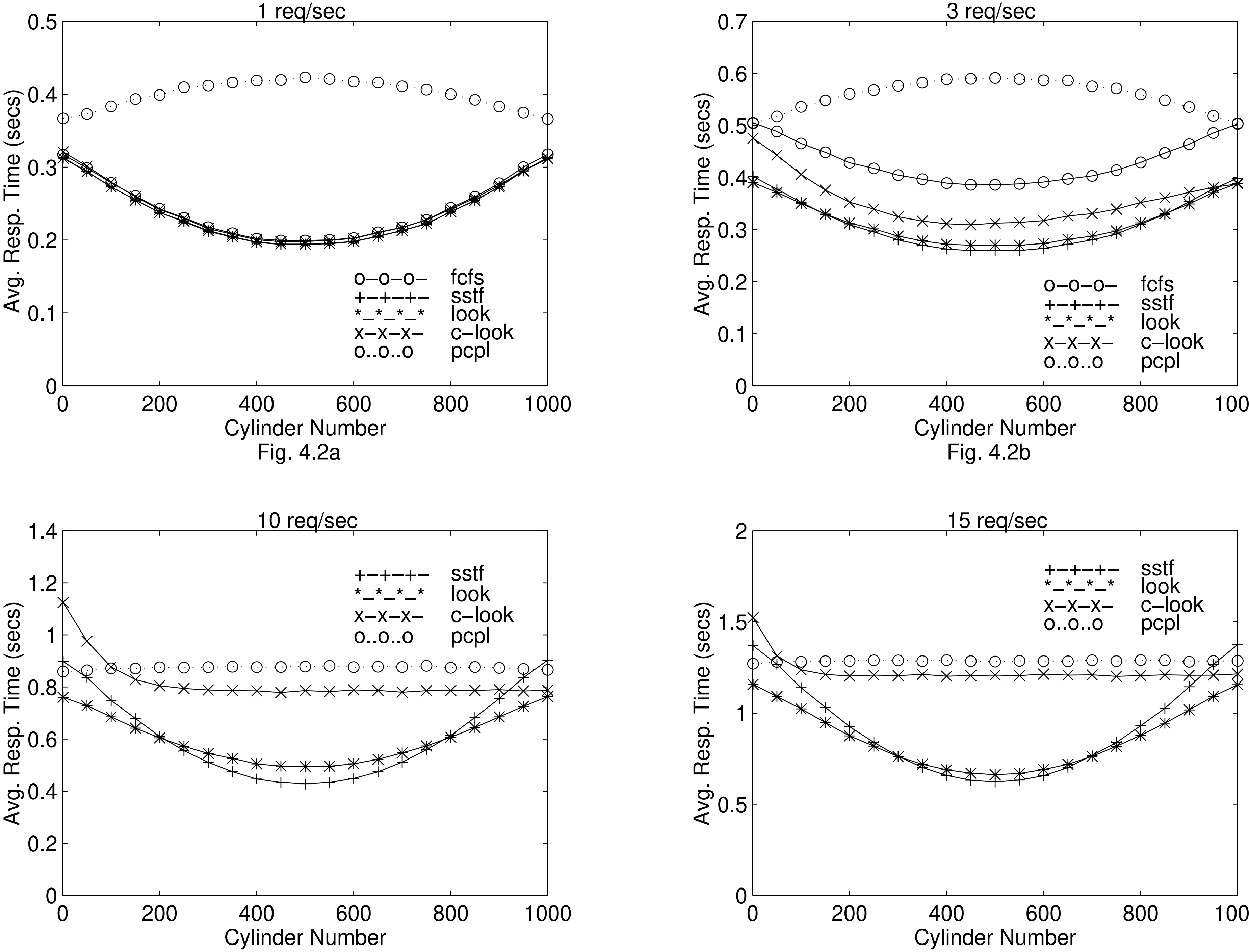


Fig. 4.2a

Fig. 4.2b

Fig. 4.2c

Fig. 4.2d

STOP Algorithms

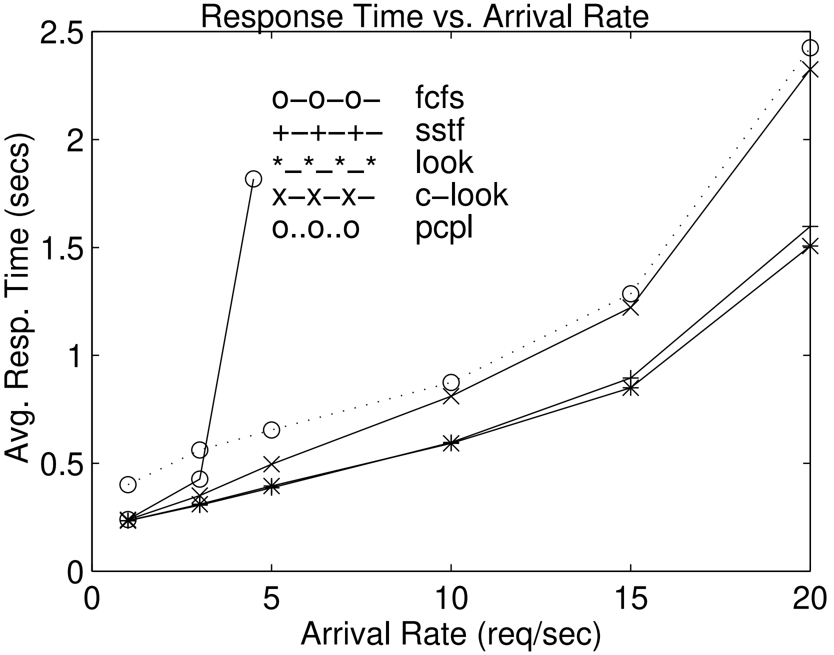


Fig. 4.2e

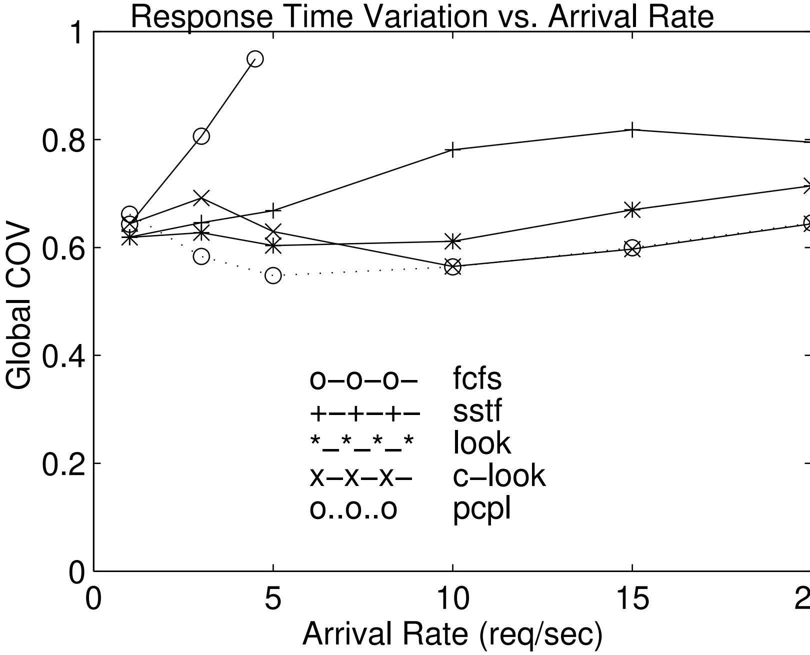


Fig. 4.2f

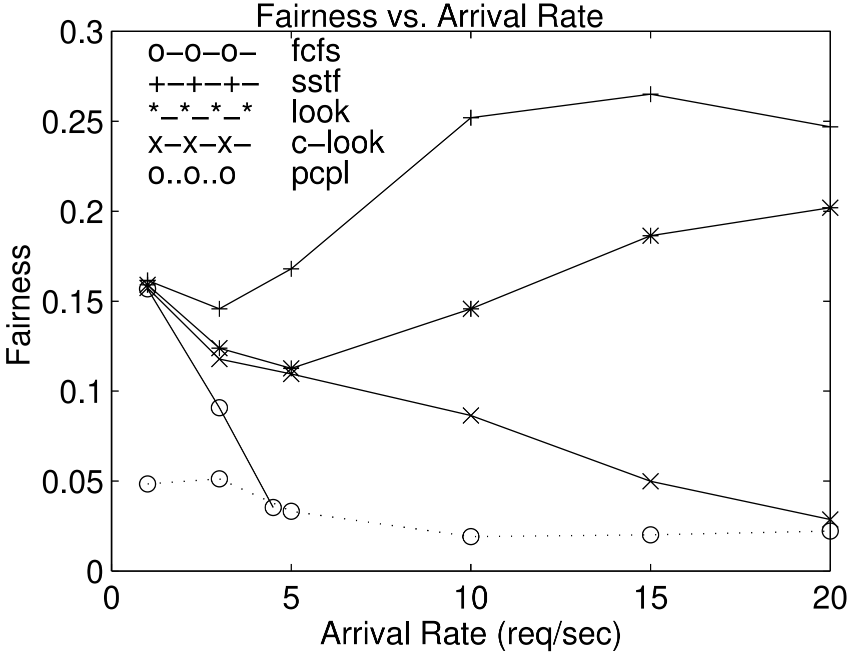


Fig. 4.2g

STOP Algorithms

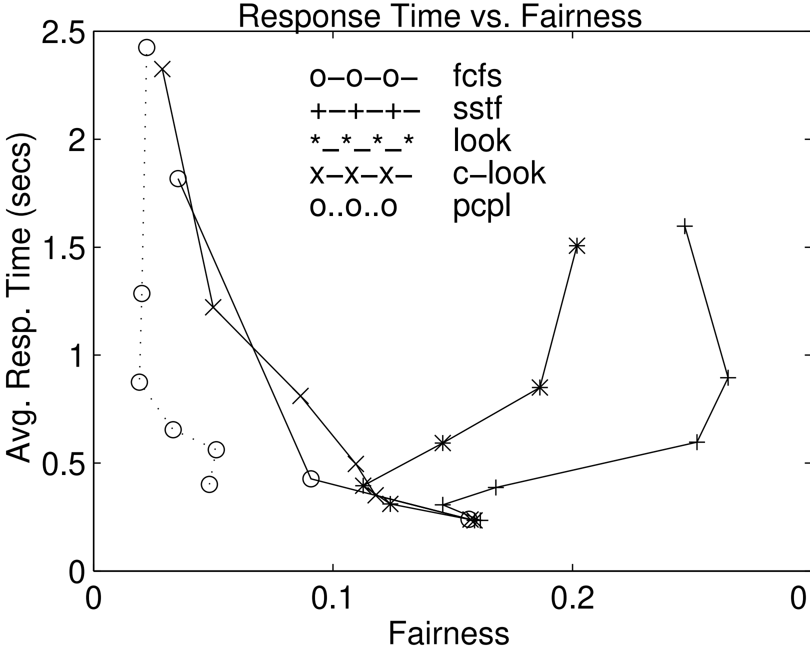


Fig. 4.2h

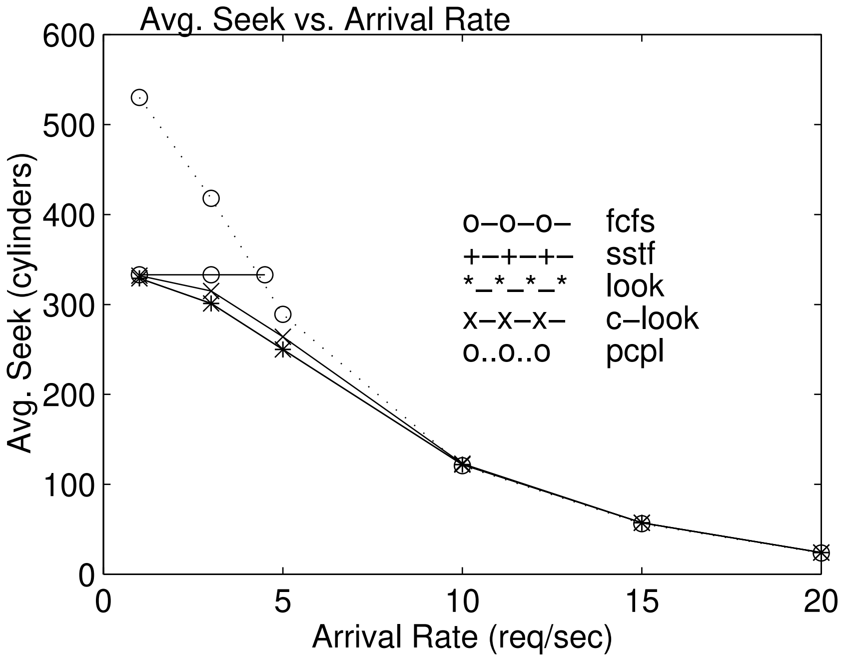


Fig. 4.2i

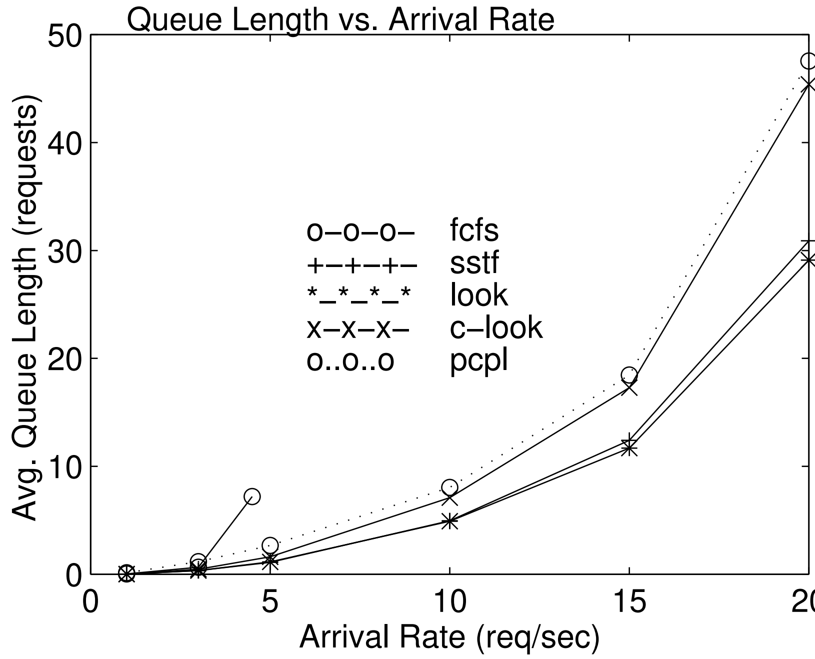


Fig. 4.2j

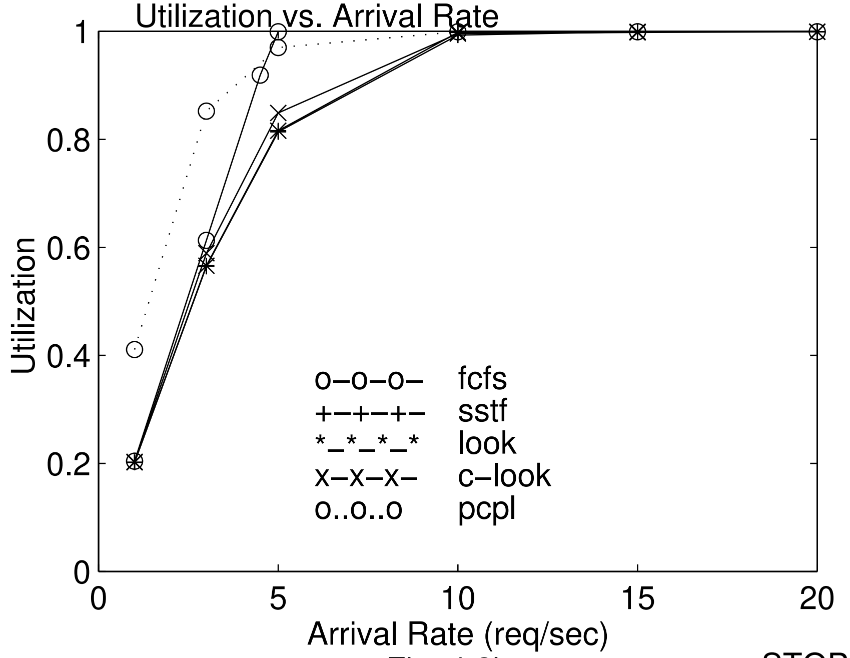
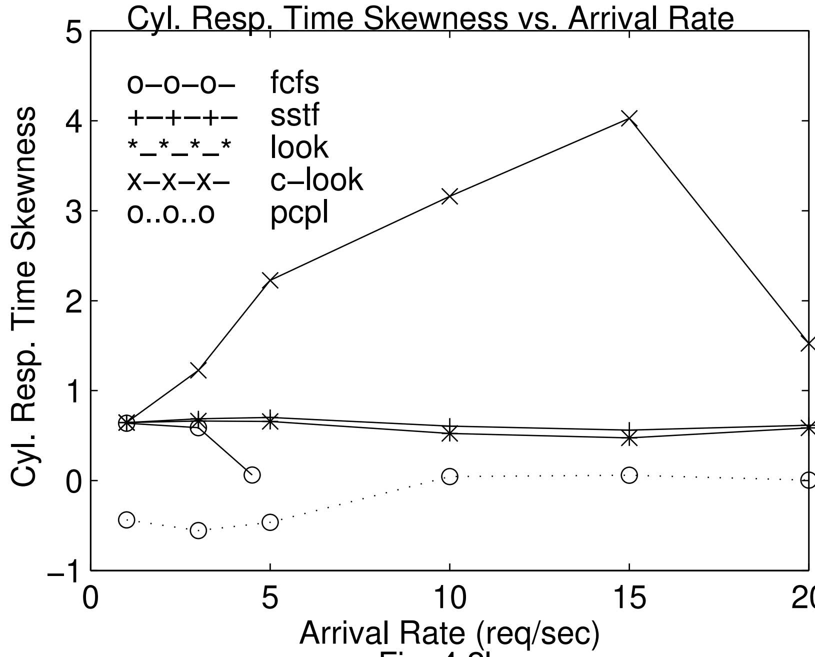


Fig. 4.2k



STOP Algorithms

Fig. 4.2l

cylinders do not receive this benefit and their response times are therefore slightly higher. At higher loadings, the disk head rarely changes its sweep direction since the disk head is busy nearly always. Therefore, it is almost identical to CLOOK; however, since the decision as to which request to serve is made only after stroking back to the innermost cylinder, the bias against the inner cylinders exhibited by CLOOK is not present here.

It should also be noted that the fairness of PCPL is obtained at the expense of a significant increase in mean response time as compared to SSTF or LOOK for the entire loading range (Figures 4.2e, 4.2h). With respect to CLOOK, however, it shows higher response time only for low arrival rates.

4.3 DYNAMIC Scheduling

We now move on to considering the DYNAMIC scheduling granularity, where scheduling decisions can be made at any time.

4.3.1 Classical Algorithms

The classical algorithms considered in this category are SSTF, LOOK and CLOOK, apart from FCFS. They are implemented as follows:

SSTF: After serving a request, the disk head moves towards the closest request. If, during the disk head movement, a new request arrives whose position is closer to the current disk head position as compared to that of the originally selected request, the scheduling decision is changed and the head starts moving towards the new arrival.

LOOK: The disk head sweeps across the disk surface serving requests. As it passes each cylinder, it serves a pending request (if any) for that cylinder. If no requests exist for cylinders in front of the current head position, the head reverses its direction and sweeps in the opposite direction.

CLOOK: The disk head sweeps across the disk surface in the inside–out direction serving requests. As it passes each cylinder, it serves a pending request (if any) for that cylinder. If no requests exist for cylinders in front of the current head position, the head reverses its direction, and strokes back until there exist no requests for cylinders that are inner to the current head position. It then begins to sweep the disk surface again in the normal fashion.

4.3.2 New Algorithm : Non-Stop Circular Poll (NSCP)

A new algorithm, NSCP, has been developed which shows consistent fairness over the entire range of system loading. It is described as follows:

NSCP: This scheduler is identical to CPOLL except that the disk head continuously sweeps the disk surface even when the system is emptied of requests.

It is implemented as follows : The disk head sweeps from the innermost cylinder to the outermost cylinder. As it passes each cylinder, it serves a pending request (if any) for that cylinder. On reaching the outermost cylinder, it strokes back to the innermost cylinder. The head continues its sweeping/stroking motion even if there are no pending requests in the system.

4.3.3 Results

The results of the experiments evaluating the performance of the above algorithms is shown in Figures 4.3a through 4.3*l*. From these figures, we observe first that, as in GATE and STOP scheduling, FCFS saturates much earlier than the other algorithms and is quite unfair under light loading. As in STOP scheduling all classical algorithms here perform essentially similarly to FCFS at very low arrival rates as new requests typically arrive to an empty system.

Moving on to SSTF, we observe that although its response time is significantly better than the other algorithms, its fairness characteristics are extremely poor. This is due to its tendency to get localized in the central cylinders which results an an increase in response times for the extreme cylinders.

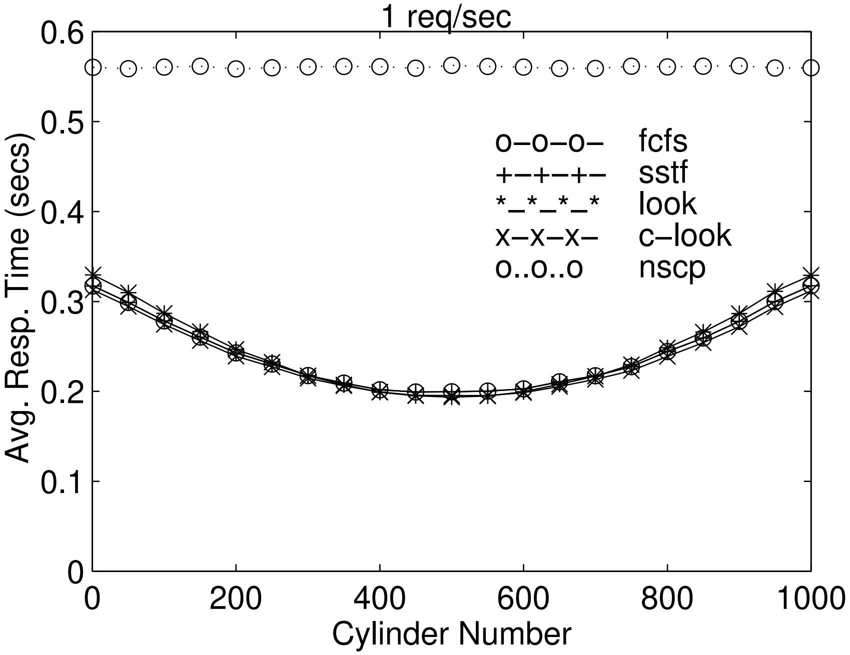


Fig. 4.3a

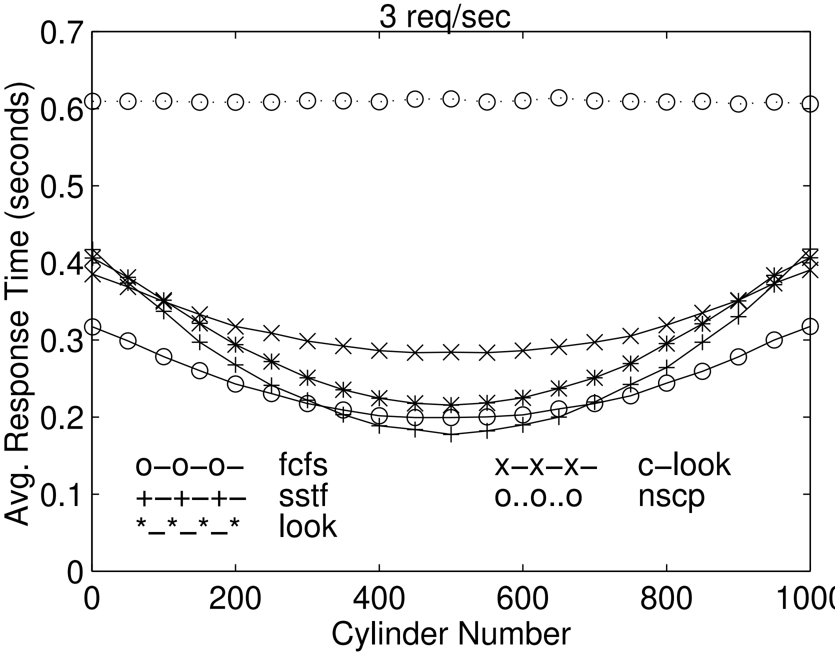


Fig. 4.3b

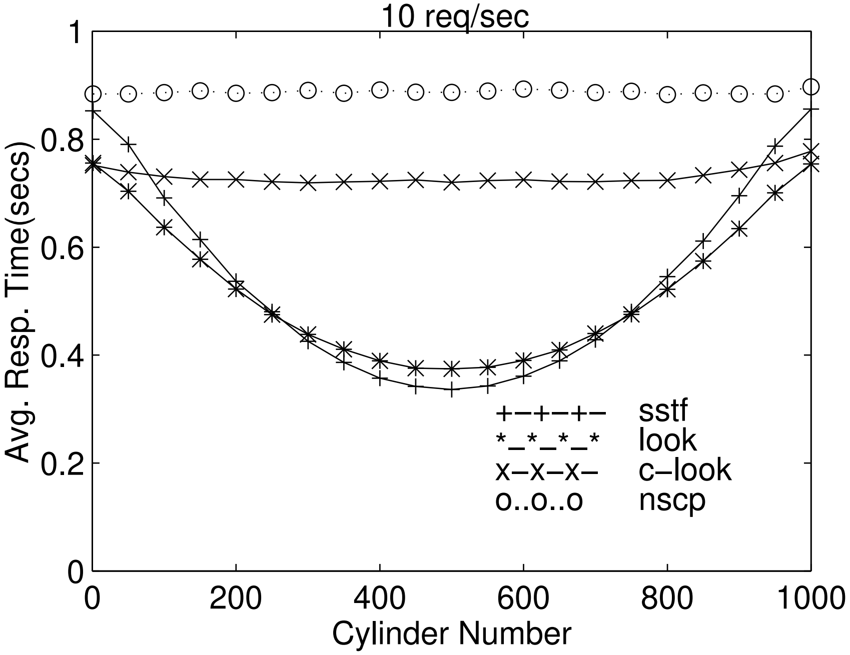


Fig. 4.3c

DYNAMIC Algorithms

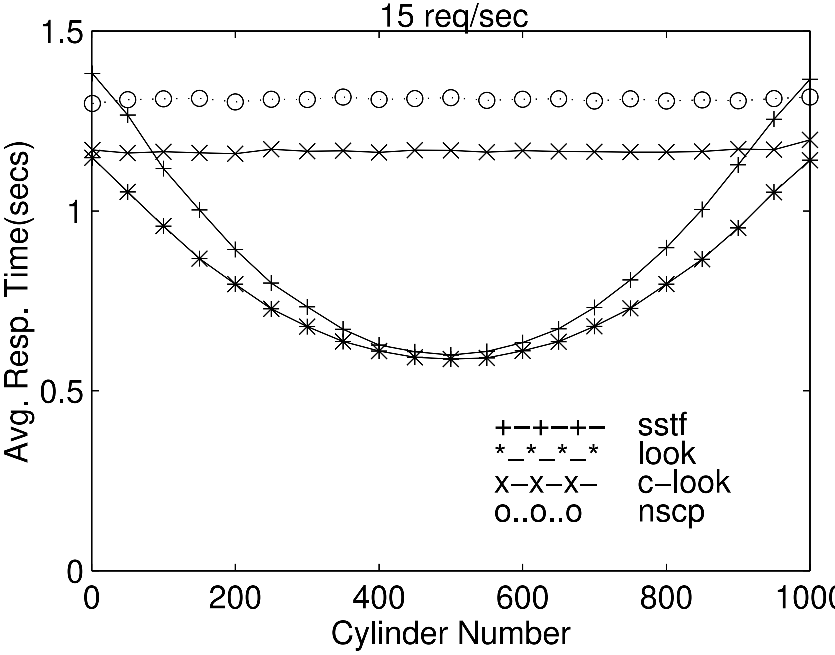


Fig. 4.3d

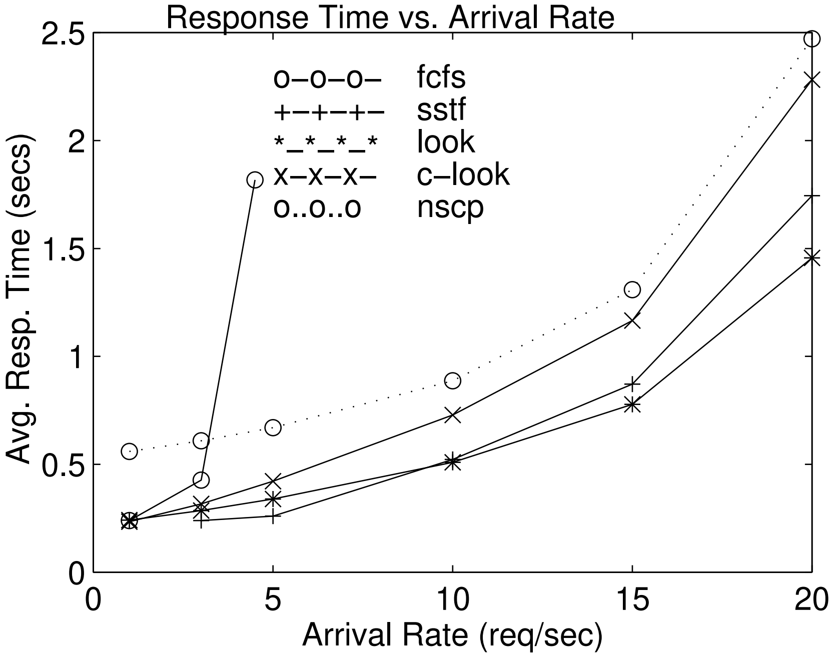


Fig. 4.3e

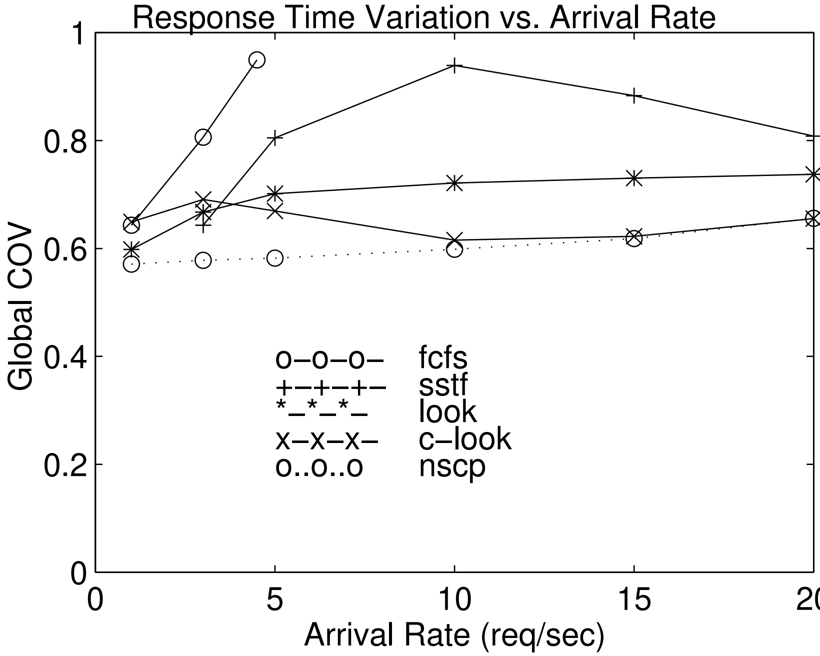


Fig. 4.3f

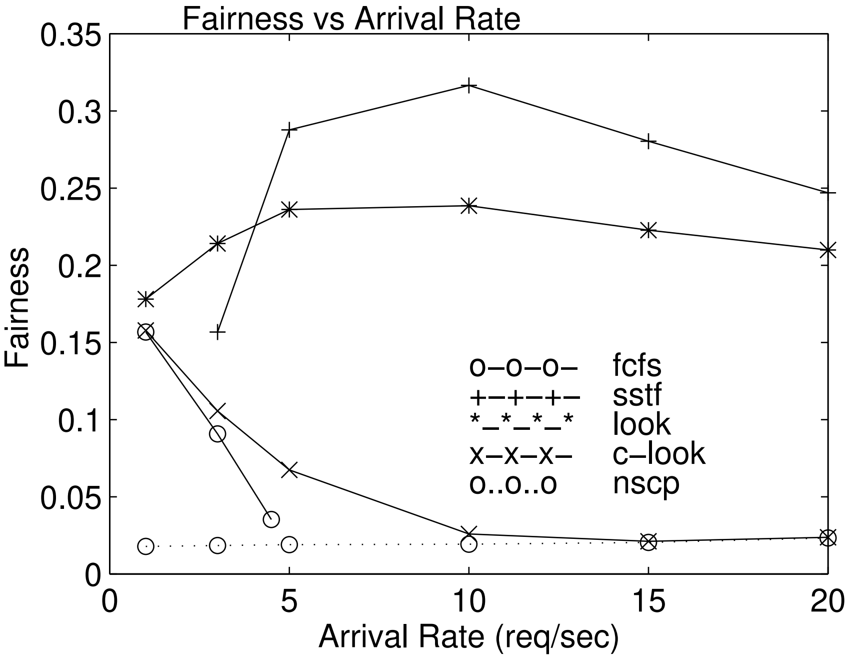


Fig. 4.3g

DYNAMIC Algorithms

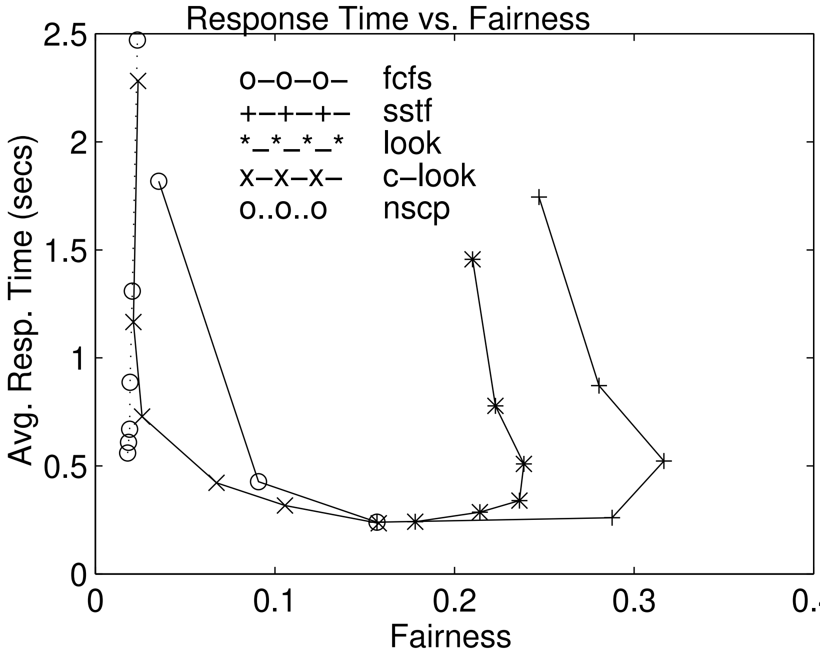


Fig. 4.3h

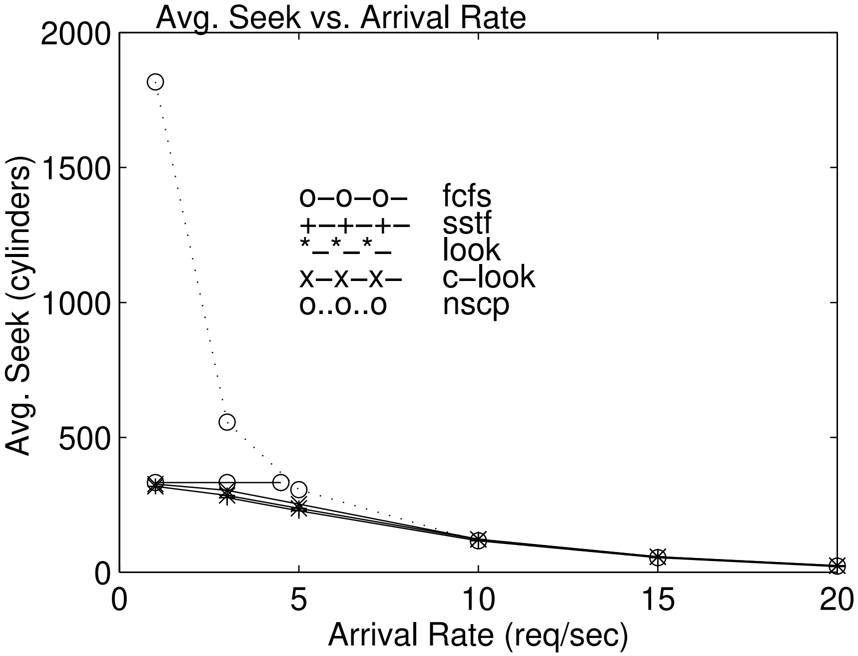


Fig. 4.3i

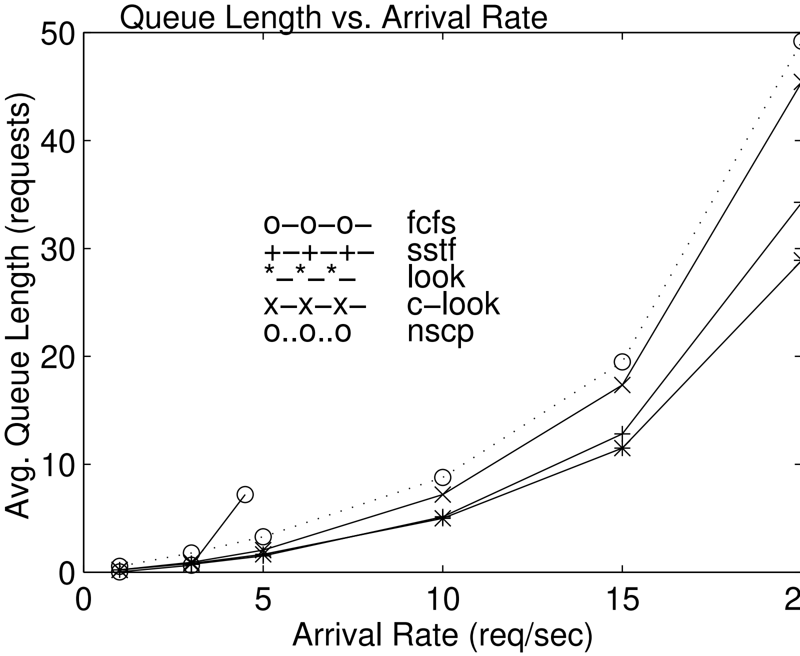


Fig. 4.3j

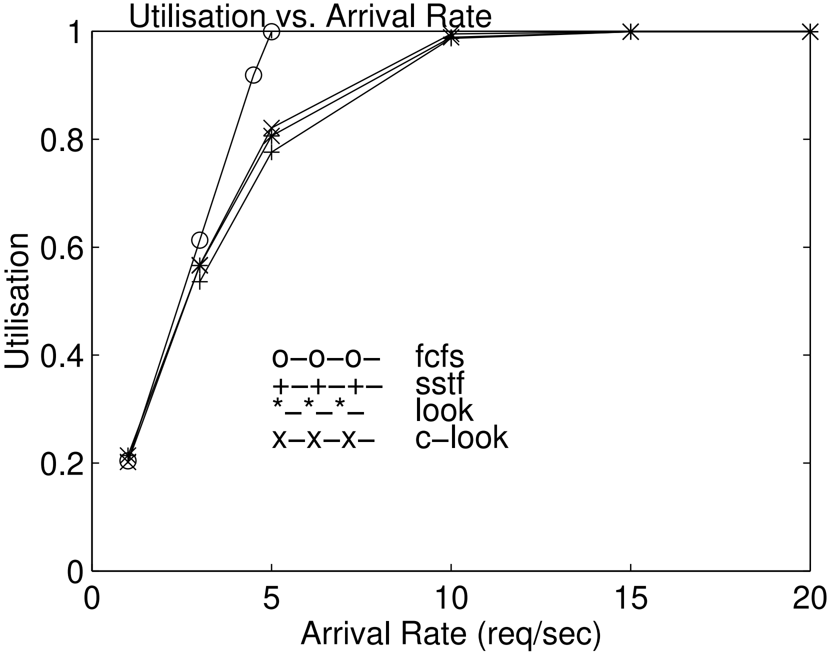


Fig. 4.3k

DYNAMIC Algorithms

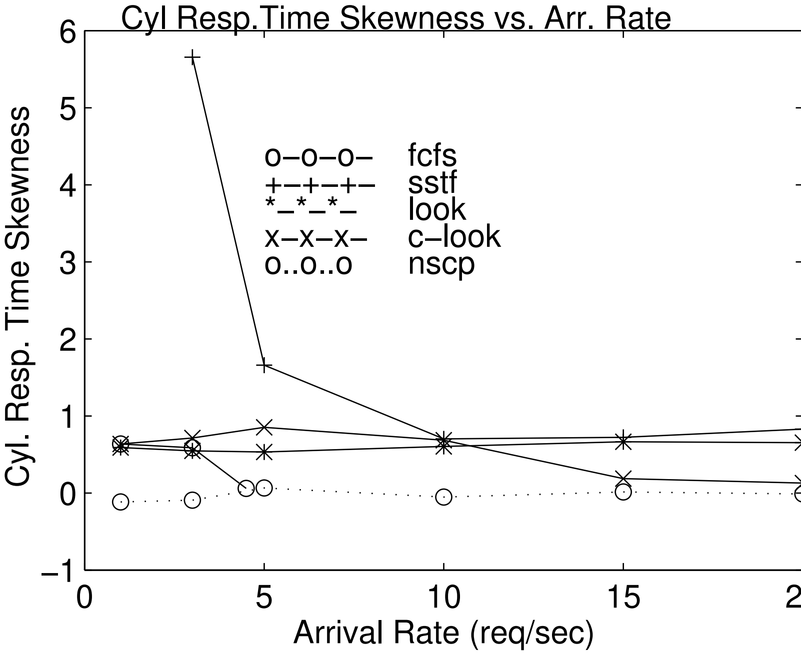


Fig. 4.3l

Turning our attention to LOOK, we see that its performance is almost as poor as that of SSTF. This is due to the same deficiency which led to its poor performance under STOP scheduling, that is, there is much higher variance in the inter-visit times of the extreme cylinders as compared to that of the central cylinders. Coming to the CLOOK scheduler, we observe that it is satisfactorily fair at high arrival rates. There is no bias against inner cylinders as in CLOOK for STOP scheduling. This is because the scheduling decision is made dynamically when the head strokes back to the lowest cylinder with a pending request.

Finally, turning our attention to NSCP, we see that it is extremely fair across the entire loading range (Figure 4.3h). The reason for its good performance is the following: Although the CPOLL scheduler performance is not shown in the graphs, it exhibited a marked bias against the inner cylinders at light loads in our experiments. This is because requests for the inner cylinders typically have to wait for the disk head to move to the outer end of the disk and then stroke back to the inner end before commencing the sweep, as discussed for GATE scheduling. The NSCP scheduler, although closely resembling CPOLL, eliminates this bias. This is because since the disk head is always moving, *any* request may need this rewinding of the disk head as the disk head may be found anywhere moving either in the sweep direction or in the rewind direction. Due to this, the average distance traveled between the arrival and the service of requests, for all cylinders is N. Note that this is not the same as the average seek plotted in Figure 4.3i, which is the average seek *between* requests. Since the disk head moves even when there are no requests in the system, for low arrival rates, this value is higher than the seek between the arrival and the service of a request. For high arrival rates, NSCP behaves just like CPOLL which is fair due to the unidirectional service. Note that the utilization for NSCP has not been plotted in Figure 4.3k. This is because, in NSCP, the head moves even when there are no requests in the system.

An analytical derivation of an expression for the mean waiting times for NSCP is given in Appendix A. It assumes the workload of Chapter 3. It is shown that for this workload, the cylinder mean waiting times are independent of the cylinder number. Hence the algorithm is fair under these conditions.

Note that, for most of classical algorithms, the plots of global coefficient of variation, and fairness (e.g. 4.1f, 4.1g, 4.1h) have kinks near arrival rate 3 req/sec. This can be explained as follows: At low arrival rates, most of these algorithms behave like FCFS as requests typically arrive to an empty system and finds the head at the cylinder of the previously served request. Hence, under these loads, the typical charateristics of these algorithms are not evident and the order of serving requests are nearly the same as for FCFS. The characteristics of the algorithms come into picture only at higher arrival rates when there are mulitiple requests waiting in the queue. Hence, as the load becomes heavier, the plots charaterize the different scheduling algorithms.

Chapter 5

Optimizations

In Chapter 4, new algorithms were presented which were found to be fair over the entire loading range. This was true for the workload and parameters of Chapter 3. However, they achieved fairness at the expense of a significant increase in mean response times. For GATE scheduling, the increase was significant only for light loads. However, for STOP and DYNAMIC schedulers, there was an increase for the entire range of loading, the amount of increase being more for low arrival rates. This situation can be described as follows: There exist unfair algorithms (classical ones), which have different cylinder mean response times for different cylinders. Fairness in the new algorithms is achieved, not by redistributing the cylinder mean response times, but by increasing them *all* beyond the highest mean cylinder response time of the unfair algorithm.

To address this problem, improved algorithms are constructed for each scheduling granularity. These algorithms are **PLOOK**, **3SPL** and **PCLOOK** for GATE, STOP and DYNAMIC granularities respectively. They reduce the mean response time at the expense of a slight deterioration in fairness values. However, the fairness values still remain satisfactory (less than 0.05) for the entire range of loading. We describe each of these algorithms below and discuss its performance in comparison with the ones proposed in Chapter 4. The workload and parameters used here are the same as that used in Chapter 4.

5.1 GATE Scheduling

The improved algorithm for GATE scheduling granularity is **Park LOOK (PLOOK)**. It is similar to PPOLL except that the disk head parks at the last cylinder in the current sweep direction *only* if there are no requests in the system. Otherwise the new group is served as follows: The disk head reverses its sweep direction and starts serving the new group by moving to the first request in the group corresponding to the current sweep direction.

5.1.1 Results

The results of the experiments evaluating the performance of PLOOK is shown in Figures 5.1a through 5.1*l*. Plots for PPOLL is also present for comparison. Apart from these we have also plotted FCFS and CLOOK. Other algorithms are not plotted to maintain graph clarity. The performance characteristics of PLOOK is compared with that of PPOLL. From the figures, we observe that the PLOOK scheduler reduces the response times for the entire range of loads, the reduction being more significant for light loads (Figure 5.1e). Though the fairness values for these loads are slightly deteriorated, they still remain less than 0.05.

For the PLOOK scheduler, the cylinder response time distribution is nearly flat for low arrival rates (Figure 5.1a, 5.1b) but for a slight bias towards the central cylinders. This is due to the following reason: After serving the current group, the disk head parks only if there are no more requests in the system. This happens for very low arrival rates, and causes the average seek for all cylinders to be $N/2$ as explained for PPOLL. For slightly heavier loads (Figure 5.1b), after serving a group, there is typically a small queue of requests in the newly formed group (often containing only one request). In such cases the head moves directly to the single request in the new group and hence behaves like FCFS. This is unlike the case in PPOLL where the disk head parks at the end of serving *each* group. This causes slight deterioration of fairness values at medium arrival rates and reduction in the seek times due to which the mean response times improve. Under heavy loads, the algorithm degenerates to LOOK which is fair under these loads as explained in

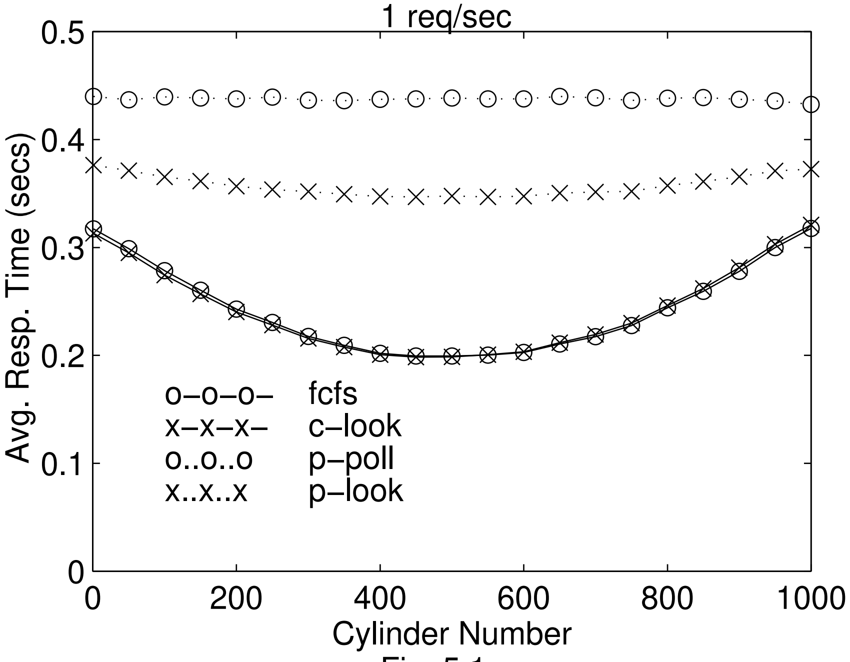


Fig. 5.1a

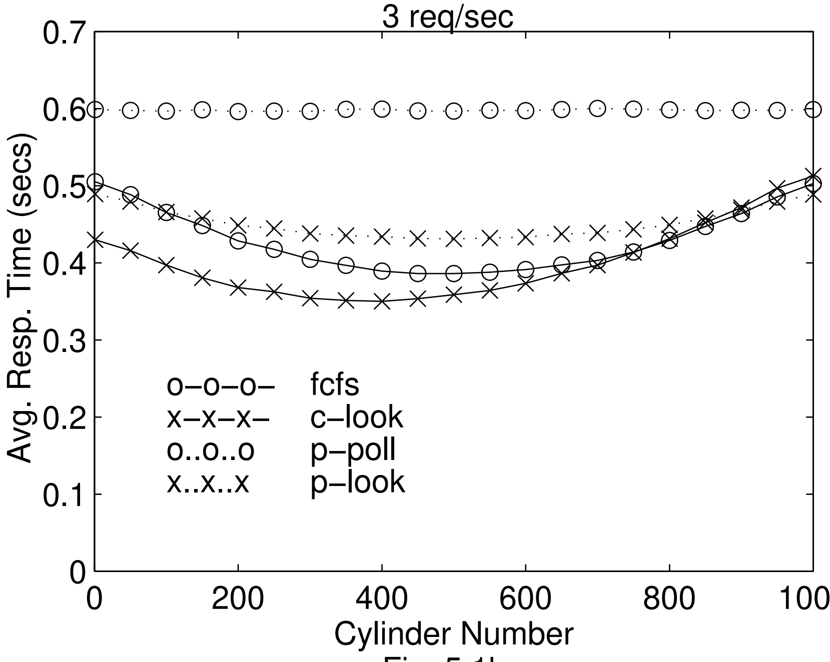


Fig. 5.1b

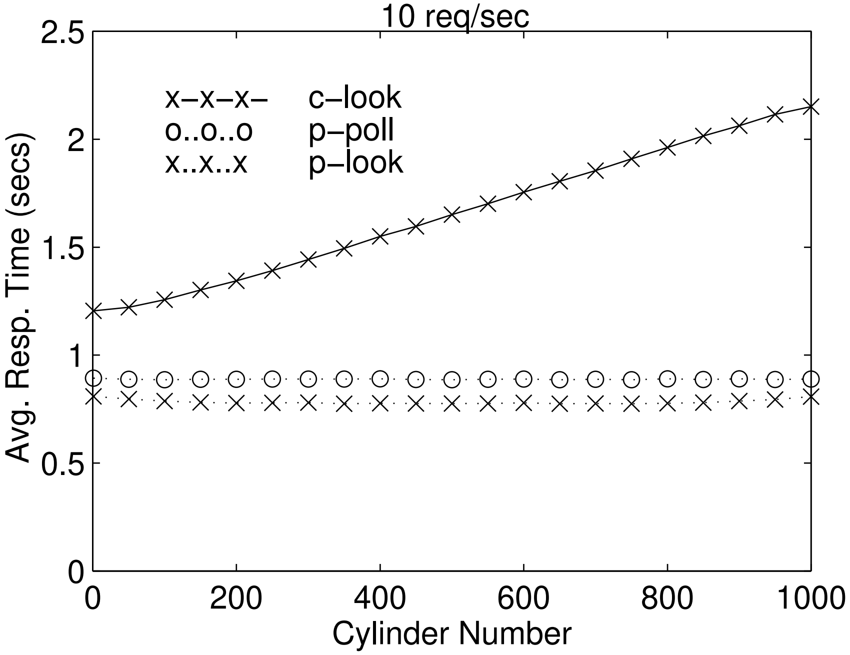


Fig. 5.1c

GATE Algorithms

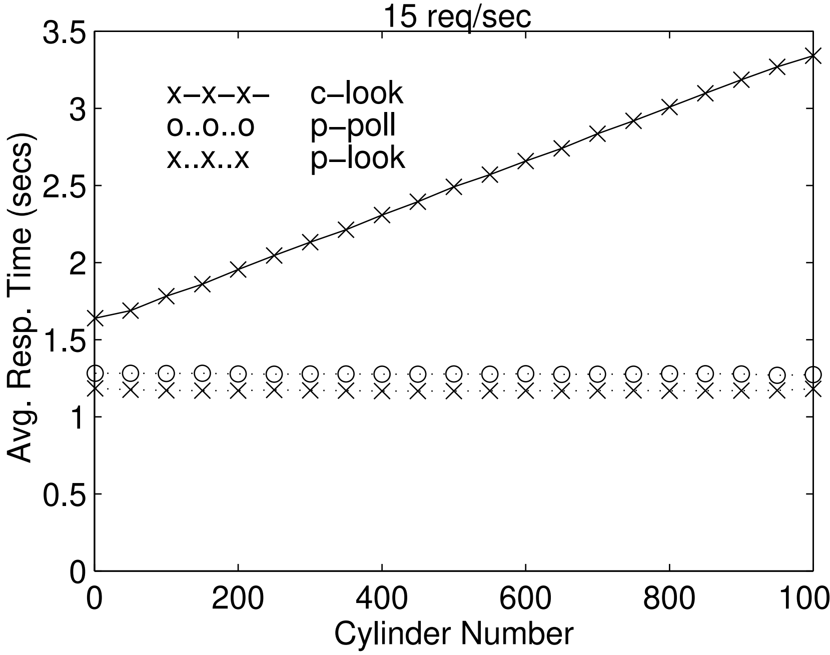


Fig. 5.1d

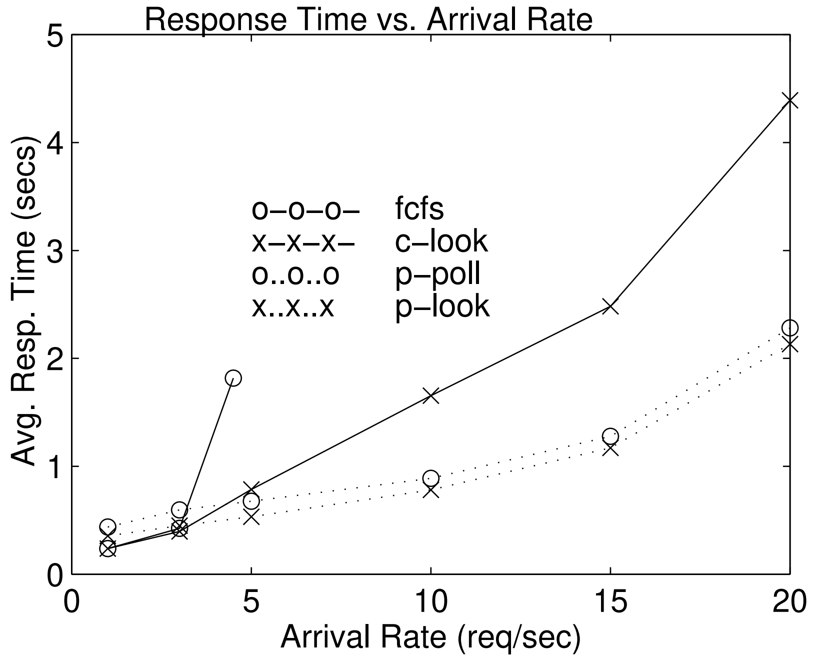


Fig. 5.1e

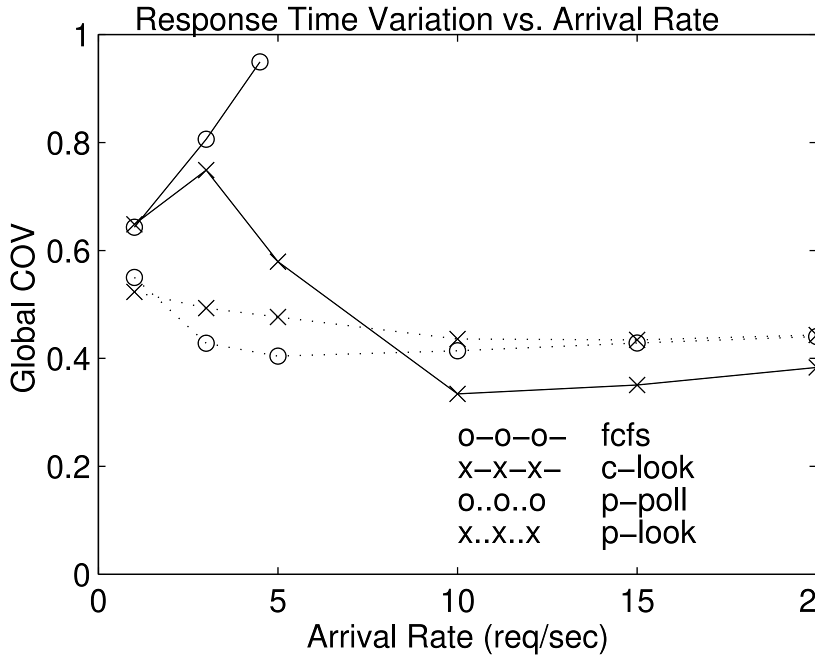


Fig. 5.1f

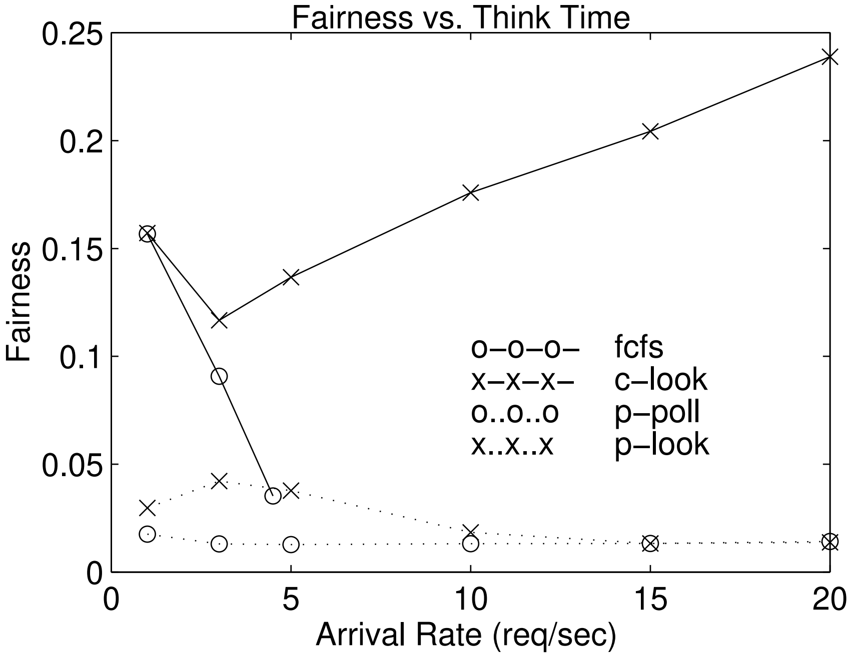


Fig. 5.1g

GATE Algorithms

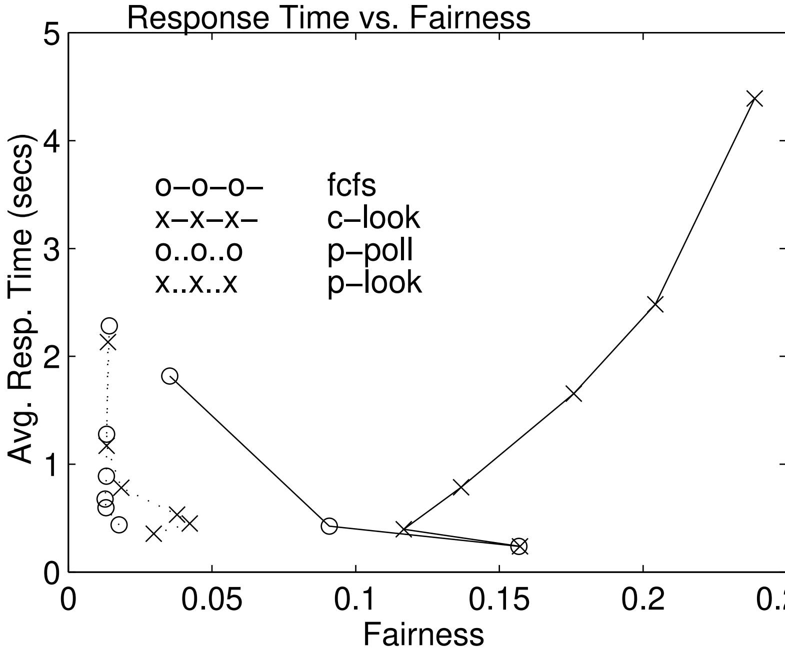


Fig. 5.1h

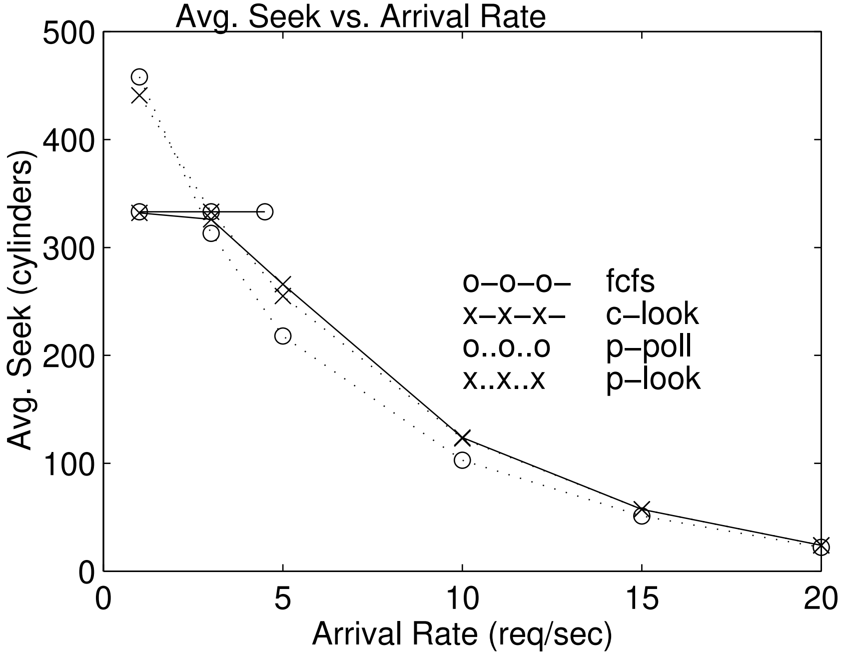


Fig. 5.1i

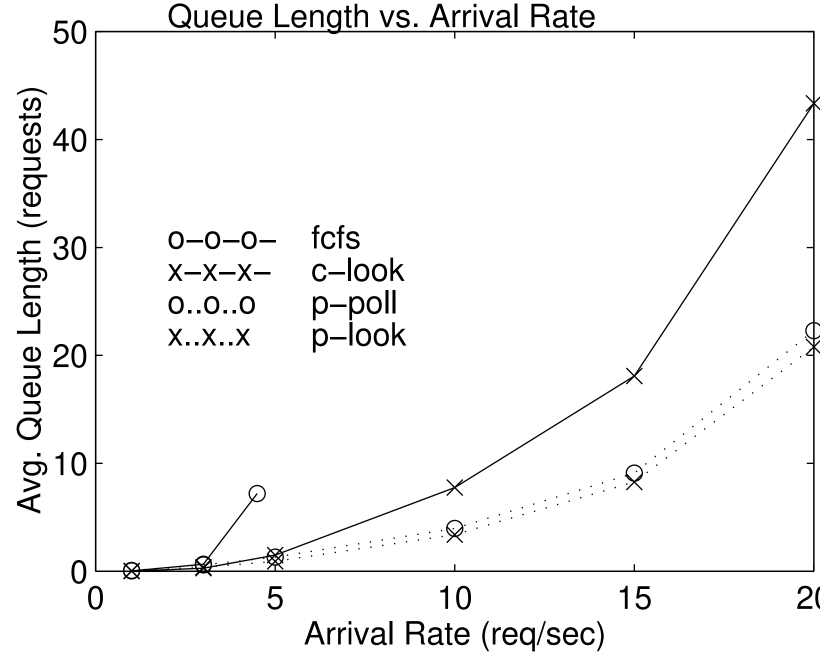


Fig. 5.1j

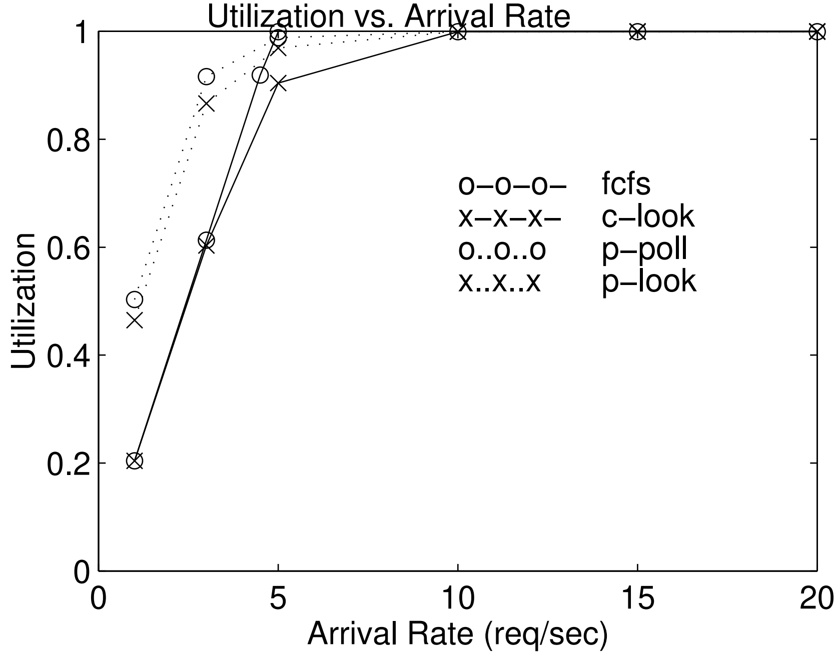


Fig. 5.1k

GATE Algorithms

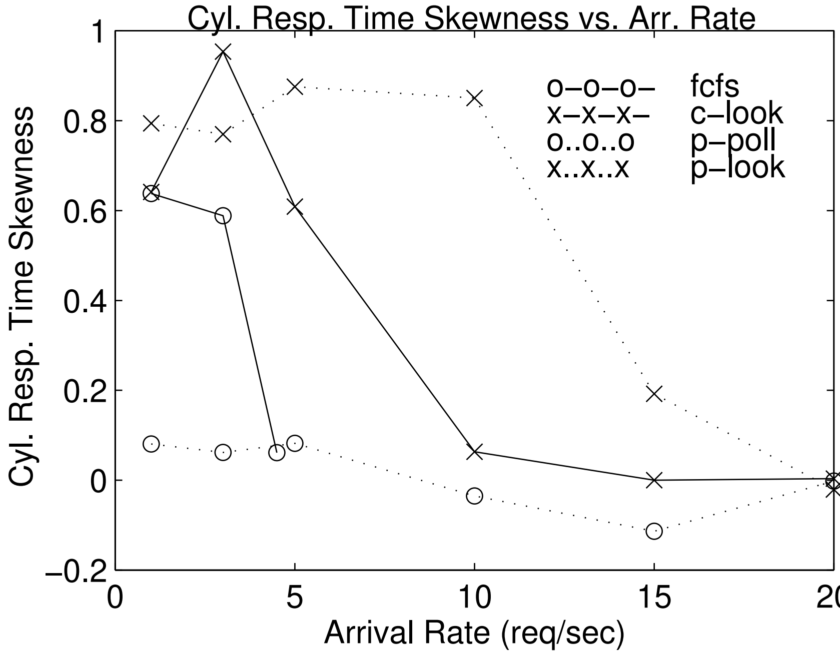


Fig. 5.1l

Section 4.1.

5.2 STOP Scheduling

The improved algorithm for STOP scheduling granularity is **3 Special Points (3SPL)**. It is described as follows: There are three special points defined, the two extreme cylinders and the central cylinder. It is identical to PCPL except for the following difference: Unlike in PCPL, where the disk head rewinds to the extreme cylinder when there are no further requests in the current sweep direction, here it rewinds to the nearest special point under such conditions. Due to this, the scheduling decision is not delayed as much as in PCPL.

5.2.1 Results

The results of the experiments evaluating the performance of 3SPL is shown in Figures 5.2a through 5.2*l*. Plots for PCPL, FCFS and CLOOK are also present for comparison. The performance characteristics of 3SPL is compared with that of PCPL. From the figures, we observe that the 3SPL scheduler improves the response times as well as the fairness values for light loads. For heavy loads there is a slight improvement of response times but this is at the expense of a marginal deterioration in the fairness value (Figure 5.2e, 5.2h).

The improvement in response times is due to the following reason: Unlike in PCPL, where the disk head always stroked back to the extreme cylinder, here it is stroked back to the nearest special point which is most often the central cylinder. This reduces the seek distance traveled by the disk head. There is also a decrease in response times for central cylinders resulting in flatter response times curves for light and medium loads. This can be explained as follows: When the sweep direction is inside–out, the inner cylinder requests which arrive after the scheduling decision is made at the central special point , are served only in the next sweep. Similarly, the outer cylinder requests suffer when the sweep direction is outside–in. The central cylinders are not affected either way and hence have a lower response time.

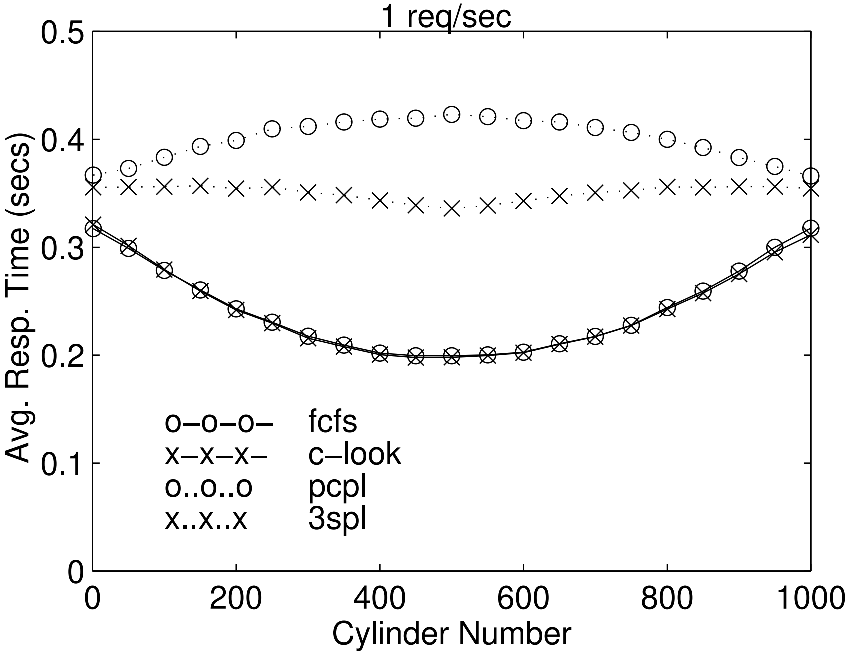


Fig. 5.2a

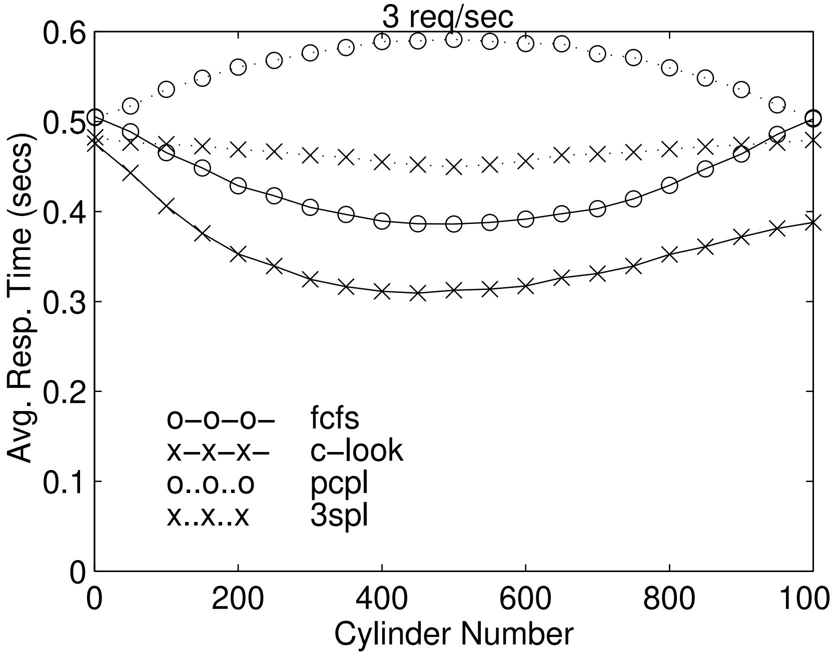


Fig. 5.2b

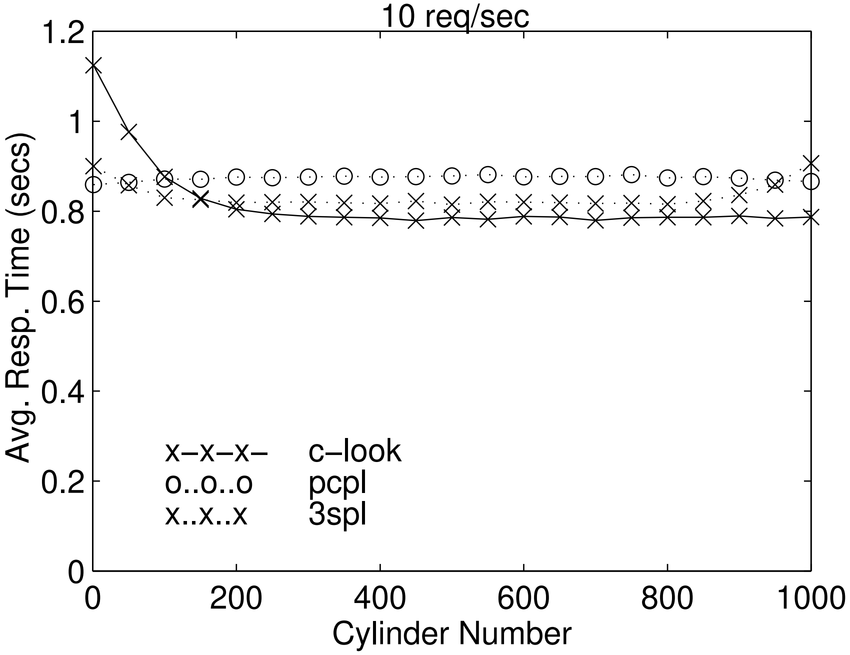


Fig. 5.2c

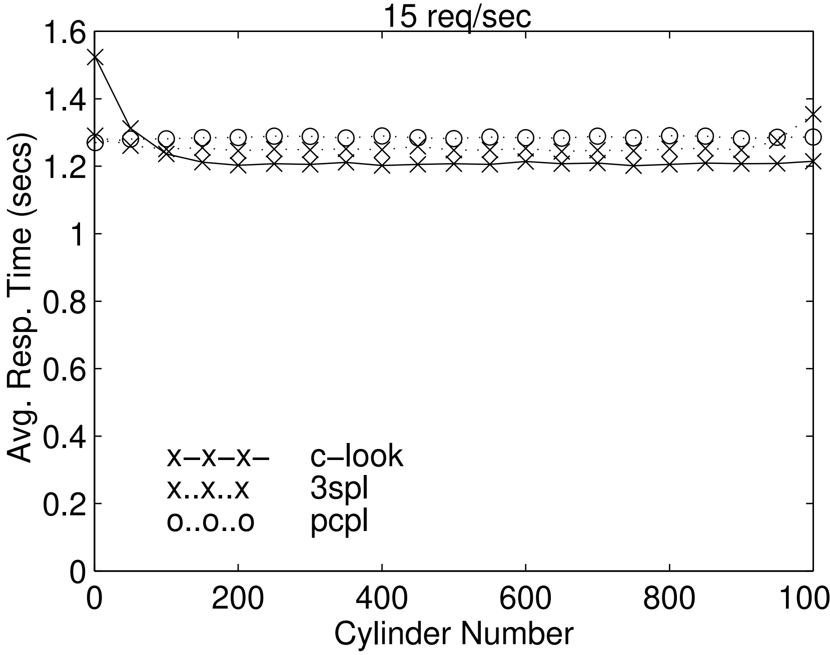


Fig. 5.2d

STOP Algorithms

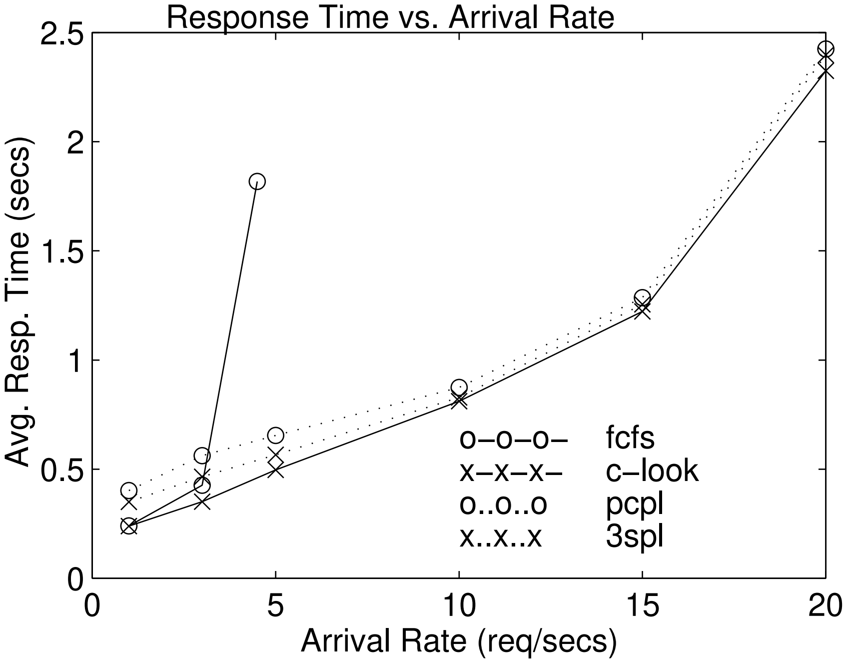


Fig. 5.2e

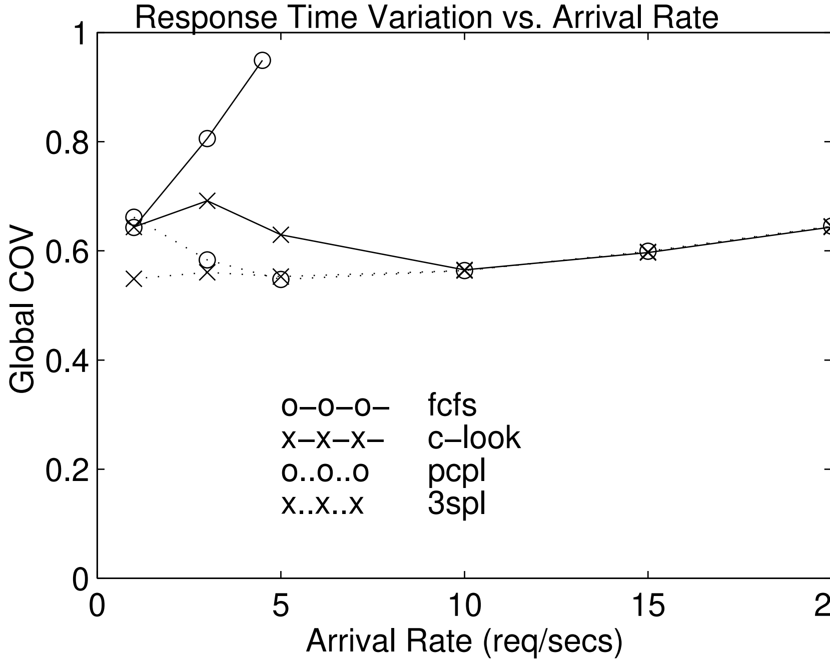


Fig. 5.2f

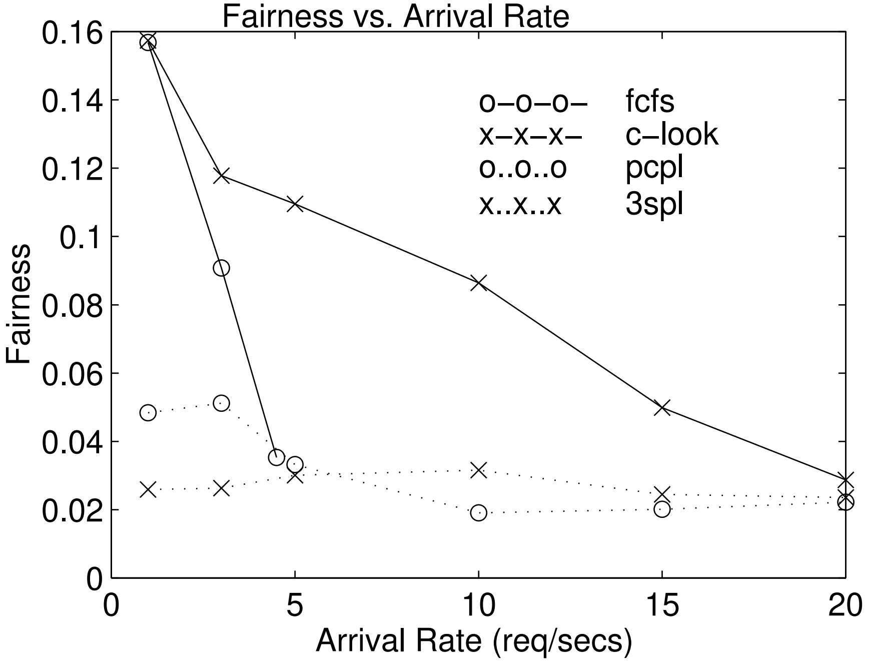


Fig. 5.2g

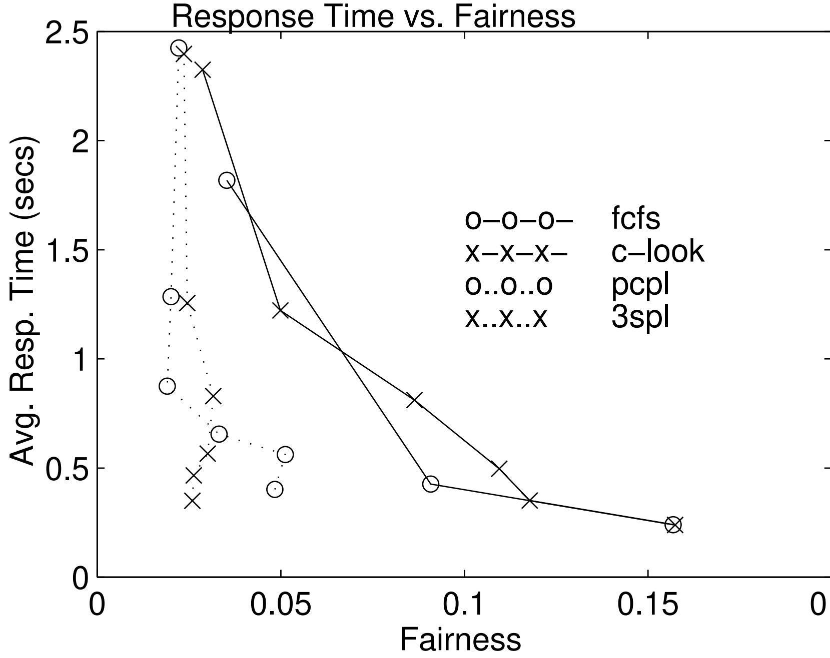


Fig. 5.2h

STOP Algorithms

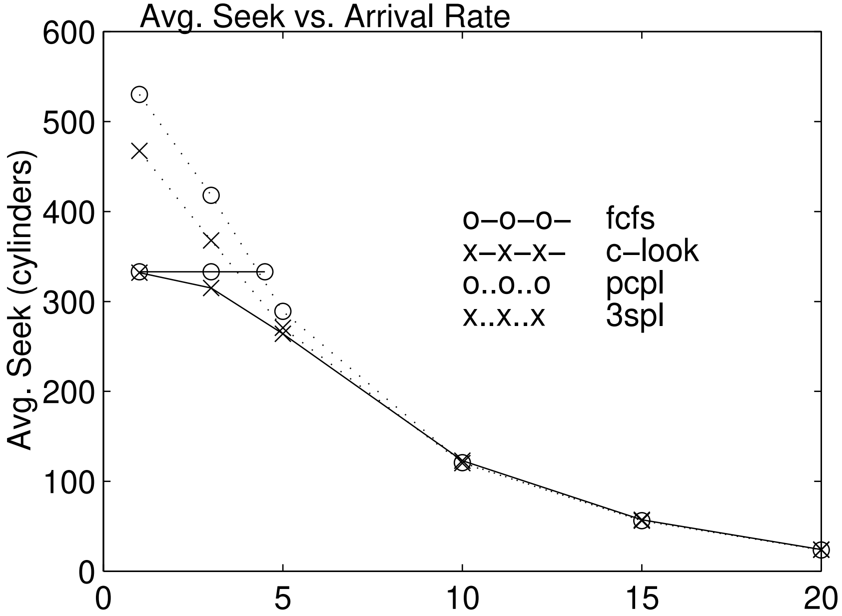


Fig. 5.2i

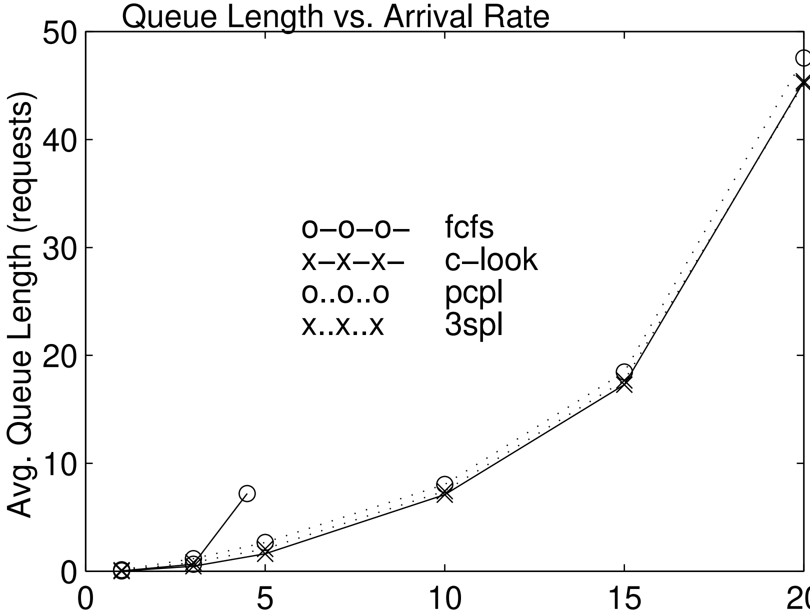


Fig. 5.2j

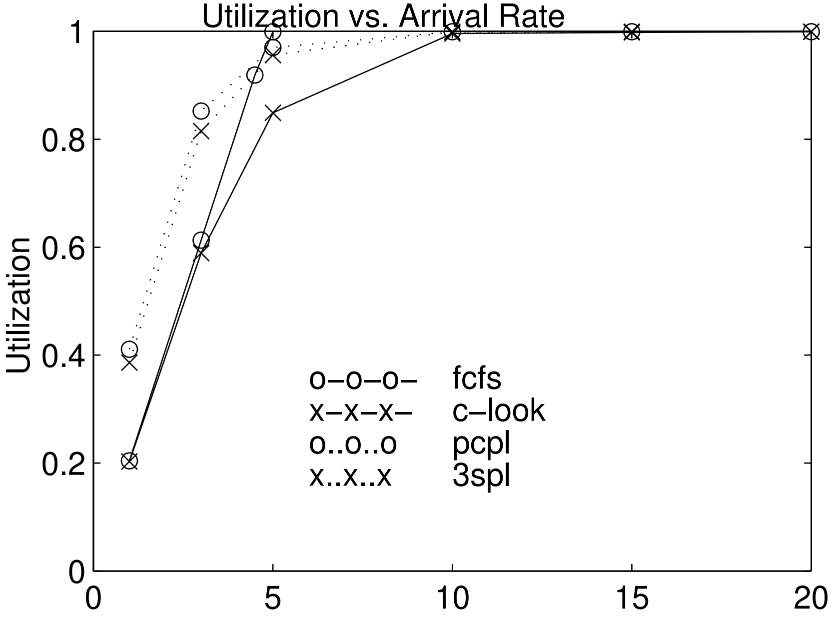


Fig. 5.2k

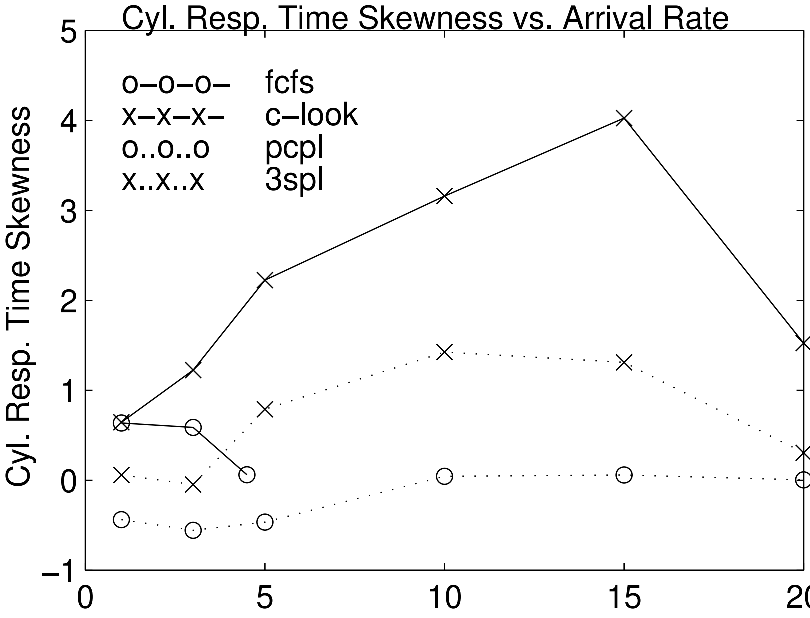


Fig. 5.2l

STOP Algorithms

5.3 DYNAMIC Scheduling

The improved algorithm for DYNAMIC scheduling granularity is **Park CLOOK (PCLOOK)**. It is the similar to CLOOK except for the following differences: The disk head parks at the nearest extremity when the system is emptied of requests. The decision to park is not changed as the disk head moves towards the parking spot, i.e. no requests are served on the way to the parking spot. The *first* request served after an idle period, is chosen before the head starts to move and this decision is not altered as the head moves towards it. If the head is parked at the inner end, the service direction for further requests is inside–out. However, if the head is parked at the outer end, the service direction for further requests is outside–in. Therefore, although the service is *always* unidirectional, the direction of the sweep may change dynamically.

5.3.1 Results

The results of the experiments evaluating the performance of PCLOOK is shown in Figures 5.3a through 5.3*l*. Plots for NSCP, FCFS and CLOOK are also present for comparison. The performance characteristics of PCLOOK and NSCP are compared. From the figures, we observe that the PCLOOK scheduler improves the response times for the entire range of loads (Figure 5.3e). There is a slight deterioration in the fairness values for lighter loads (Figure 5.3h).

Under light loads, the PCLOOK scheduler favours the central cylinders slightly (Figure 5.3a, 5.3b). The response time characteristics for these loads are as a result of two factors. Firstly, after the service of a request, if there are no further requests in the system, the head parks at one of the extremities. Due to this, the average seek distance for all cylinders is $N/2$. Secondly, after the service of a request, if there is a request pending (at low arrival rates, this will typically be the only request in the queue), the disk head moves straight to serve this request. Due to this, the average seek distance for the central cylinders is $N/4$ as compared to $N/2$ for the extreme cylinders. These two effects result in a slight bias towards the central cylinders under light loads. Under heavy loads, the algorithm behaves like CLOOK which is fair (Section 4.3). Note that under heavy loads, NSCP

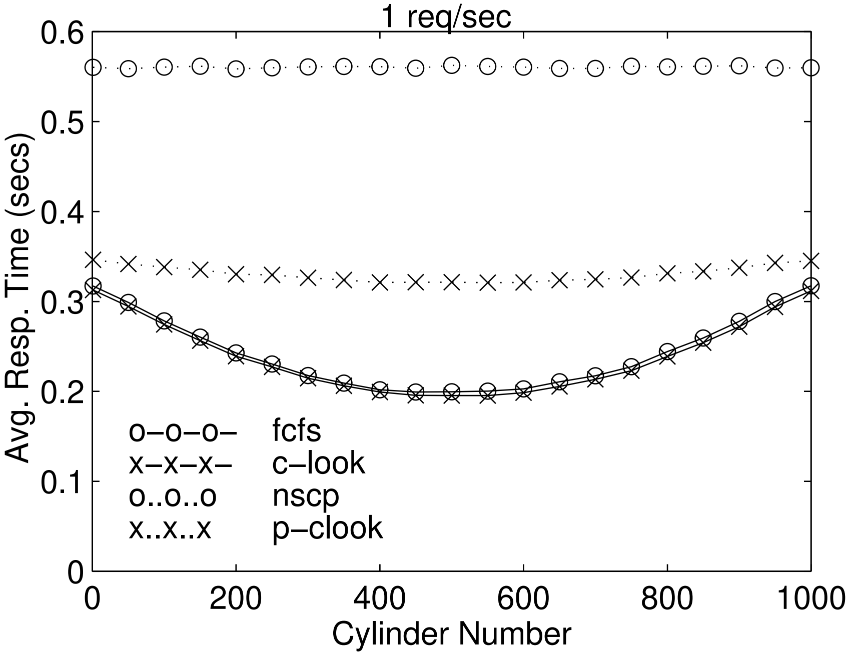


Fig. 5.3a

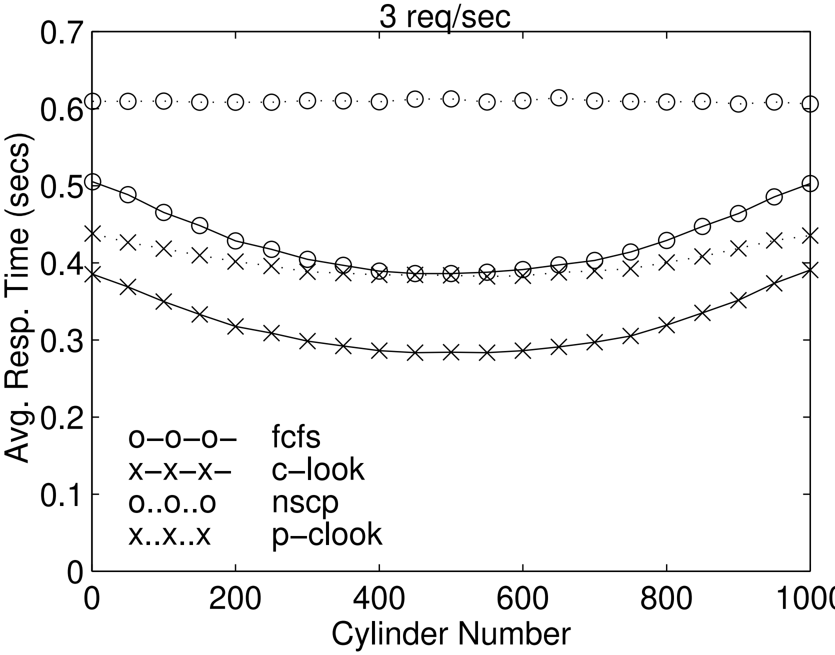


Fig. 5.3b

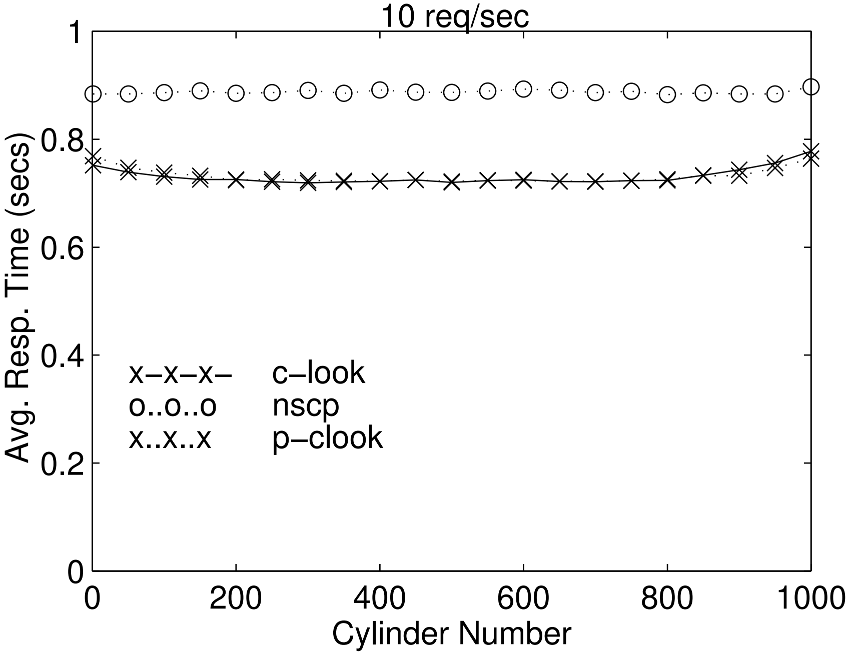
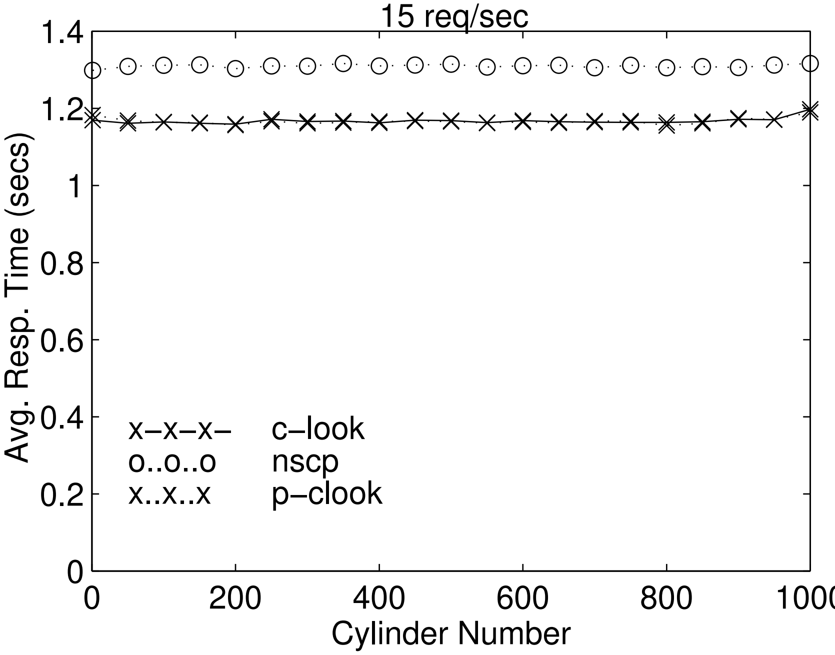


Fig. 5.3c



DYNAMIC Algorithms

Fig. 5.3d

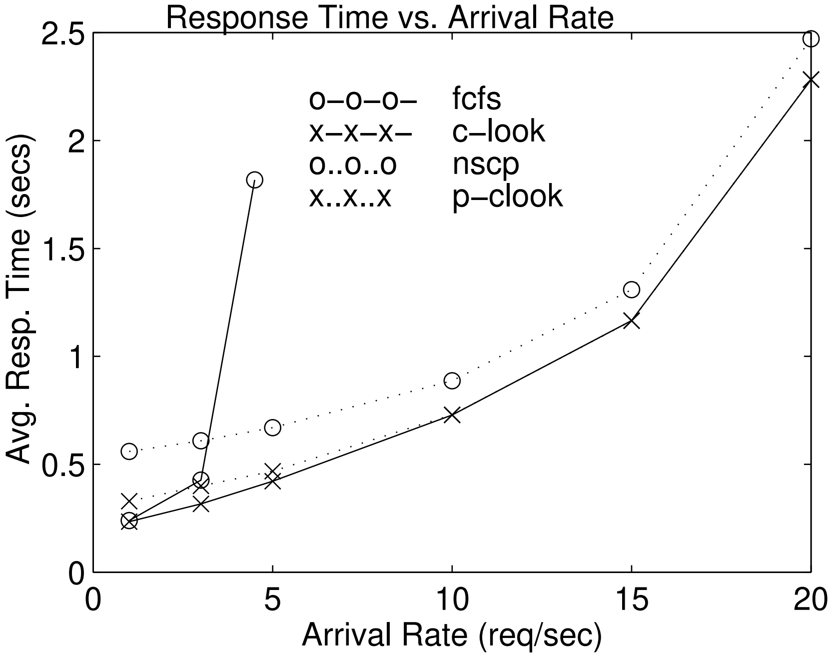


Fig. 5.3e

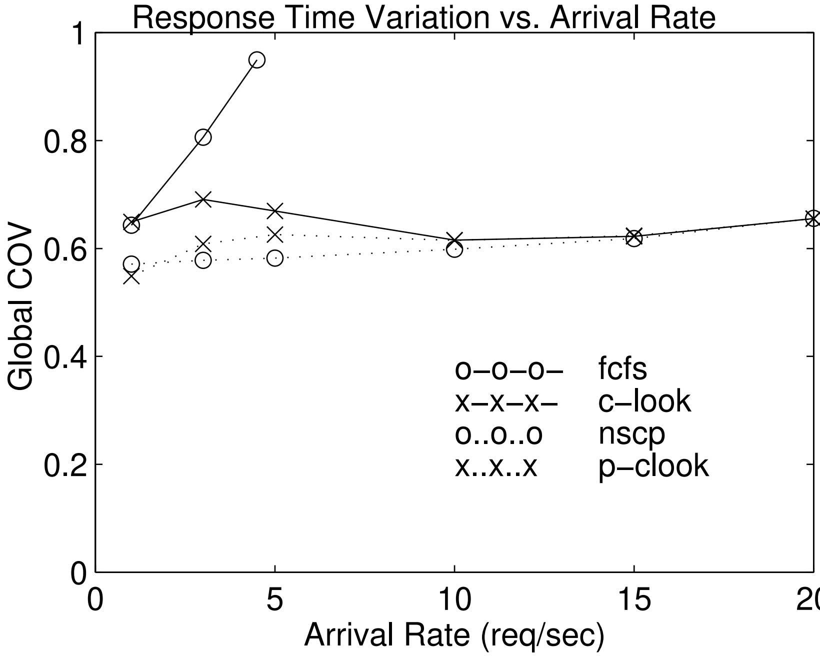


Fig. 5.3f

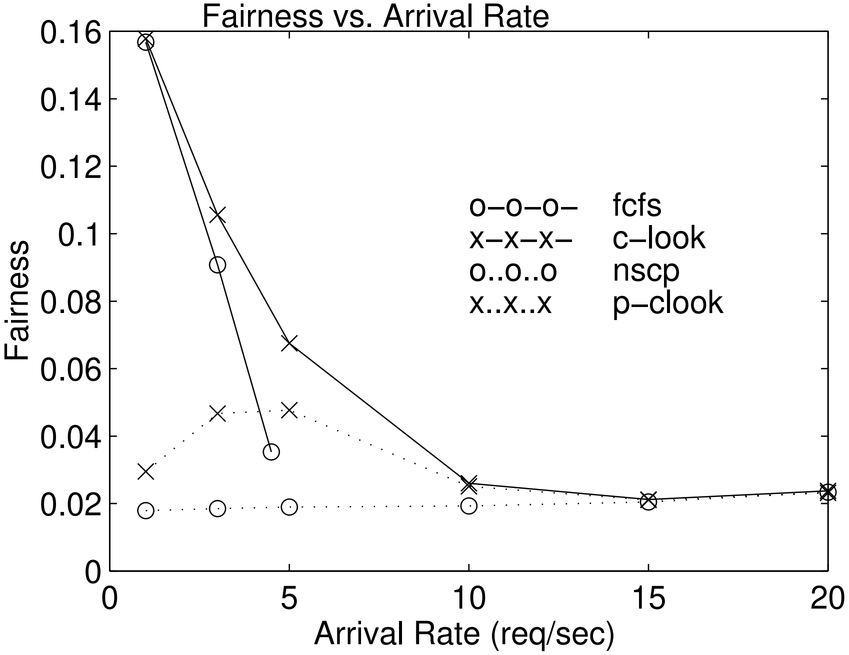


Fig. 5.3g

DYNAMIC Algorithms

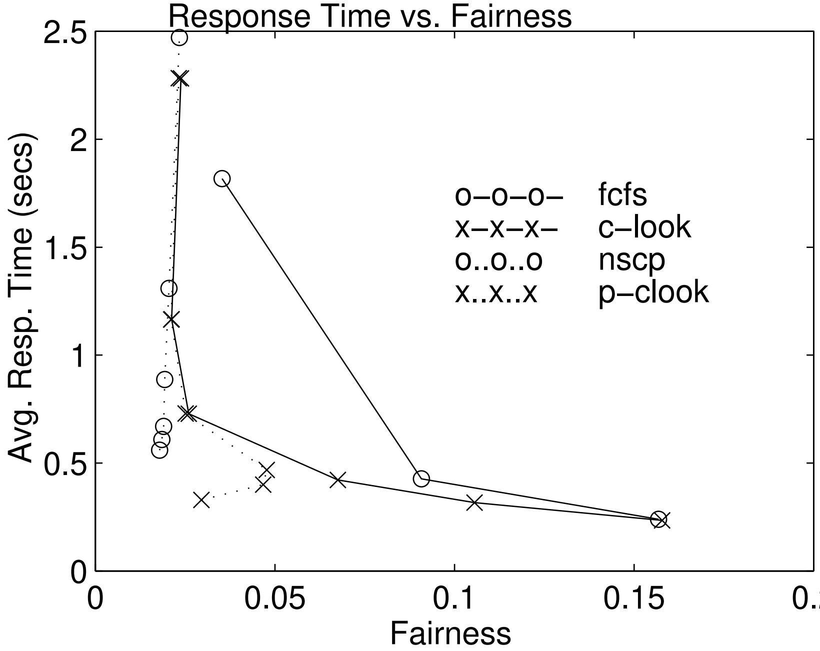


Fig. 5.3h

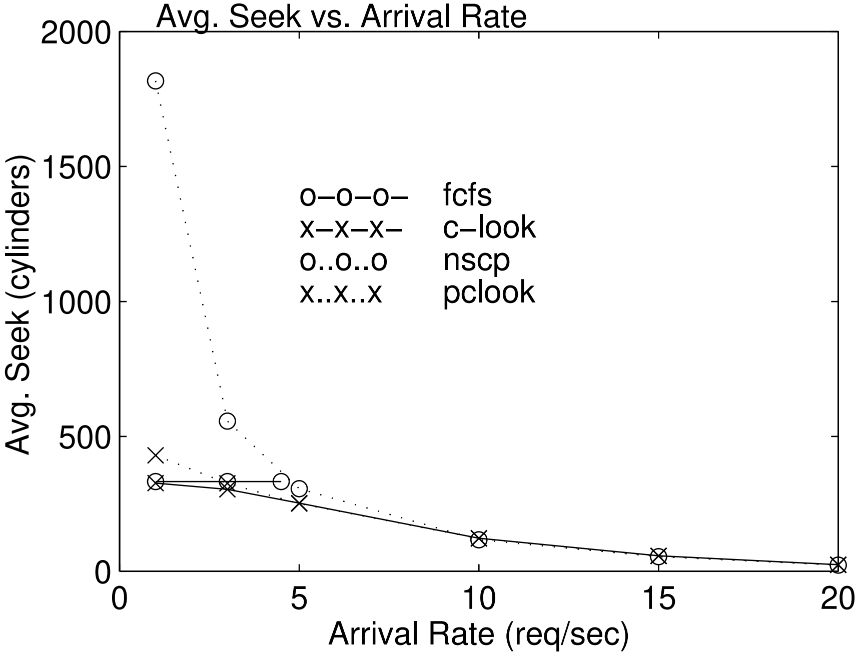


Fig. 5.3i

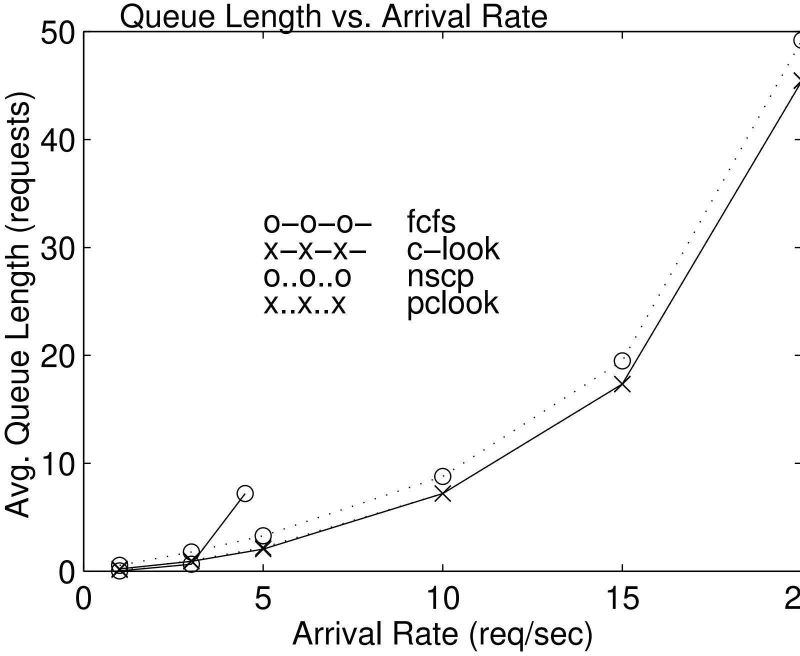


Fig. 5.3j

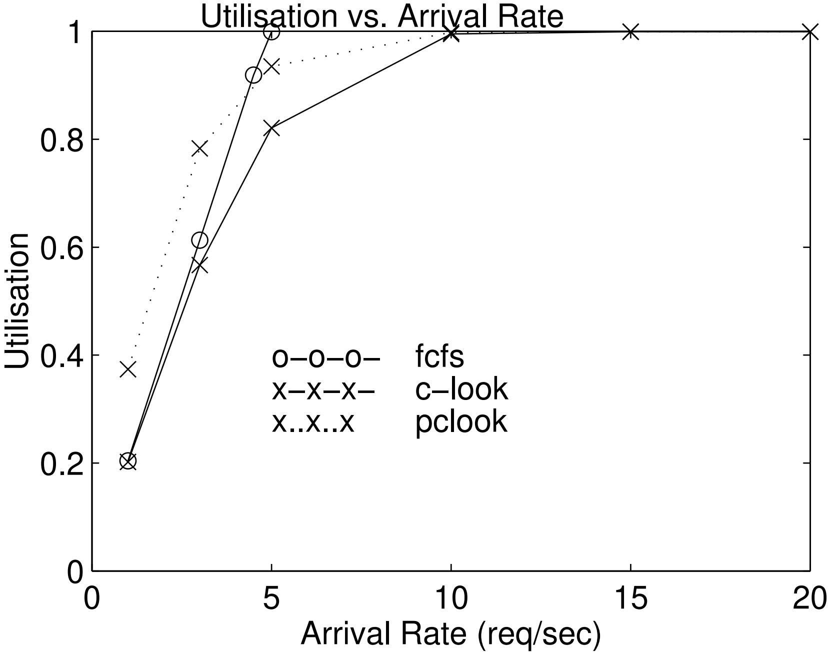


Fig. 5.3k

DYNAMIC Algorithms

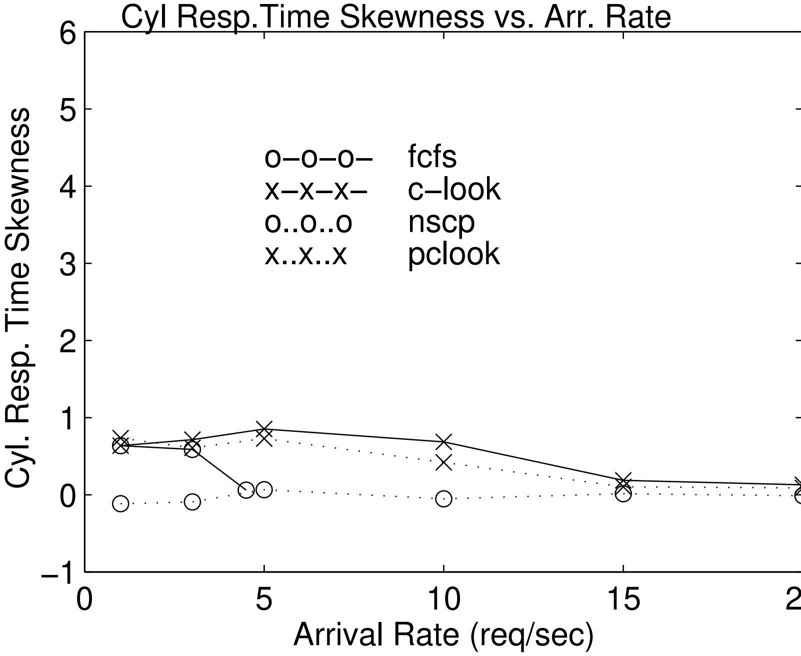


Fig. 5.3l

behaved like CPOLL which has a higher mean response time than CLOOK.

Chapter 6

Inherent Difficulties

In Chapter 5, improved algorithms for each scheduling granularity were presented and it was seen that these algorithms reduced the mean response time, keeping the fairness values satisfactory. However, this is not good enough. We would like to have an *ideal* fair algorithm for each granularity. This can be explained as follows: Consider an algorithm $\mathcal{S}$, which minimizes the overall mean response time. Most probably, this would correspond to one of the classical algorithms with a cup-shaped cylinder response time distribution as shown in Figure 6.1. An *ideal* fair algorithm, $\mathcal{I}$, will be that which is fair and has the same mean response time as that of $\mathcal{S}$. This is pictorially shown in Figure 6.1. Here, $\mathcal{I}$ achieves fairness by 'taking from the rich and giving it to the poor', i.e. redistributing the cylinder mean response times. We now discuss whether or not such an ideal fair algorithm is feasible. For the discussion below, the data requests are assumed to be uniformly distributed across the entire disk surface.

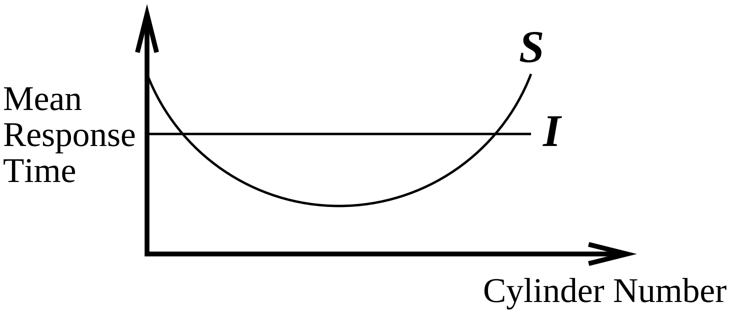


Figure 6.1: An Ideal Fair Disk Scheduler

6.1 Is an Ideal Fair Algorithm Feasible ?

Let us consider the system under light and heavy loads separately:

Case 1 - Light Loads

Let the arrival rates be such that new arrivals typically find the system emptied of requests. In most classical algorithms, the disk head would be found idle at the position of the previously served request. Due to this, all these algorithms would behave like FCFS. Since the data request distribution is uniform, from Section 4.1, we know that, the average seek for the extreme cylinders will be $N/2$ whereas that for the central cylinders will be $N/4$. The mean seek over all cylinders will be $N/3$. An ideal fair algorithm should make the average seek for *all* cylinders to be $N/3$. This will involve reducing the average seek for the extreme cylinders from $N/2$ to $N/3$. This is not possible as explained below.

Consider any algorithm in which the disk head remains at the position of the previously served request when idle. As explained before, these algorithms will not be fair. To reduce the average seek for any cylinder, the disk head should perform an anticipatory seek when it is idle. Due to this, the cylinders which are close to the parking spot will have smaller seeks compared to those which are not. Since we want to reduce the average seek for the extreme cylinders, the parking spot should be chosen close to the extreme cylinders. A single parking spot, close to one extremity will cause the average seek for the other extremity to become higher. Hence, to reduce the average seeks for both the extremities, there should be multiple parking spots symmetrically positioned about the centre. Also they should be used equally for parking. Since nothing is known about future requests, parking spots cannot be chosen to optimize the seek to the next request. Hence if the head is idle, the parking spot can be chosen randomly. Since the parking spots are symmetrically placed and used equally, the average seek for extreme cylinders will be $N/2$. This is because requests will typically find the disk head idle at one of the parking spots. This shows that the average seek for the extreme cylinders cannot be reduced beyond $N/2$. Therefore, under such light loads, the *optimal* fair algorithm is the one that makes the average seek for all cylinders $N/2$. This corresponds to the response time of the extreme cylinders in the FCFS case. Hence the mean cylinder response time of all cylinders, for the optimal fair algorithm, will be equal to that of the extreme cylinders in FCFS. As seen in Figures 5.1a, 5.2a and 5.3a, the improved algorithms come close to achieving this bound.

Case 2 - Heavy Loads

Let the arrival rate be such that the disk head is always busy serving requests. We look at two cases:

Case 2.1 - GATE Scheduling

The algorithms of GATE scheduling granularity serve requests in groups. Let us assume that algorithm $\mathcal{A}$ is the optimal fair algorithm for this granularity. We shall find an expression for the mean response time for this algorithm. Let T_A be the mean time taken for scheduler $\mathcal{A}$ to serve a group. Also, let C be the mean number of cylinders traversed by the disk head after it completes serving a group and before it forms the next group. We assuma that the head begins to serve the new group as soon as it is formed. Let $w_{gA}(x)$ be the mean time for the disk head to come to cylinder x after a new group has been formed. Then, the cylinder mean waiting time is given by

$$W_A = \frac{T_A}{2} + w_{gA}(x) + C\beta \tag{6.1}$$

where the first term corresponds to the mean waiting time while the previous group is being served. Since the algorithm is fair, W_A is independent of the cylinder number. Hence $w_{gA}(x)$ is also independent of x. So, w_{gA} will be equal to the mean cylinder waiting time before the disk head comes to a cylinder. Therefore,

$$w_{gA} = \frac{T_A}{2} \tag{6.2}$$

Substituting in 6.1, we get,

$$W_A = T_A + C\beta \tag{6.3}$$

Since algorithm $\mathcal{A}$ is the optimal fair algorithm, it minimizes W_A. C can be minimized to 0 if a new group is formed as soon as the previous group completes service. To minimize

T_A, a group of requests is served optimally so that it involves minimum seek. This can be achieved if the requests are served as the disk head sweeps the disk surface from one edge to the other. Hence, LOOK algorithm should minimize W_A. This is evident from Figure 5.1d. Since PLOOK degenerates to LOOK at such loads, it is the optimal fair algorithm for these loads.

Case 2.2 - STOP and DYNAMIC Scheduling

Any algorithm that does not sweep the disk surface unidirectionally while serving requests cannot be fair. This can be explained as follows: Let us assume that algorithm $\mathcal{A}$ is fair, but does not sweep the disk surface unidirectionally while serving requests. Since algorithm $\mathcal{A}$ is fair and the data requests are uniformly distributed, the mean inter-visit times of the head for all cylinders must be equal. As the disk head does not sweep the disk surface unidirectionally, the inter-visit times of the head for all cylinders will not be periodic. Also, the pattern of the inter-visit times will not be the same for all cylinders. This is because of the physical separation between the cylinders. Due to this, the cylinder mean response times will be higher for those cylinders which have a greater difference in the inter-visit times. This is because of the large queue that will build up during a long inter-visit time. Hence, algorithm $\mathcal{A}$ *cannot* have the same mean cylinder response time for all cylinders. This makes it unfair. Therefore, all fair algorithms must sweep the disk surface unidirectionally while serving requests.

Let algorithm $\mathcal{A}$ be fair and hence satisfy the above condition. The mean response times and hence the mean inter-visit times for all cylinders must be the same. Since the request distribution is uniform and algorithm $\mathcal{A}$ does not cause starvation for any cylinder requests, the following is true: On the average, between two consecutive visits to the same cylinder, the disk head serves all other $(N-1)$ cylinders. Also, the average number of cylinders traversed between two consecutive visits to the same cylinder is atleast $2N$. This is true because of the unidirectional service of requests. Let T_A be the mean cylinder inter-visit time. During this time the disk head travels a minimum distance of $2N$ cylinders and serves $(N-1)\lambda T_A/N$ requests. Since each requests requires A units of service time,

T_A can be written as

$$T_A \geq (N-1)\lambda T_A A/N + 2N\beta \tag{6.4}$$

This gives,

$$T_A > \frac{2N\beta}{1-\lambda A} \tag{6.5}$$

Let W_A be the cylinder mean waiting time (same for all cylinders). Then,

$$W_A = \frac{1}{2}[T_A + \lambda T_A A/N] \tag{6.6}$$

where the first term corresponds to the waiting time while the head serves other cylinders and the second term is due to the service of other requests for the same cylinder. Substituting from 6.5, we get,

$$W_A > \frac{N\beta}{1-\lambda A} \tag{6.7}$$

From [Coff73], we know that for POLL algorithm, the mean cylinder waiting time for the extreme cylinders is $N\beta/(1-\lambda A)$. Comparing with 6.7, we conclude that the mean waiting time for any fair algorithm cannot be less than that of the extreme cylinders in POLL algorithm. Hence, the mean response time of the optimal fair algorithm will be equal to that of the extreme cylinders in POLL algorithm. From Figures 5.2d and 5.3d, it is observed that the improved algorithms described in Chapter 5 (3SPL, PCLOOK) come close to acheiving it (Though the performance POLL is not shown in the plots, it behaves like LOOK at high arrival rates).

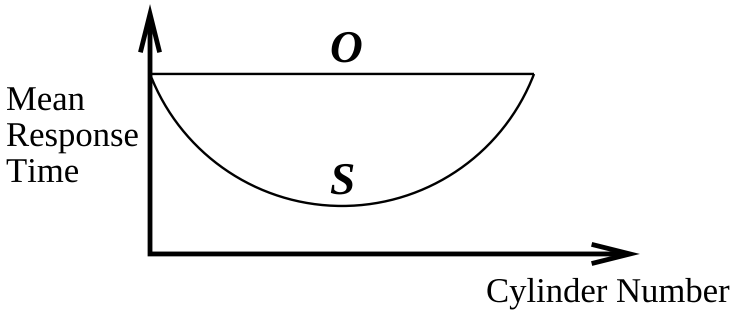


Figure 6.2: An Optimal Fair Disk Scheduler

From the discussion above, it is evident that fairness *cannot* be achieved by redistribution of the cylinder mean response times of an unfair algorithm which minimizes the overall mean response time. The *optimal* fair algorithm, $\mathcal{O}$, is the one whose mean response times are equal to that of extreme cylinders of algorithm $\mathcal{S}$. This is pictorially depicted in Figure 6.2.

6.2 Merits and Demerits

We have seen that the optimal fair algorithms are the ones that are satisfactorily fair and minimize the mean response times for all arrival rates. If these fair algorithms are compared with the classical ones, which are not fair, but have lower mean response times (e.g. SSTF, SCAN), it is observed that the response times for the extreme cylinders is nearly the same in both the cases whereas for the central cylinders, it is more for the fair algorithms. This is found to true for light as well as heavy loads. Thus, fairness is achieved by *increasing*, not redistributing the mean response times. In most applications, this kind of improvement over classical algorithms will not be of any advantage, because for all cylinders, either the response times are increased or remain the same. However, in some applications this might be useful. Consider the following situation: Suppose there is an application where the job is divided into several subtasks to be done parallelly, each requiring independent disk access. If the mean response times for all cylinders are not equal, some subtasks will complete before others. Also, assume that this is to be followed by a requirement of an exclusive resource (e.g. a printer), which will be used only when *all* subtasks are over. Now, the subtask that completes the earliest may lock this resource. But until all the subtasks get over, this resource will not be used. This results in wasting the resource's time as it is locked but not used till all the subtasks complete. Hence it is better for all subtasks to finish with the slowest one. This will be the case if either one of PLOOK, 3SPL or PCLOOK is implemented for scheduling requests. Applications where it is more advantageous for requests to wait in the input queue rather than the output one would prefer these algorithms. An analogy for this kind of a situation is that of "wafer fabrication" where it is preferable for the chips to wait before being processed, rather than later on, where it might get damaged, while waiting for other chips to complete their processing.

Chapter 7

Robustness

The simulations done till now used the simple model of Chapter 3. The model was open with Poisson stream of arrivals. The data requests were uniformly distributed over the entire disk surface and the disk parameters were similar to that in [Teor72]. This was done to aid comparison of performance results with previous, related work. In this chapter, we examine the robustness of the fair algorithms with respect to variations in the model, workload and the input parameters used in the experiments. We have varied the following :

- Disk parameters
- Data request distribution
- Simulation model
- Arrival process

We discuss each of these variations separately and their effect on the performance characteristics of the different algorithms.

7.1 Disk Parameters

All earlier experiments were performed using the disk parameters similar to the ones in [Teor72]. These values are somewhat outdated but were used to aid comparison of results with previous, related studies. We repeated the experiments with disk parameters of a recent disk, the $HPC2200A$ disk ([Ruem94]). The values of the disk constants for this disk are given in Table 7.1.

Parameter	Values
NumCyl	1449
Alpha	3.5 msec.
Beta	0.0236 msec./cylinder
Access	41.1 msec.

Table 7.1: HPC2200A Disk Parameters

The corresponding performance characteristics of the proposed algorithms of Chapter 5 (PLOOK, 3SPL and PCLOOK) remain qualitatively similar to those for the older disk parameters.

7.2 Data Request Distribution

In all the previous experiments, the data requests were uniformly distributed over the entire disk. This is not always true as very often the disk accesses are not independent and exhibit some kind of locality, resulting in a skewed distribution.

The "skew" in a distribution can be of two different kinds: "spatial" and "temporal". In spatial skew, the relative number of requests for every cylinder is not the same when a large number of requests are considered i.e. the requests show spatial locality. In "temporal" skew, if a large number of requests are considered, all cylinders are accessed equally, but in a small period of time the relative number of accesses are different for different cylinders. Here the requests exhibit temporal locality. We look at both these kinds of skews and the corresponding performance of the different algorithms. The spatially skewed distributions considered here are "single spike skew" and "double spike skew". As an example of temporally skewed distribution, we have considered the "moving gaussian" distribution.

7.2.1 Single Spike Skew

The distribution for Single Spike Skew is shown in Figure 7.1. This is an example of spatially skewed distribution. There is a peak at the inner cylinders which shows that they are accessed much more frequently than the rest of the disk. A practical example of this kind of distribution is a disk consisting of a single filesystem which has inodes located at these inner cylinders. The inodes are usually accessed many more times than the regular files.

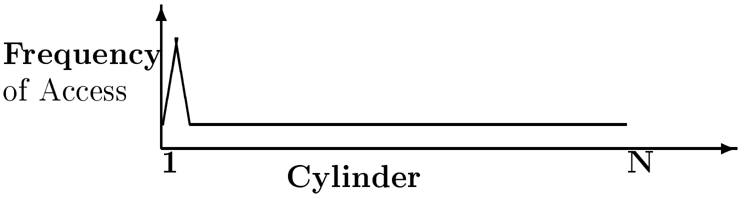


Figure 6.2: Single Spike Skewed Distribution

The algorithms of Chapter 4 and Chapter 5 are simulated with the following two differences:

- Any algorithm that parked the disk head at the closer disk extremity when idle, now parks alternately at the two extremities. This is done for the following reason: The basic idea behind parking at the disk ends was to have the same average seek distance for all cylinders, at light loads. For uniformly distributed requests both options give the same results (average seek distance of $N/2$ for all cylinders). However, for a skewed distribution as above the “closer edge policy” will lead to unequal average seeks across the cylinders. This is because of the following reason: Since most requests are for the inner cylinders, the head will park mostly at the inner end, thus resulting in smaller average seeks for the inner cylinders. To avoid this difference in average seek distance across cylinders, we use “alternate edge policy” which results in equating the average seek distance for all cylinders to $N/2$.
- “Exhaustive service” policy is used instead of “single service” policy. This is done

to reduce the average response times for those cylinders which are accessed more frequently than the rest. For uniform distribution, this modification would not cause any significant change in the results because the number of cylinders are much more than the mean queue length, even at high utilizations. This results in very few cylinders having more than one pending request.

We now look at the performance of the algorithms for each scheduling granularity separately.

GATE Scheduling

The results of the experiments evaluating the performance of the algorithms are shown in Figures 7.2a through 7.2h. From these figures, we observe that FCFS saturates earlier than the other algorithms. Also, it is unfair under light loads, favouring the inner cylinders that are close to the set of cylinders accessed more frequently. This is because under light loads a new arrival mostly finds the head idle at one of the peak cylinders. This reduces the average seek for cylinders close to the peak cylinders. We note that both CLOOK and LOOK behave like FCFS under light loads (Figure 7.2a). This is due to the fact that most of the groups consists of a single request.

The LOOK scheduler becomes fairer as the loading becomes heavier whereas CLOOK shows a distinct bias towards the inner peak cylinders. This is because of the undirectional service of groups which leads to higher waiting times for the outer cylinders. It is even higher than that for the uniform distribution case because typically in each group most of the requests are for the peak cylinders which are served before the requests for other cylinders.

Turning our attention to CPOLL, we notice a bias towards the outer cylinders for low arrival rates. At higher arrival rates, the bias is towards the inner cylinders. This behaviour is similar to that for uniform distribution with the difference that the increase/decrease of response times are no longer linear as before. This is due to the following reason: For light loads (Figure 7.2a), the outer cylinders have a lower response

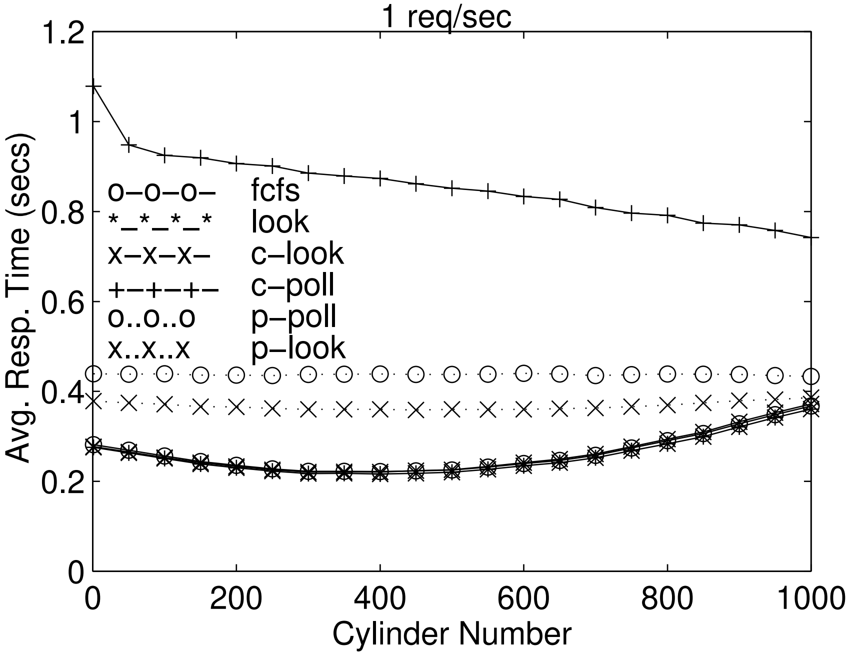


Fig. 7.2a

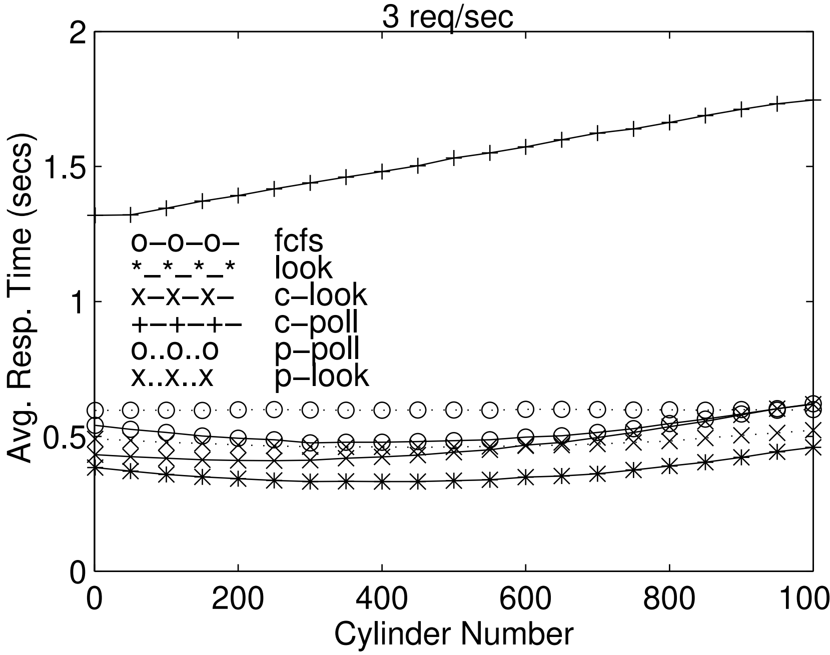


Fig. 7.2b

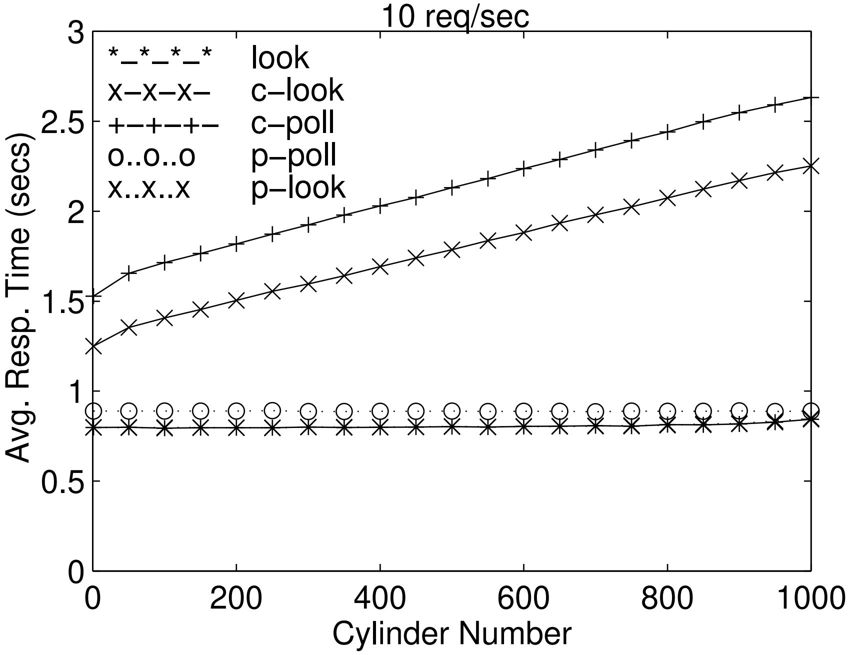


Fig. 7.2c

GATE Algorithms

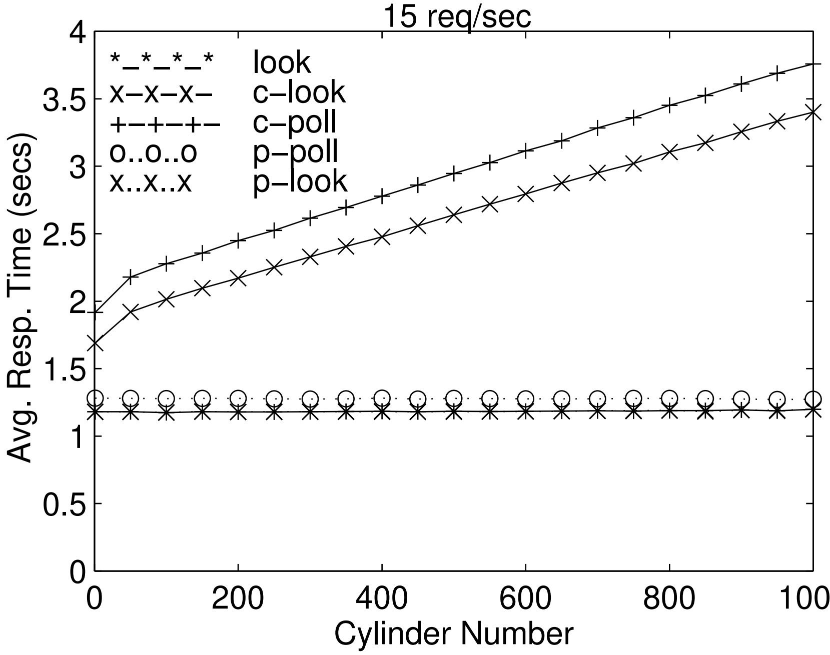


Fig. 7.2d

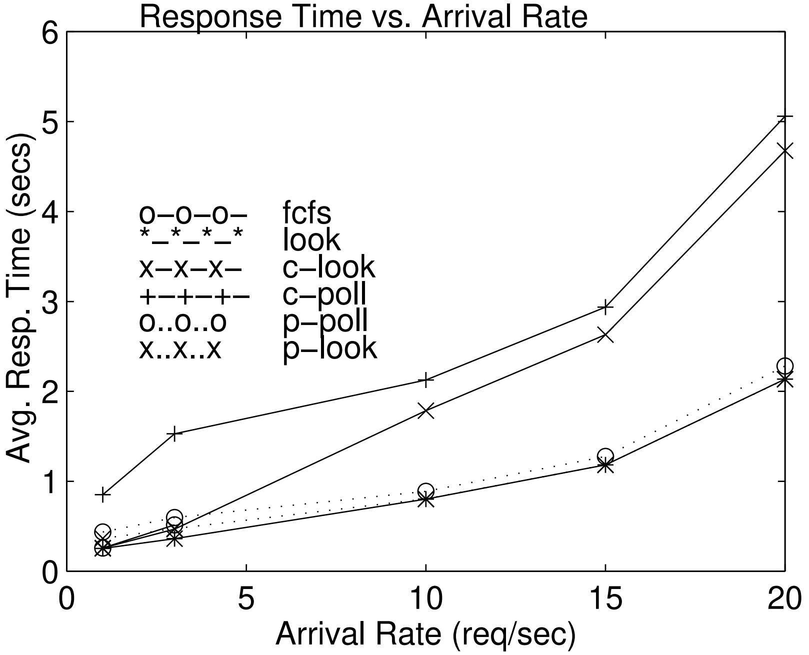


Fig. 7.2e

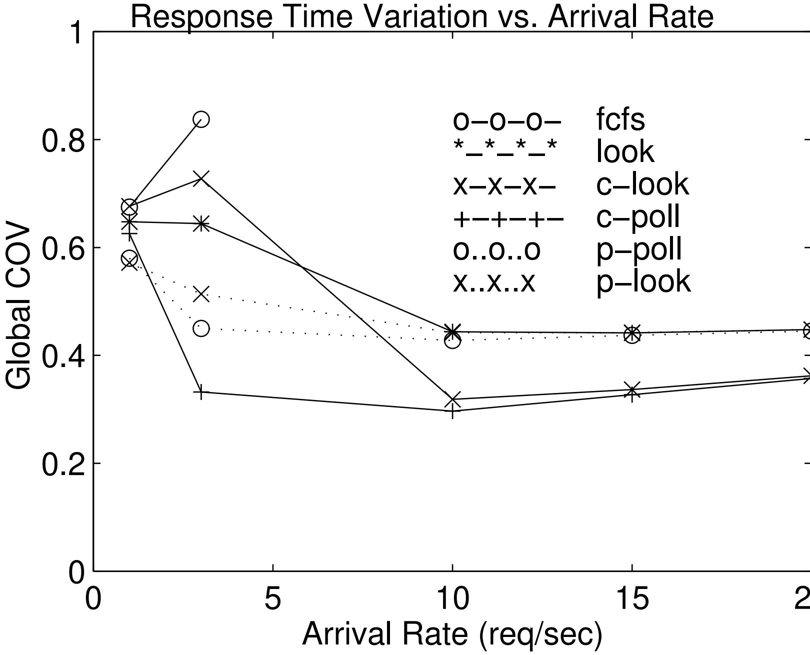


Fig. 7.2f

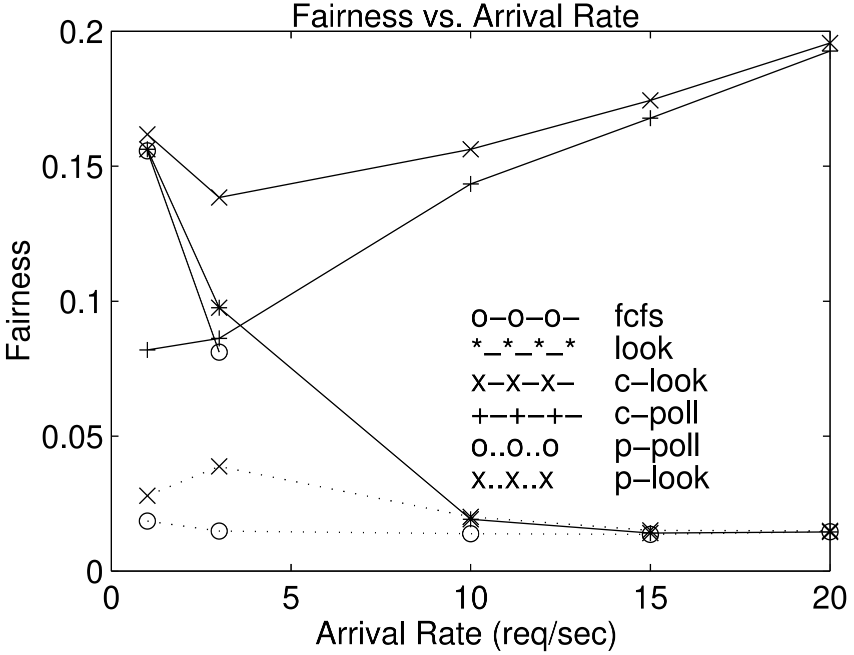


Fig. 7.2g

GATE Algorithms

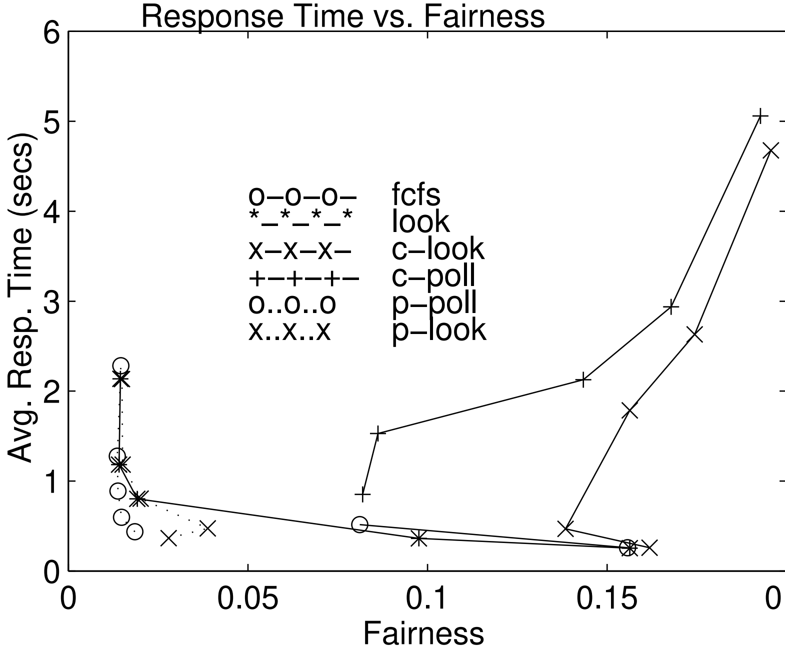


Fig. 7.2h

time because typically a new request finds the head idle at one of the peak cylinders, unlike *any* cylinder as in the uniform case. Hence, the requests for outer cylinders mostly do not have to wait for the disk head to rewind to the inner cylinders before getting served. For heavy loads, the outer cylinders have a higher response time for the same reasons as described above for CLOOK.

Moving on to PPOLL and PLOOK, we notice that the cylinder response time distribution remains nearly unchanged for the entire loading range. This is because of the parking policy, for low arrival rates and alternate sweeps in both directions, for high arrival rates, as explained in Section 4.1 and 5.1.

STOP Scheduling

The results of the experiments evaluating the performance of the algorithms are shown in Figures 7.3a through 7.3h. From these figures, we observe that as in the case of uniform data request distribution, all classical algorithms namely SSTF, LOOK and CLOOK behave essentially like FCFS at very low arrival rates (Figure 7.3a) which is unfair due to the reasons given for GATE scheduling.

Looking at SSTF, we observe that, under heavy loads, it is quite unfair favouring the central cylinders. Also, the inner spike cylinders have a lower mean response times as compared to the outer ones. This is because once the head comes to the inner extreme, where there are more number of requests waiting, it takes a long time before it goes and serves requests for the outer cylinders. This leads to higher waiting times for the outer cylinders.

Moving on to CLOOK, we observe that as the arrival rate increases, the response times for the inner cylinders increase. This is for similar reasons given for uniform distribution (Section 4.2). The LOOK scheduler is also unfair under heavy loads where the central cylinders receive better treatment than the extreme ones. This is due to the alternate long and short inter-visit times of the disk head, as explained in Section 4.2.

As in the uniform distribution case, both PCPL and 3SPL remain satisfactorily fair for the entire loading range. This is due to the parking policy, for lighter loads and delayed

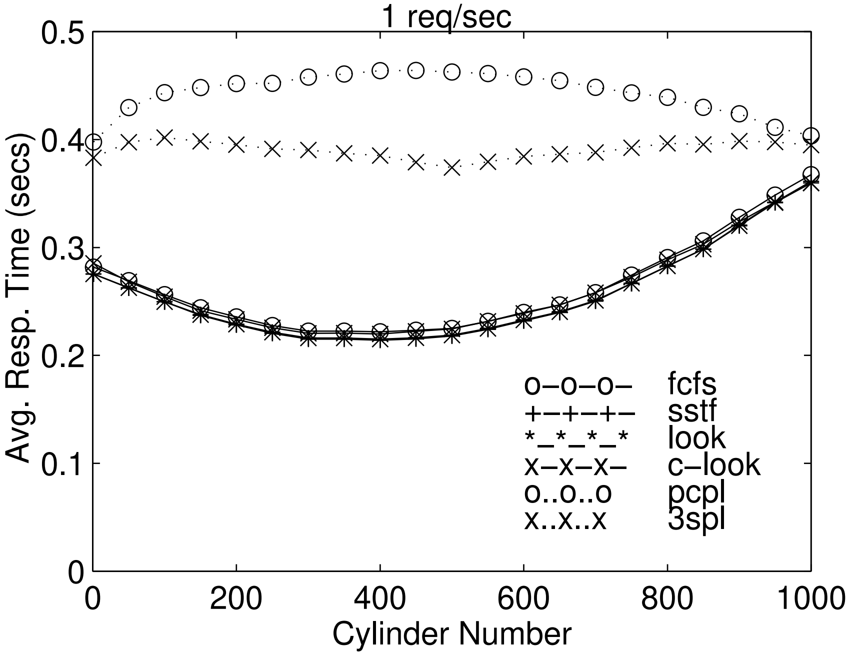


Fig. 7.3a

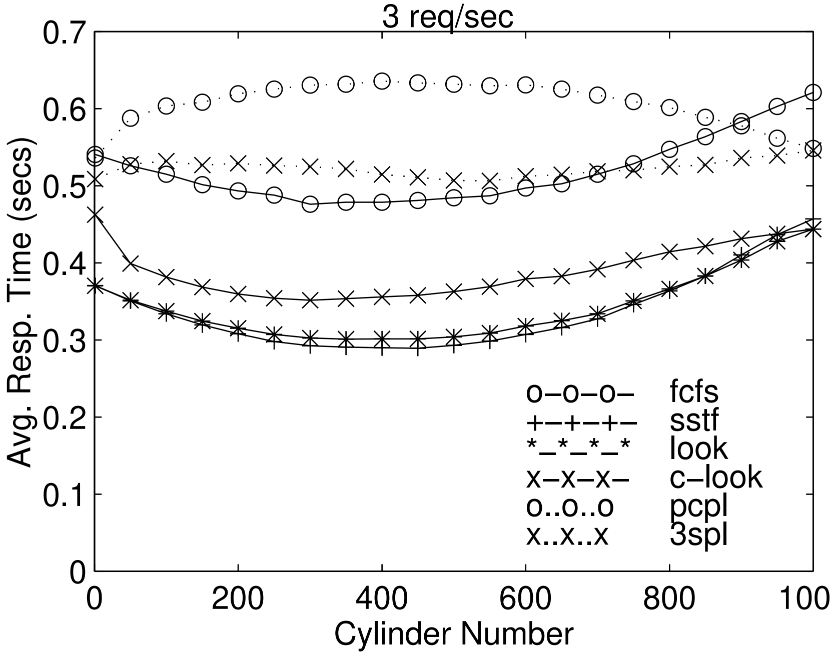


Fig. 7.3b

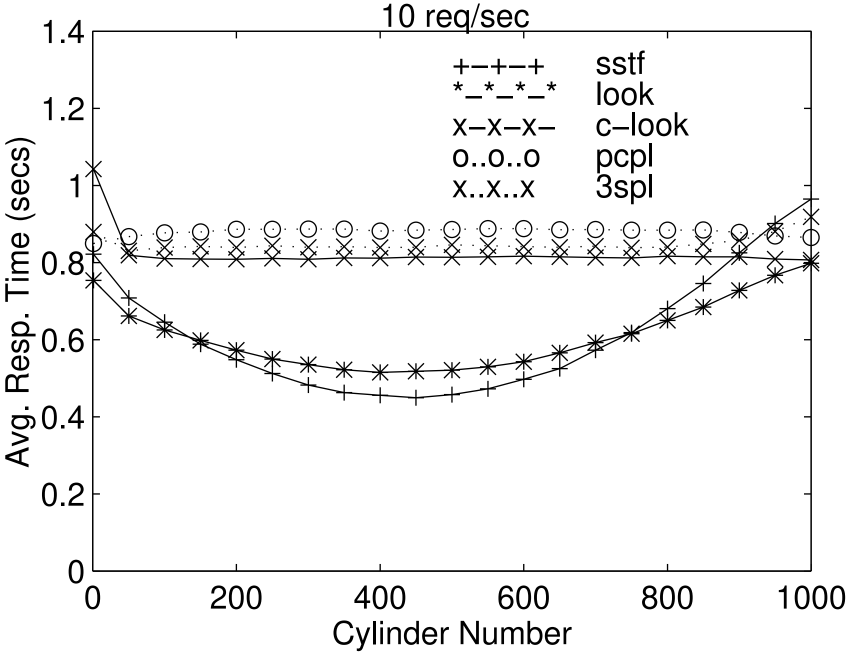


Fig. 7.3c

STOP Algorithms

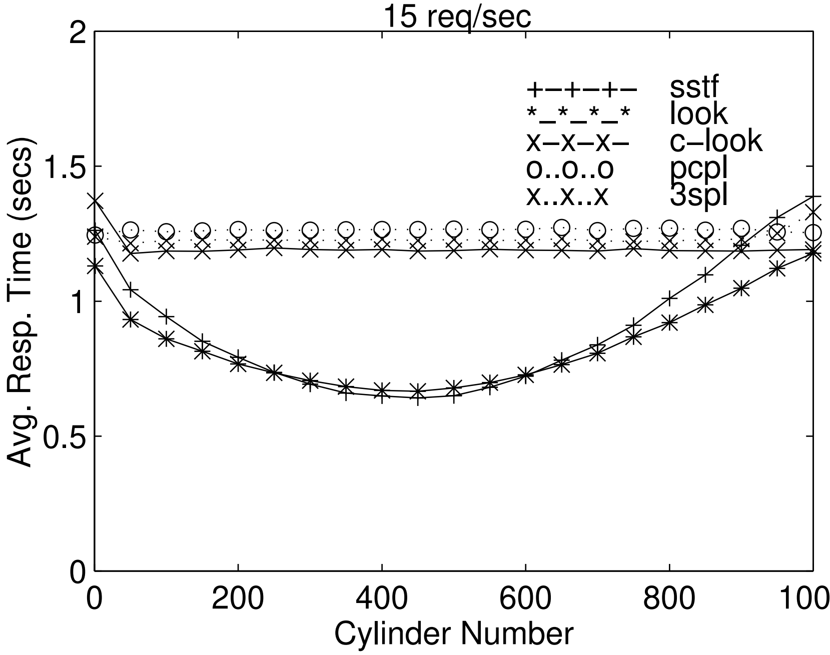


Fig. 7.3d

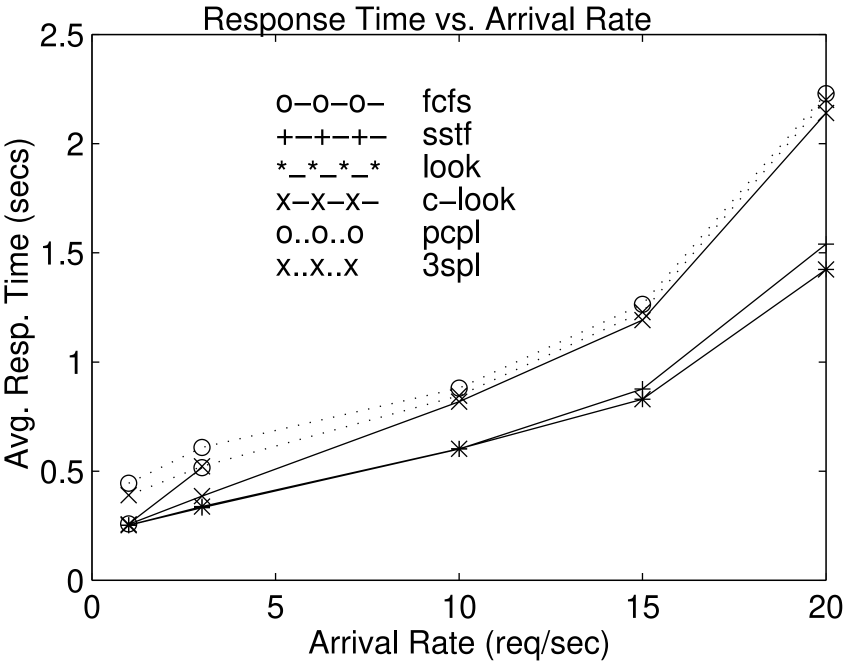


Fig. 7.3e

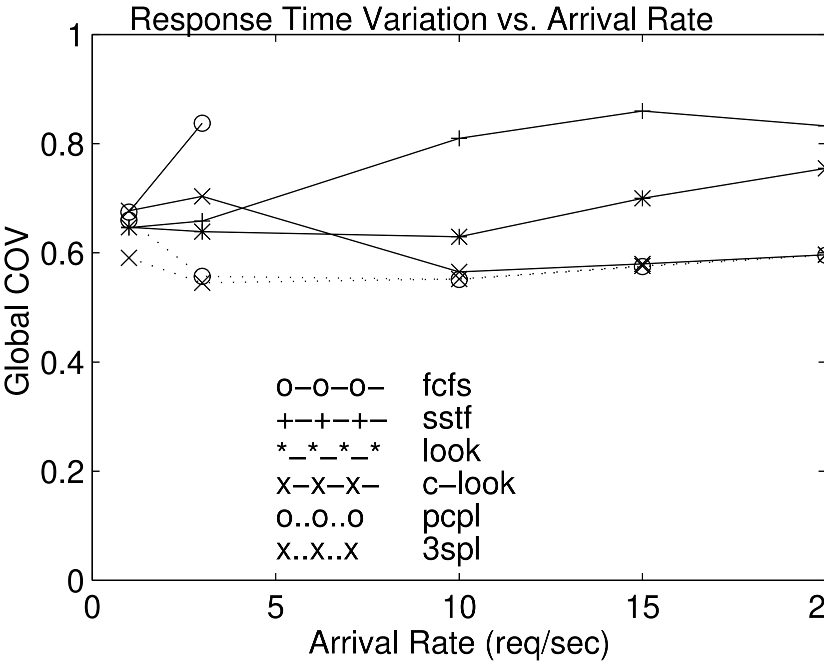


Fig. 7.3f

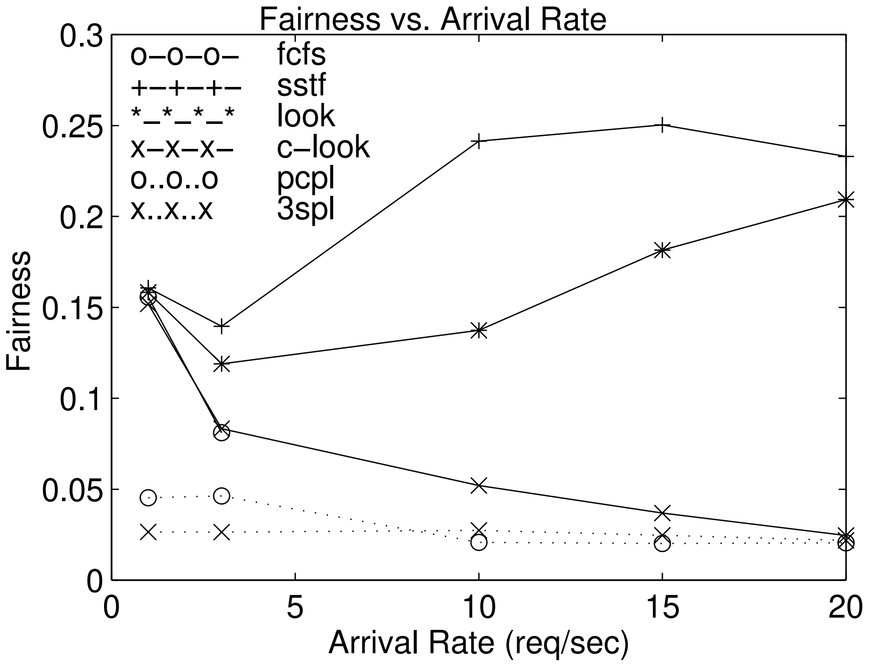


Fig. 7.3g

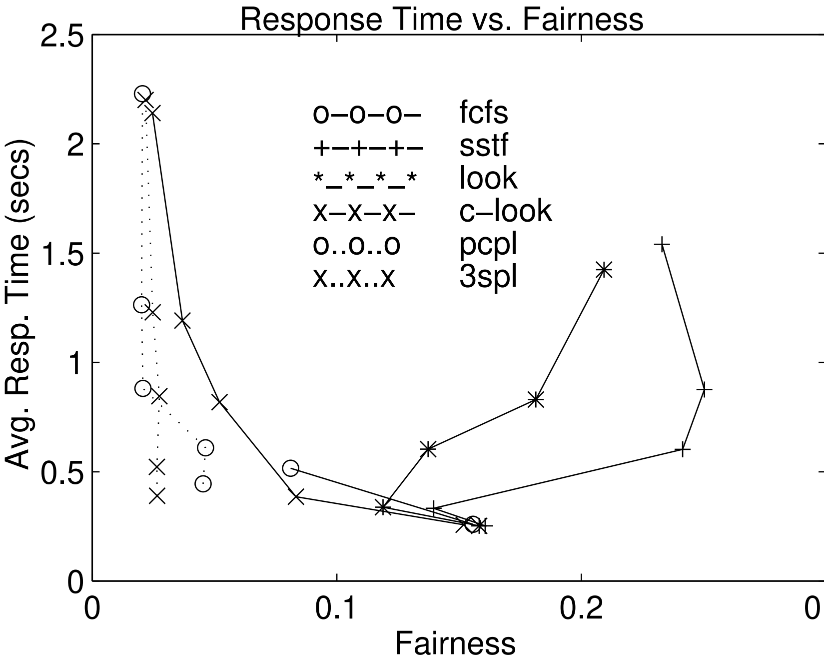


Fig. 7.3h

STOP Algorithms

scheduling decision, for heavy loads, as explained in Section 4.2 and 5.2.

DYNAMIC Scheduling

The results of the experiments evaluating the performance of the algorithms are shown in Figures 7.4a to 7.4h. From these figures, it is observed that all classical algorithms behave like FCFS at low arrival rates which is unfair due to the reasons given for GATE scheduling.

The SSTF scheduler also shows unfairness under heavy loadings. It favours central cylinders more than the extreme ones. Also, the response times for the inner cylinders is lower than that for the outer cylinders. This is due to the reasons given for STOP scheduling.

Looking at CLOOK, we observe that, as in the uniform data request distribution case, the algorithm becomes satisfactorily fair at high arrival rates. This is due the same reasons as given in Section 4.3. As in STOP scheduling, the LOOK scheduler is unfair under heavy loads, for similar reasons.

As in the uniform distribution case, both NSCP and PCLOOK remain satisfactorily fair for the entire loading range. For NSCP, this is because of the continuous movement of the disk head, for lighter loads. For heavier loads, it behaves like CPOLL, which is satisfactorily fair at these loads (Section 4.3). PCLOOK shows fairness due to the parking policy, for lighter loads (Section 5.3). For heavy loads, it degenerates to CLOOK, which is again fair for reasons given in Section 4.3.

7.2.2 Double Spike Skew

The distribution for Double Spike Skew is shown in Figure 7.5. This is also an example of spatially skewed distribution. There are two equally spaced peaks in the middle of the disk which shows that these cylinders are accessed much more frequently than the rest of the disk. A practical example of this kind of distribution is a disk which has some commonly used executables (e.g. unix commands, `vi`, `ls`) in the two frequently accessed set of cylinders or a disk containing two filesystems with their inodes located in these

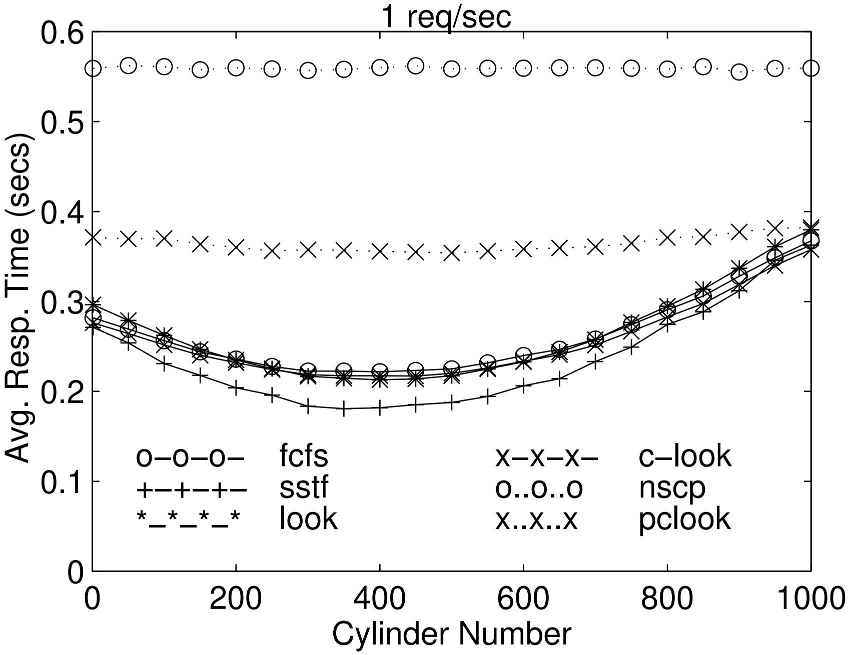


Fig. 7.4a

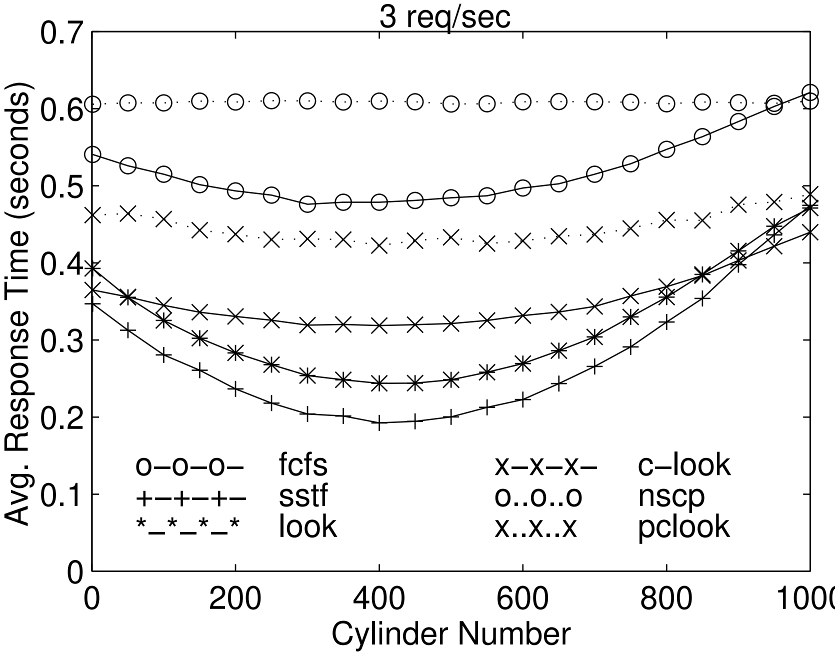


Fig. 7.4b

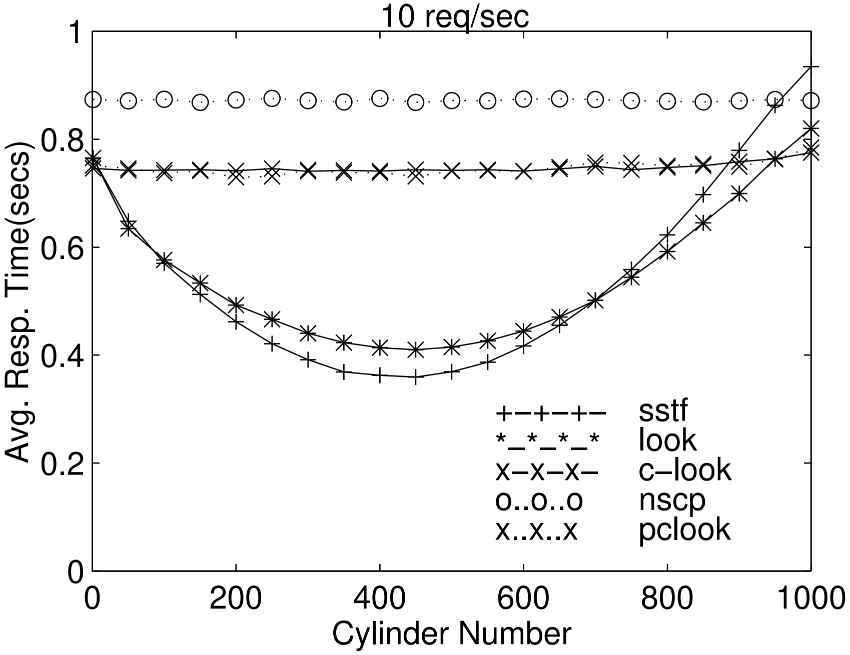


Fig. 7.4c

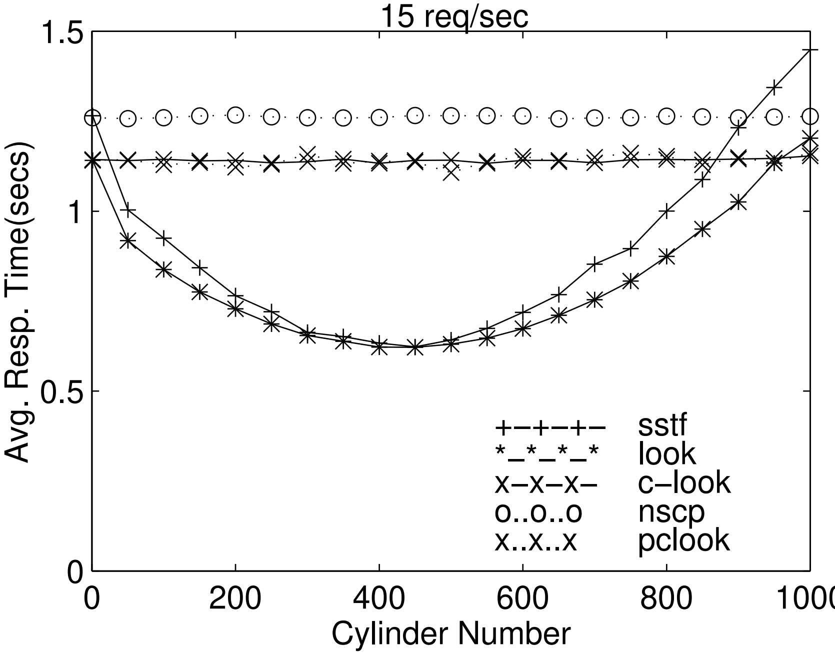


Fig. 7.4d

DYNAMIC Algorithms

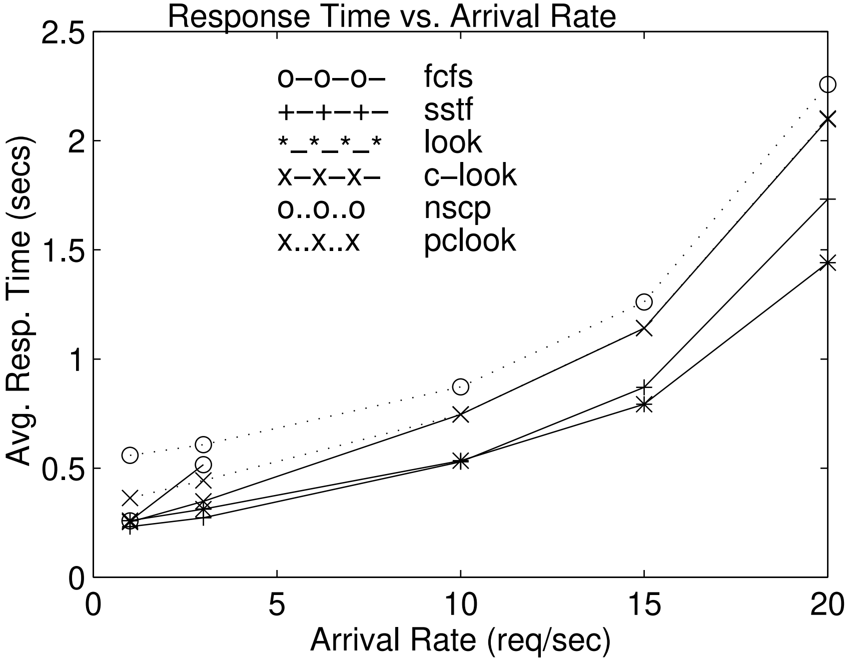


Fig. 7.4e

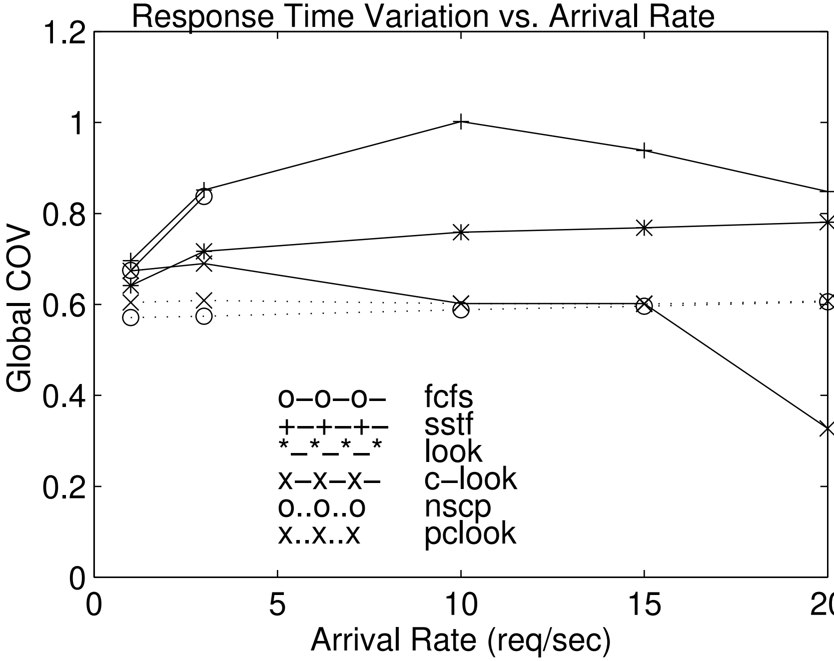


Fig. 7.4f

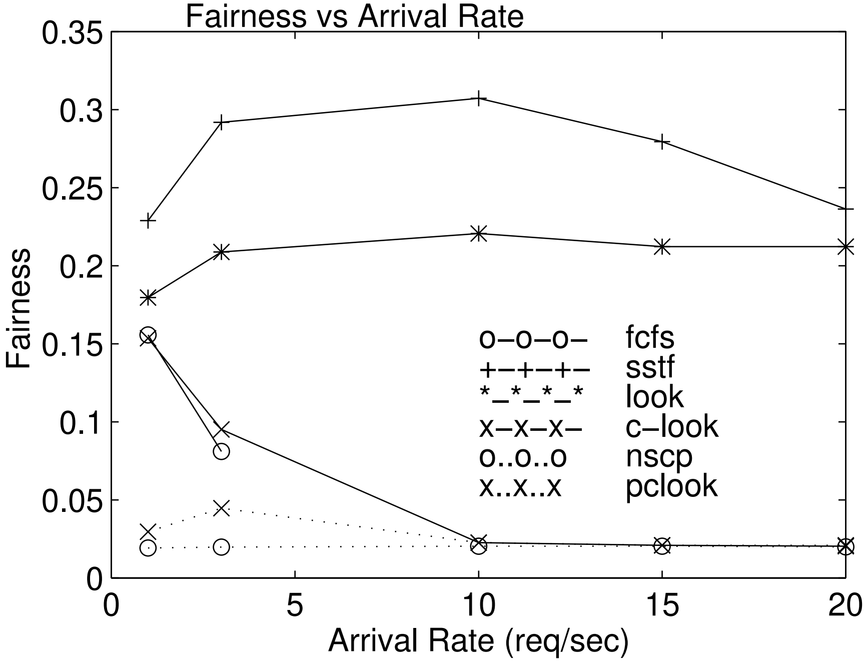


Fig. 7.4g

DYNAMIC Algorithms

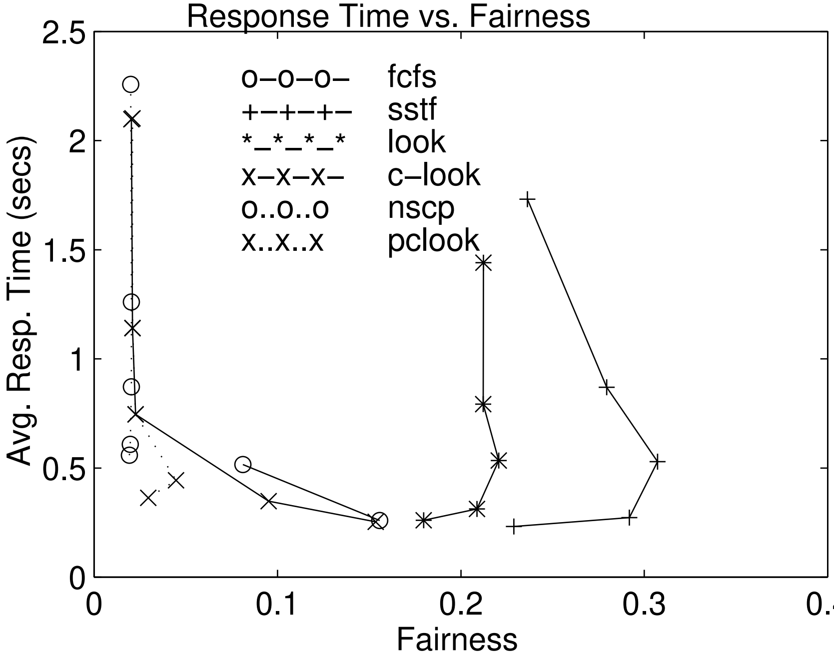


Fig. 7.4h

regions. As in the single spike distribution, the algorithms are simulated with the same two modifications. We now discuss the results for each scheduling granularity separately below.

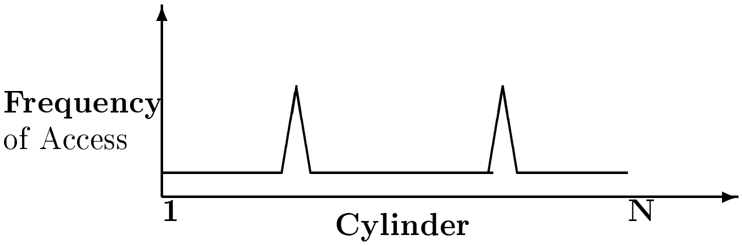


Figure 6.3: Double Spike Skewed Distribution

GATE Scheduling

The results of the experiments evaluating the performance of the algorithms are shown in Figures 7.6a through 7.6h. From these figures, we observe that FCFS saturates earlier than the other algorithms and is unfair for light loads. It has a flat cylinder time distribution for cylinders lying between the two set of peak cylinders. Outside this interval, there is a bias against the outer cylinders. This is due to the following reason: New requests typically arrive to an empty system and find the disk head idle at one of the cylinders in the two peaks. Hence, the average seek for the requests for cylinders in the above interval is $N/4$ and increases in the outer neighbourhood of the peaks reaching $N/2$ for the extreme cylinders. As in single spike skewed distribution, both CLOOK and LOOK behave like FCFS for such light loads (Figure 7.6a).

The LOOK scheduler becomes fairer as the load becomes heavier whereas CLOOK shows a bias towards the inner cylinders as in the uniform distribution case. But as in single spike distribution, the increase is not linear as we go from cylinder 1 to cylinder N. There are two sudden increasing jumps for cylinders just in front of the two peaks. This is because groups mostly contain requests for the peak cylinders and hence, cylinders that are served after these peak cylinders have to wait longer for service.

Moving on to CPOLL scheduler , we notice that under light loads, it favours outer

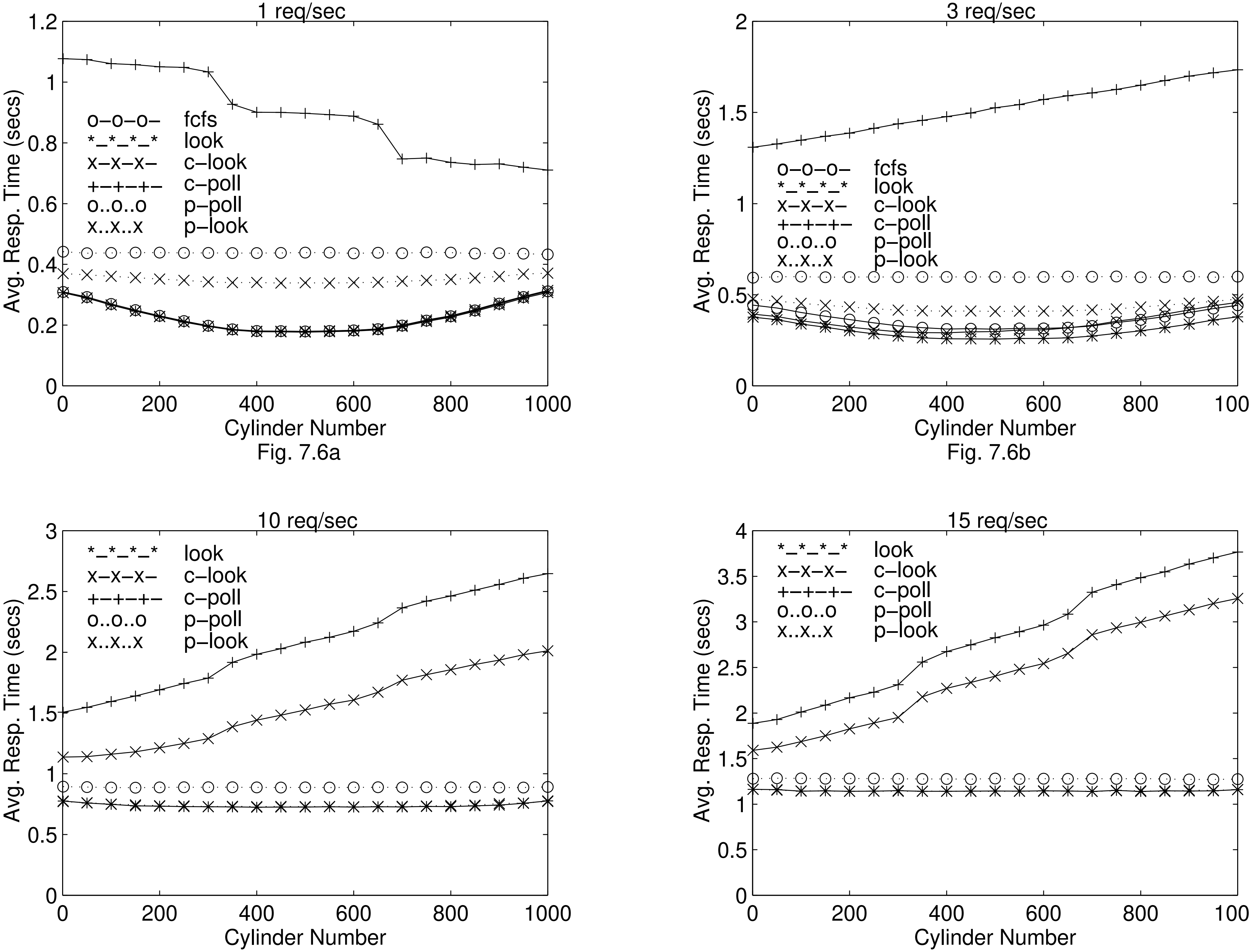


Fig. 7.6a

Fig. 7.6b

Fig. 7.6c

Fig. 7.6d

GATE Algorithms

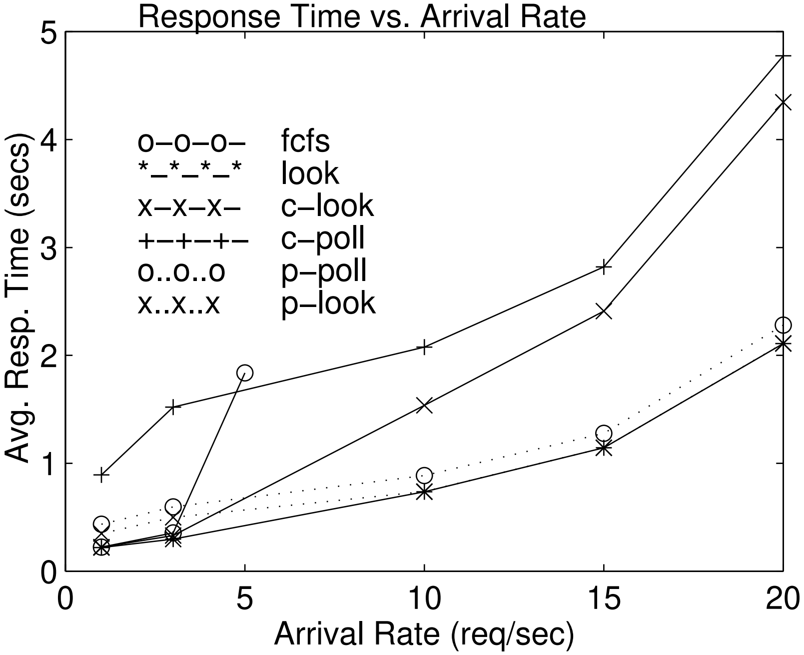


Fig. 7.6e

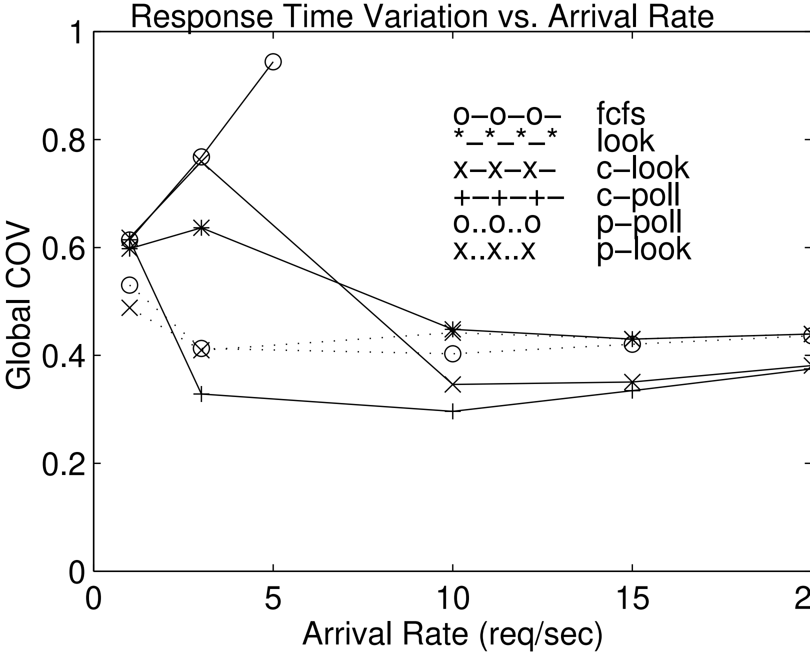


Fig. 7.6f

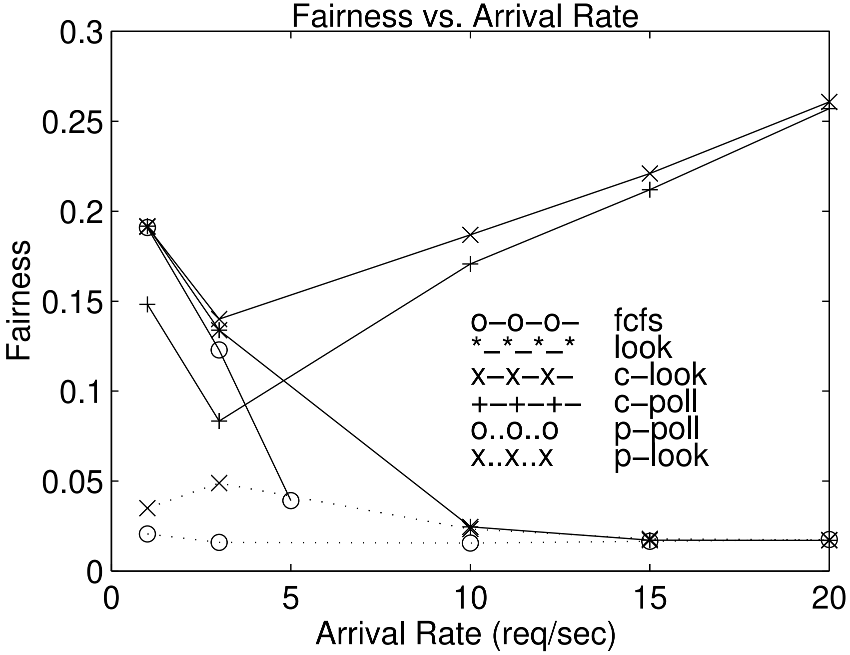


Fig. 7.6g

GATE Algorithms

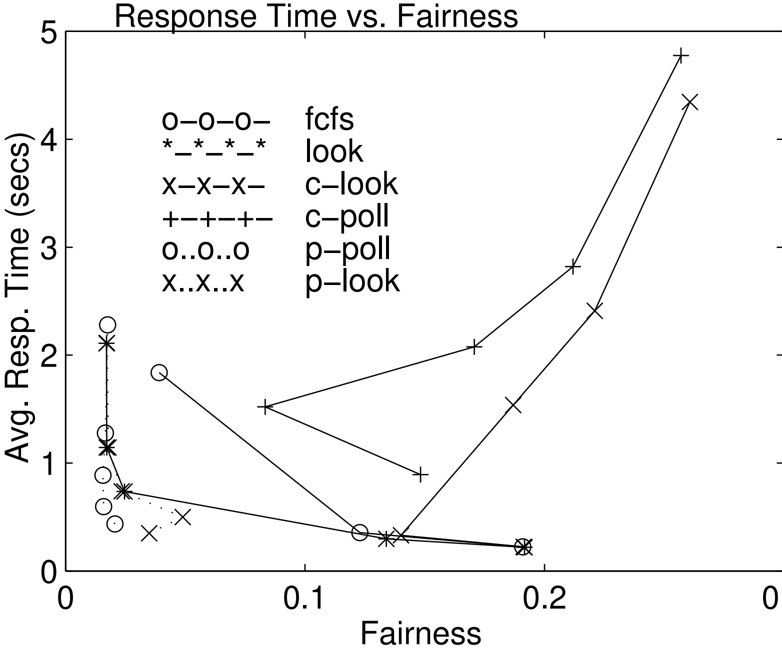


Fig. 7.6h

cylinders. This is because typically for these cylinders, the disk head would not need to rewind before serving the request. There are two sudden decreasing jumps in the response time curve just after the peak cylinders. This is because new requests typically find the head idle at one of the peak cylinders. This leads to a small seek distance to be traveled for requests for cylinders just ahead of these peaks. For heavy loads, the cylinder response time distribution is similar to that of CLOOK and for the same reasons as described above.

Finally, we notice that both PPOLL and PLOOK are satisfactorily fair over the entire loading range as in the uniform distribution case. This is due to the same reasons as explained in Section 4.1 and 5.1.

STOP Scheduling

The results of the experiments evaluating the performance of the algorithms are shown in Figures 7.7a through 7.7h. From these figures, we observe that, as in GATE scheduling, FCFS saturates much before any other algorithm and is unfair under light loads. SSTF, LOOK and CLOOK behave like FCFS at such low arrival rates (Figure 7.7a) due to requests arriving typically to an empty system.

Moving on to SSTF, we observe that it is quite unfair especially under heavy loads, when the response time for cylinders between the two spikes is lower as compared to that for the outer cylinders. Also, in the region between the two spikes, the response time distribution is flat and increases as we go from the spikes to the extreme cylinders. This can be explained as follows: Since the majority of the requests are for the spike cylinders, the head tends to get localized to the region between the two spikes thereby increasing the response times for the outer cylinders.

The LOOK scheduler is also unfair for the entire loading range. As the loading becomes heavier, the difference in the response times between the set of cylinders between the two spikes and the outer cylinders increase. This is because there are relatively more requests for the spike cylinders which are served at regular intervals due to near-periodic inter-visit times for these cylinders unlike that for the outer cylinders. Moving on to CLOOK, we

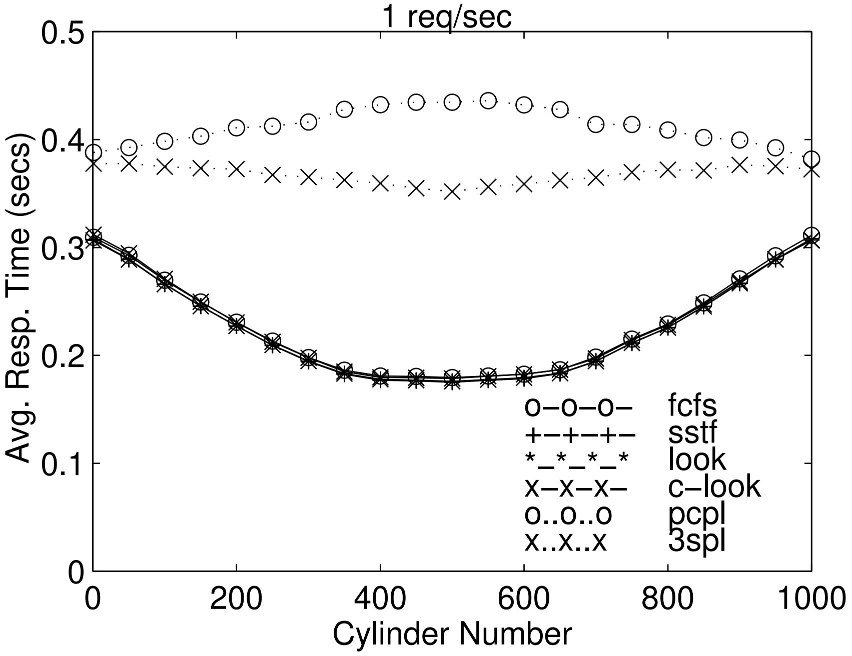


Fig. 7.7a

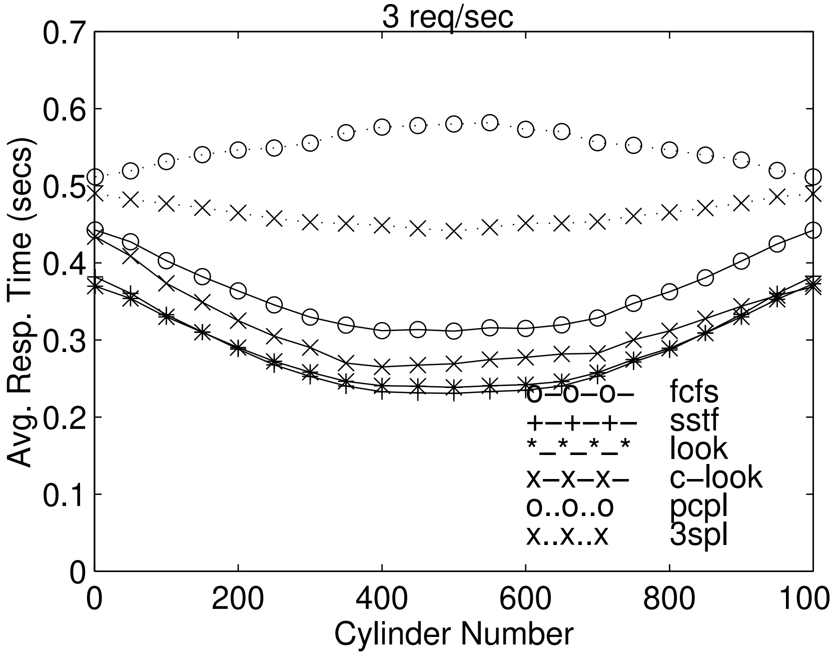


Fig. 7.7b

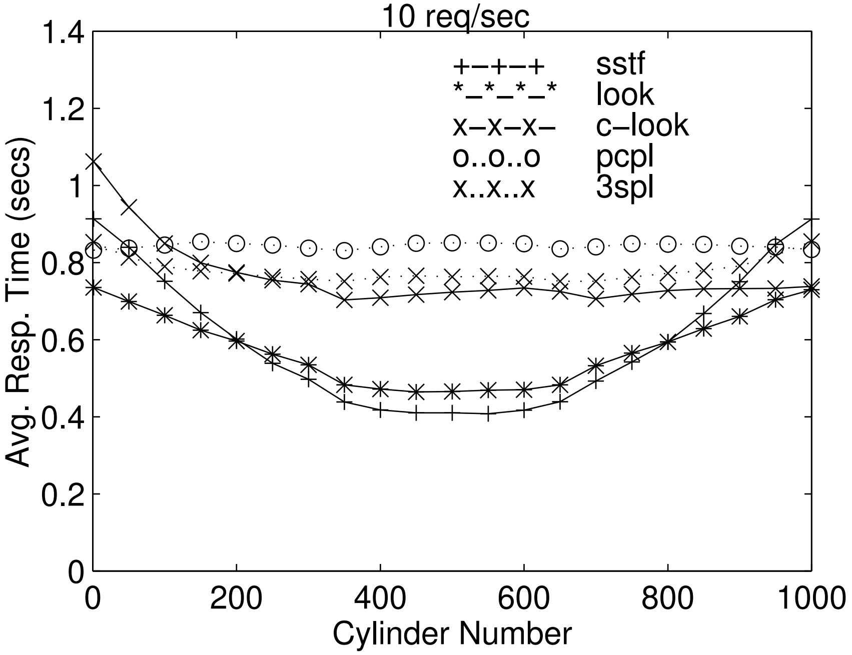


Fig. 7.7c

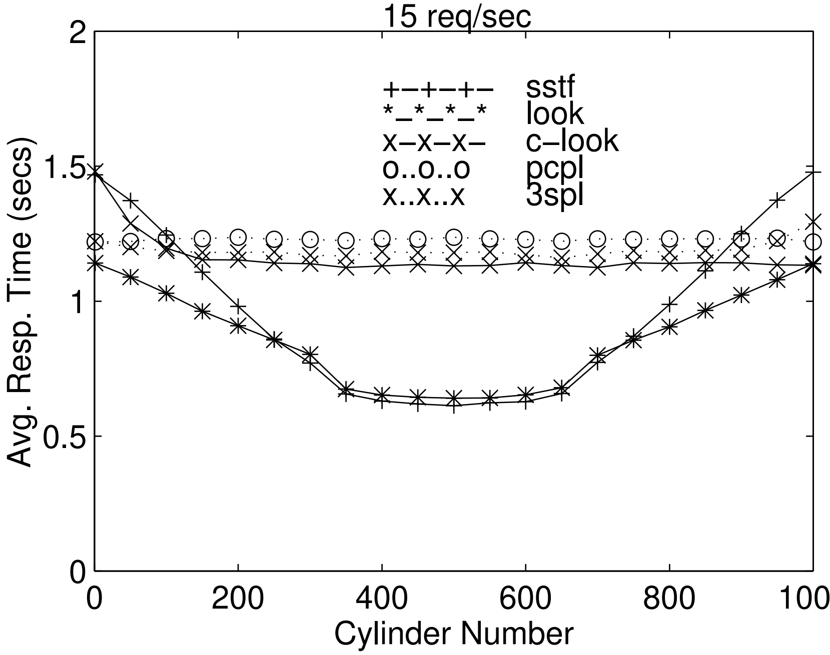


Fig. 7.7d

STOP Algorithms

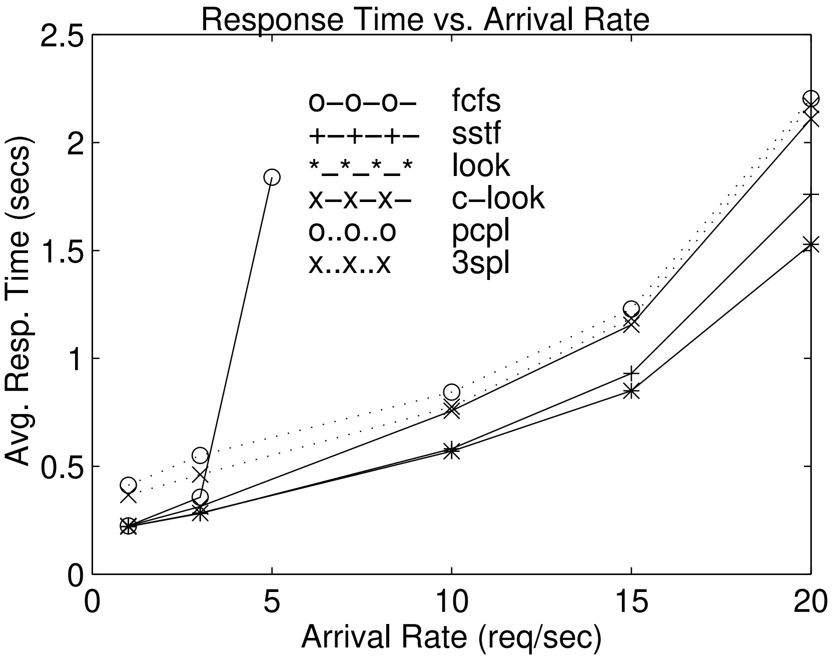


Fig. 7.7e

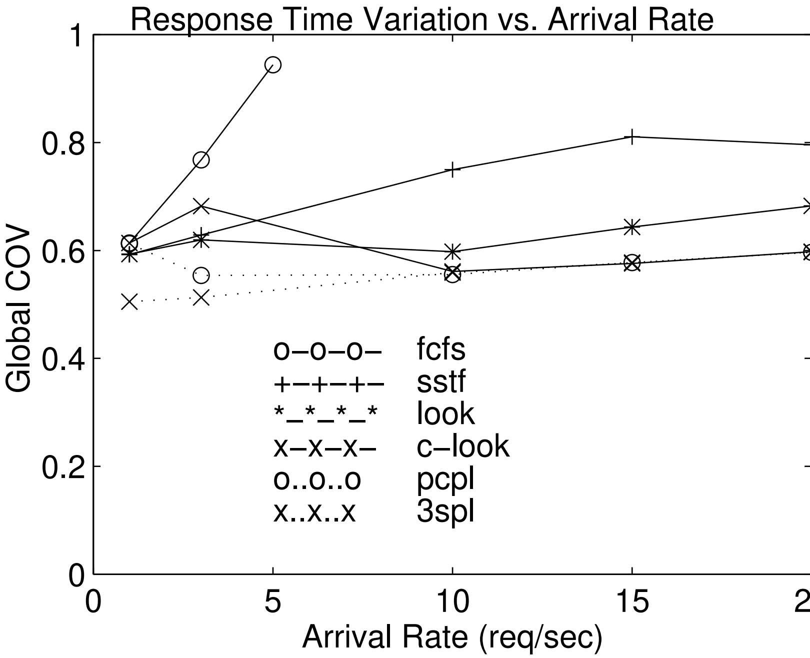


Fig. 7.7f

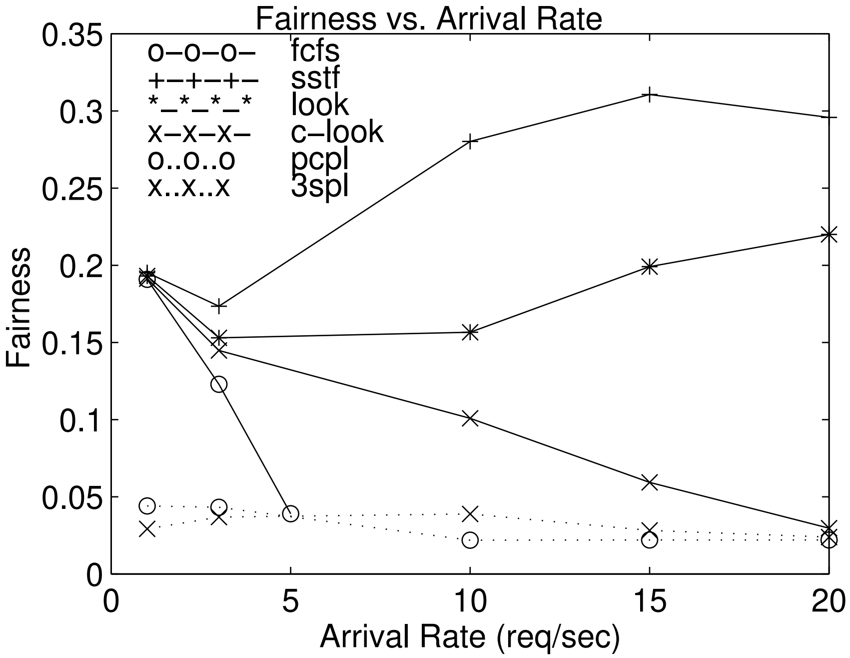


Fig. 7.7g

STOP Algorithms

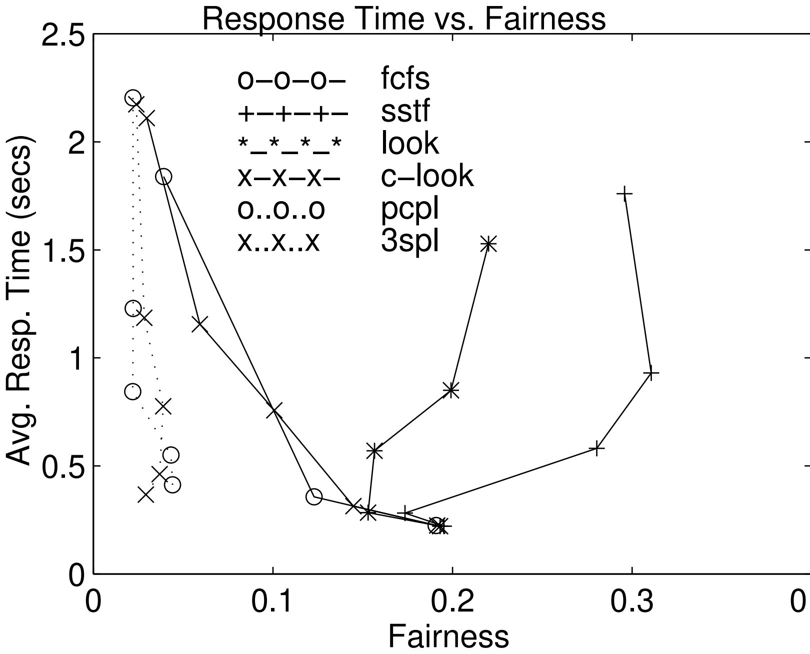


Fig. 7.7h

observe that it is not fair under heavy loading due to a bias against the inner cylinders. This is because of the same reasons as explained in the case of uniform data request distribution.

Also, as in the uniform distribution case, both PCPL and 3SPL remain satisfactorily fair for the entire range of loading. This is due to similar reasons as given in Section 4.2 and 5.2.

DYNAMIC Scheduling

The results of the experiments evaluating the performance of the algorithms are shown in Figures 7.8a through 7.8h. From these figures, we observe that as in GATE scheduling, FCFS saturates earlier than any other algorithm and is unfair under light loads. LOOK and CLOOK both behave like FCFS under such light loads as requests typically arrive to an empty system.

The SSTF scheduler favours the central cylinders due to the reasons given for STOP scheduling. Looking at LOOK scheduler, we observe that when the load becomes heavier, it becomes unfair, favouring the set of cylinders situated between the two spikes. As in STOP scheduling, this is due to smaller differences in the inter-visit times of the disk head for these cylinders as compared to that for the other cylinders. Moving on to CLOOK, we find that it becomes increasingly fair as as the load becomes heavier. This is due to the same reasons as given for uniformly distributed data requests (Section 4.3).

The NSCP scheduler is found to be satisfactorily fair over the entire loading range, but with a significantly higher mean response time. This is due to similar reasons as given in Section 4.3.

Finally turning our attention to PCLOOK, we find that it is unfair for low and medium arrival rates. As explained in Section 5.3, the response time charateristics at such loads are a result of two factors: Firstly, the parking of the head when idle leads to nearly equal average seeks for all cylinders. Secondly, when there is a pending request at the end of service of a request, the head goes straight to it and serves it. Since most requests are for the spike cylinders, this results in lower average seeks for the set of cylinders between

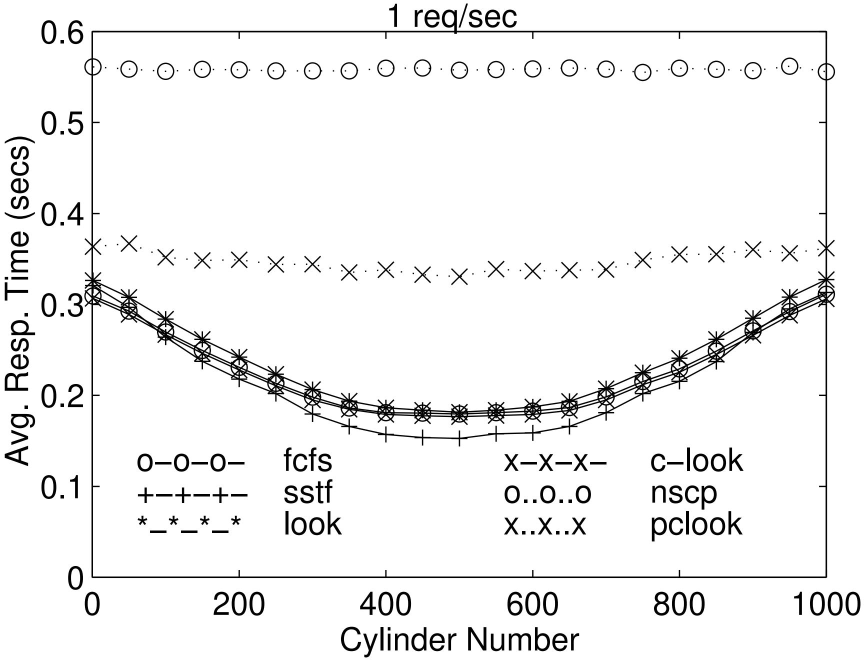


Fig. 7.8a

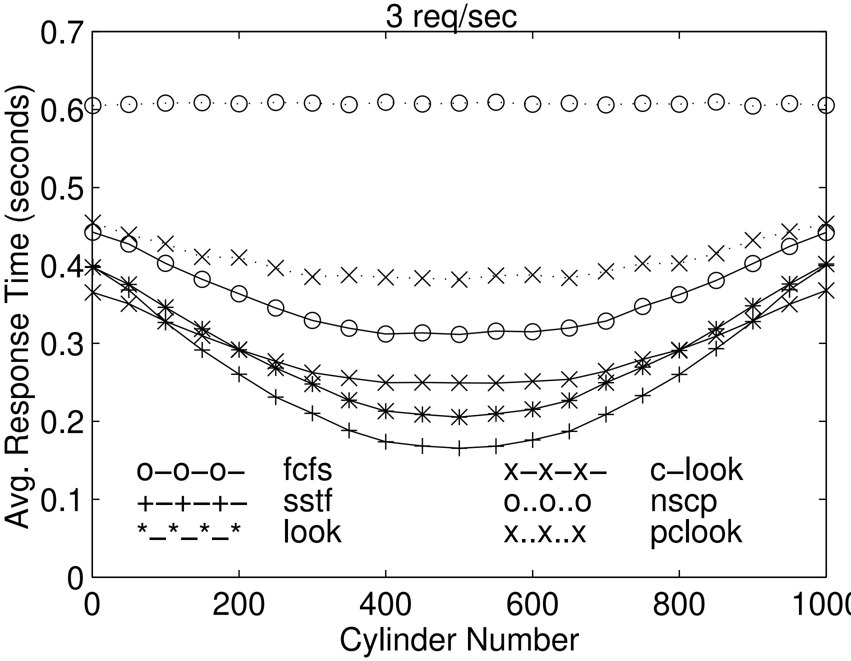


Fig. 7.8b

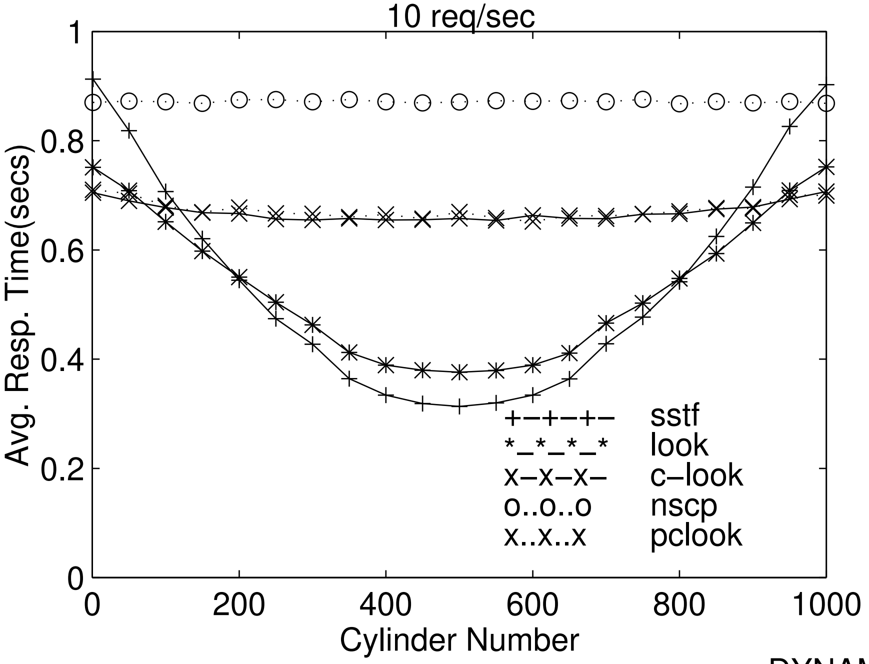


Fig. 7.8c

DYNAMIC Algorithms

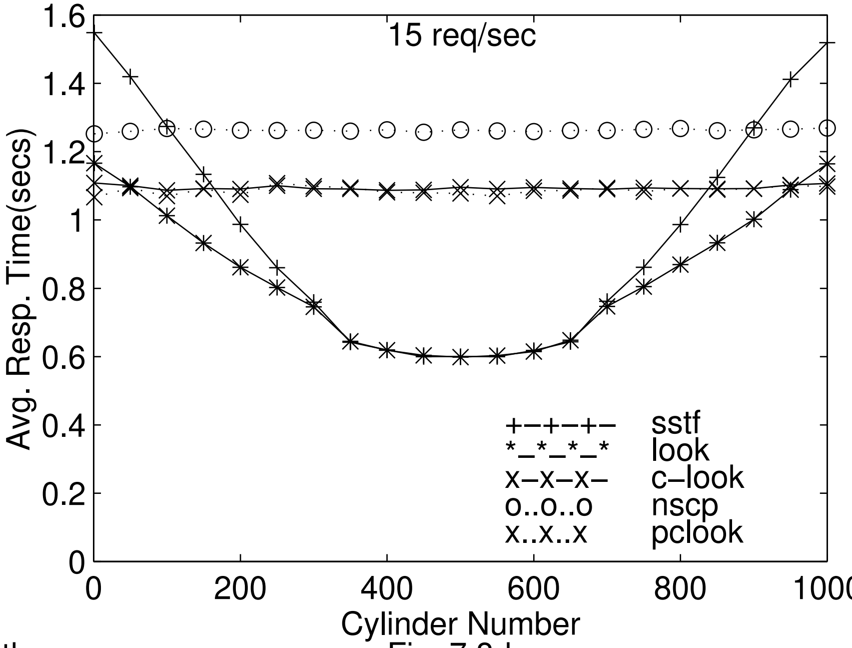


Fig. 7.8d

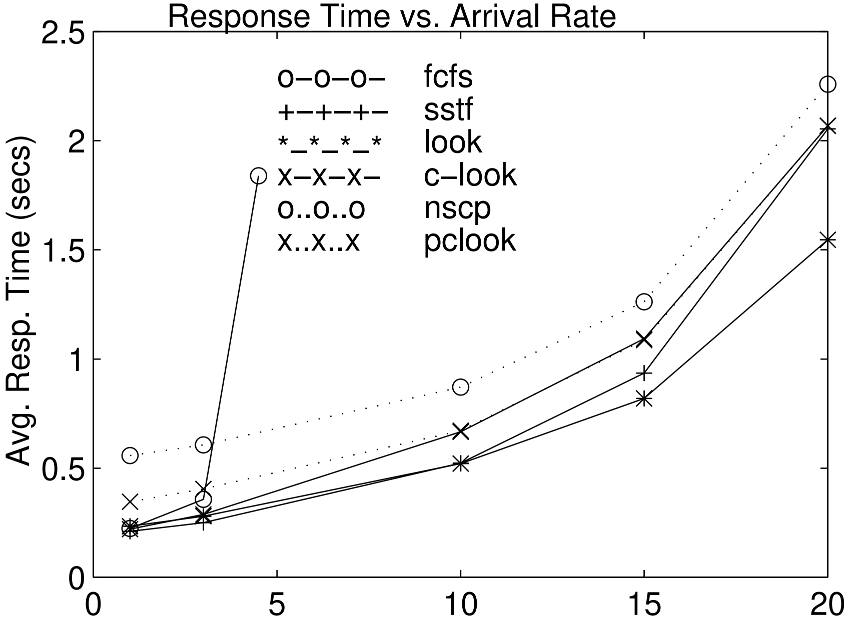


Fig. 7.8e

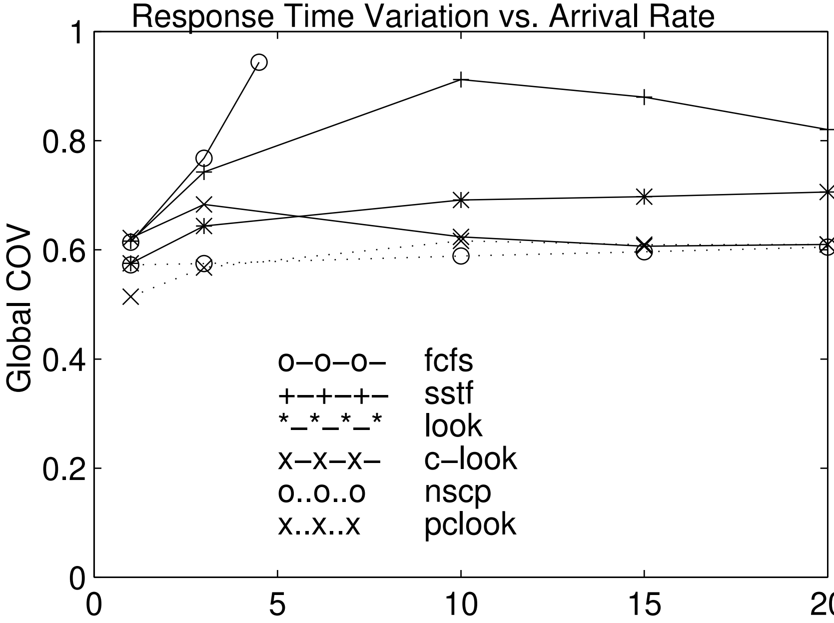


Fig. 7.8f

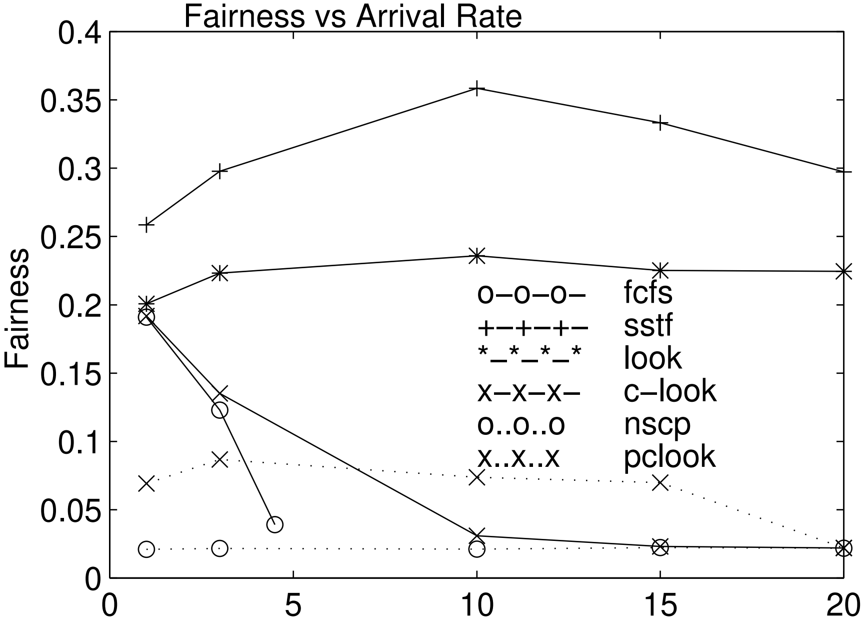


Fig. 7.8g

DYNAMIC Algorithms

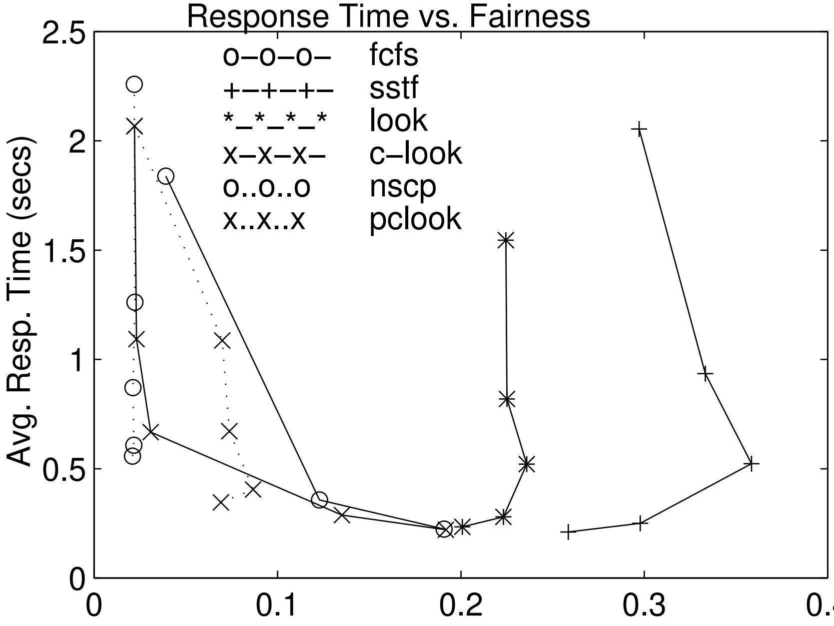


Fig. 7.8h

the two spikes. As the arrival rate increases, this effect is reduced as there are generally multiple pending requests in the queue. Hence, under these loads the algorithm behaves like CLOOK which is satisfactorily fair for high arrival rates.

7.2.3 Moving Gaussian Skew

In this distribution, given the last cylinder number for which there was a request, the next cylinder number to be requested for, is determined using a normal distribution with mean as the last cylinder number. This leads to a high probability of consecutive requests being in the neighbourhood of each other. It represents the fact that most files are accessed sequentially and are allocated contiguous blocks. The size of the neighbourhood is decided by the standard deviation used to generate the normally distributed random variable. We have used $\sigma = 5$, where σ is the standard deviation. The probability of requests to be in $\pm 2\sigma$ neighbouhood is 0.95.

GATE Scheduling

The results of the experiments evaluating the performnce of the algorithms are shown in Figures 7.9a through 7.9h. From these figures, we observe that FCFS, LOOK and CLOOK behave in a similar fashion for the entire loading range (Figures 7.9a - 7.9d). This is because, at any instant of time, there will be requests only for a small set of neighbouring cylinders (due to the temporal locality exhibited by the requests). Hence, all these algorithms will have a very small value of seek distance to be traveled between requests. This results in low cylinder mean response times for all cylinders. Also, since the seek time is so small, the response time is dominated by latency and transfer times, which are modeled to be constant for all requests. Due to this, all these algorithms are satisfactorily fair over the entire range of loading and have a very small mean response time (Figures 7.9e, 7.9g). For this kind of an access pattern, FCFS does not saturate earlier than other algorithms, unlike for other data request distrubutions.

Moving on to CPOLL, it is observed that the algorithm shows considerable fairness but this is at the expense of a significant increase in the mean response time (Figure 7.9h).

This is due to the sweeping of the disk surface from one edge to another even though requests are confined to a set of neighbouring cylinders at any instant of time. For light loads, there are two kinks in the cylinder mean response time distribution curve. The mean response time for the outer cylinders is less than that of the inner cylinders. This is due to the following reason: Due to the skewed access pattern of the requests, a request for any cylinder will arrive to find the disk head idle at a neighbouring cylinder. For the *innermost* cylinders, most of the neighbourhood cylinders will be ahead of the particular cylinder. Hence, requests for these cylinders typically have to wait for the head to move to the outermost cylinder and then to the innermost cylinder before recommencing the sweep. Requests for the *outermost* cylinders will not be required to wait for this, as for these cylinders, most of the neighbourhood cylinders will be behind the particular cylinder. For other cylinders, the neighbourhood will consist of cylinders on both sides. Under heavy loads, the CPOLL scheduler shows the opposite behaviour. The mean response time for the innermost cylinders is less than that of the outermost cylinders. This can be explained as follows: Each group will contain requests for a few neighbouring cylinders and will be served as the disk head moves outwards. The innermost cylinders will typically belong to a group which has them in the beginning of the group. Hence, they will be served earlier within the group. Similarly, the outermost cylinders typically form the outer end of a group and hence, have to wait longer for service than the others in the group.

Turning our attention to PPOLL, it is observed that though it is satisfactorily fair for all arrival rates, it has a high mean response time as compared to the classical algorithms like FCFS, LOOK and CLOOK. The fairness is due to the same reasons as given in Section 4.1. The increase in response time is due to unnecessary head movement to the disk extremities after serving each group.

Finally, looking at PLOOK, it is observed that it is unfair for the entire loading range, being most unfair under medium loads (Figure 7.9g, 7.9h). There is a bias against the extreme cylinders. This can be explained if we compare the performances of PLOOK and PPOLL. Consider the following situation: Suppose the head is at an extreme cylinder, say, one of the inner cylinders and after serving the current group, there are no more

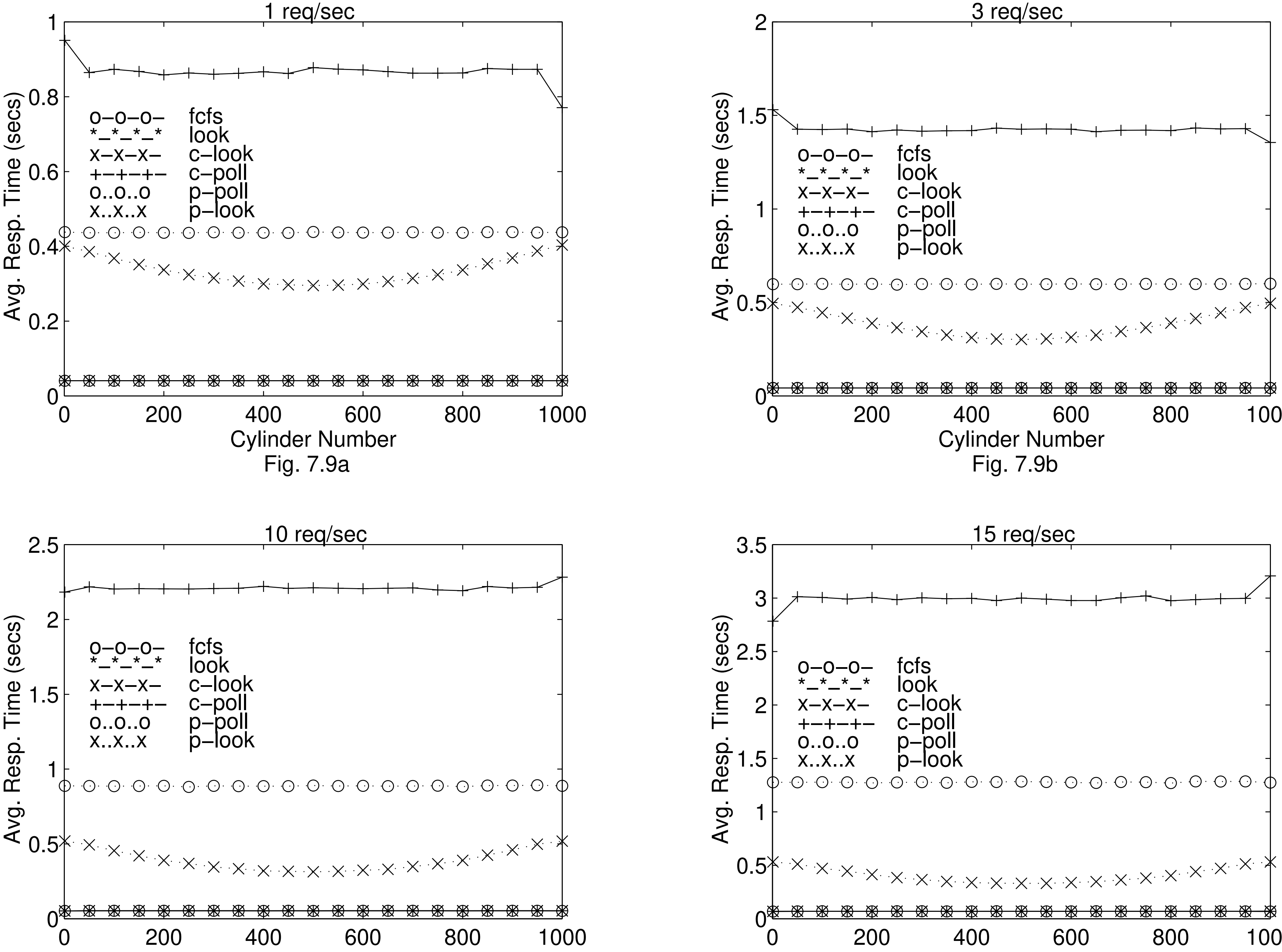


Fig. 7.9a

Fig. 7.9b

Fig. 7.9c

Fig. 7.9d

GATE Algorithms

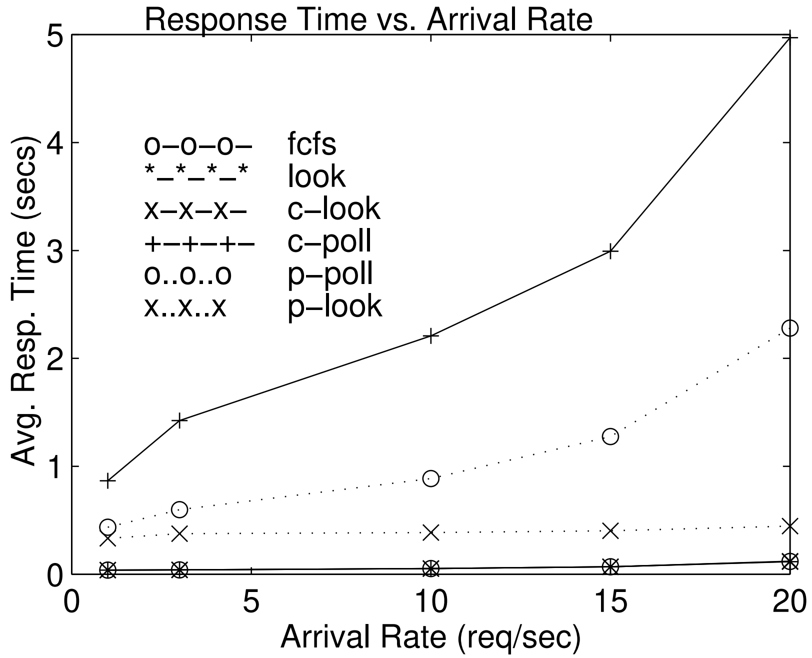


Fig. 7.9e

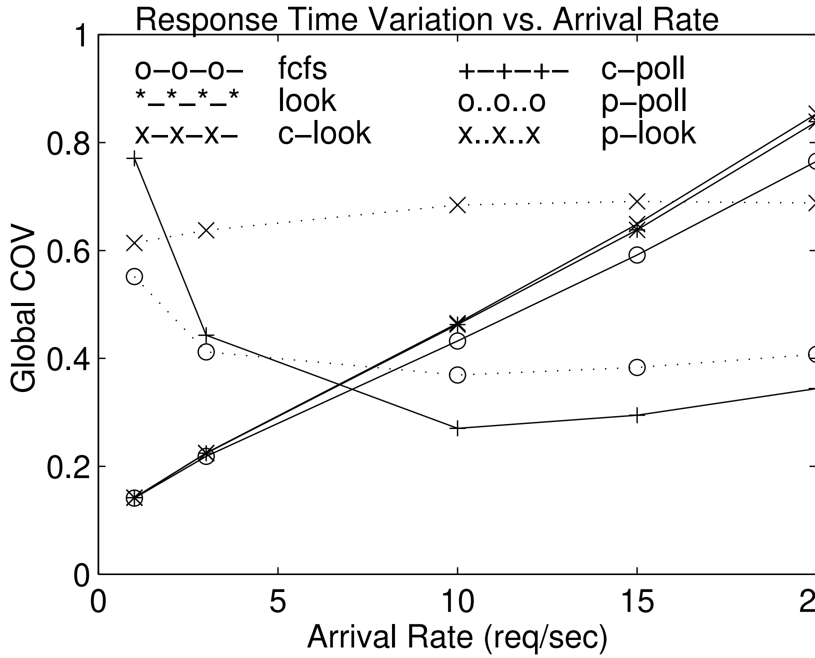


Fig. 7.9f

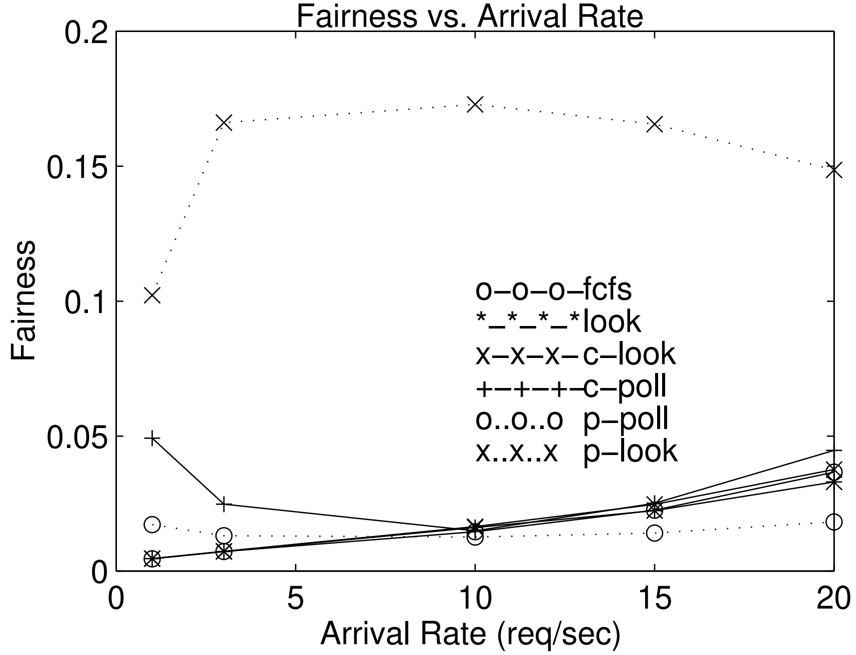


Fig. 7.9g

GATE Algorithms

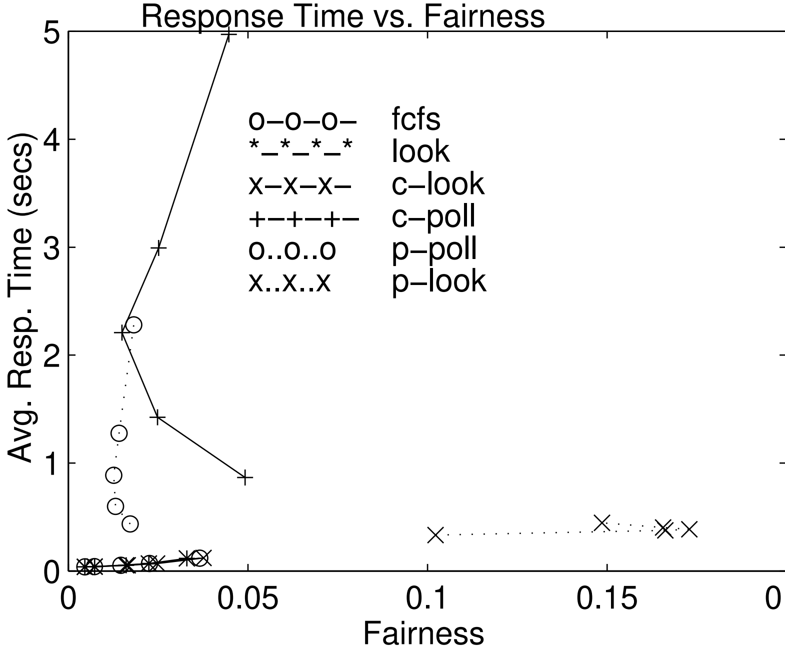


Fig. 7.9h

requests in the system. In this case, both PPOLL and PLOOK will go and park at one of the edges. Let this be the outer edge. After reaching the parking spot, a new group is made and the disk head starts to move towards the first cylinder to be served. This group would also contain requests for the inner cylinders (due to the temporal skew). The disk head would need to sweep nearly the whole disk surface before it reaches the first cylinder to be served. During this time, the requests arriving would be added to the next group. These would again be for the inner cylinders. After the head completes serving this group, PPOLL would now park at the inner edge before serving the next group whereas PLOOK would start serving the next group immediately. Anyway, since this parking spot for PPOLL is so close to the group, there is not much of a difference between the two algorithms. However, a similar situation when the requests are for the central cylinders, would save PLOOK a sweep of nearly half the disk surface as compared to PPOLL. Hence, PLOOK reduces the mean response times for the central cylinders more than that for the extreme cylinders. This effect is dominant at medium loads and reduces under heavier loads, when the idle periods are reduced (Figure 7.9g).

STOP Scheduling

The results of the experiments evaluating the performance of the algorithms are shown in Figures 7.10a - 7.10h. From these figures, we observe that, similar to GATE scheduling, FCFS, SSTF, LOOK and CLOOK, all behave in a similar fashion for all arrival rates. This is due to the same reasons as given for GATE scheduling.

The PCPL scheduler is no longer fair for any arrival rate. It shows a distinct bias towards the central cylinders. This is due to the following reason: When the sweep direction is inside–out, the outer cylinders suffer. This can be explained as follows: Suppose the requests in the queue are for the outer cylinders (all the requests will be for neighbouring cylinders due to the temporal locality exhibited by the requests). When there are no further requests in the current direction, the disk head rewinds to the inner end before making the next scheduling decision. As the head moves from the inner end to the cylinder to be served next (which will again be one of the outer cylinders), requests

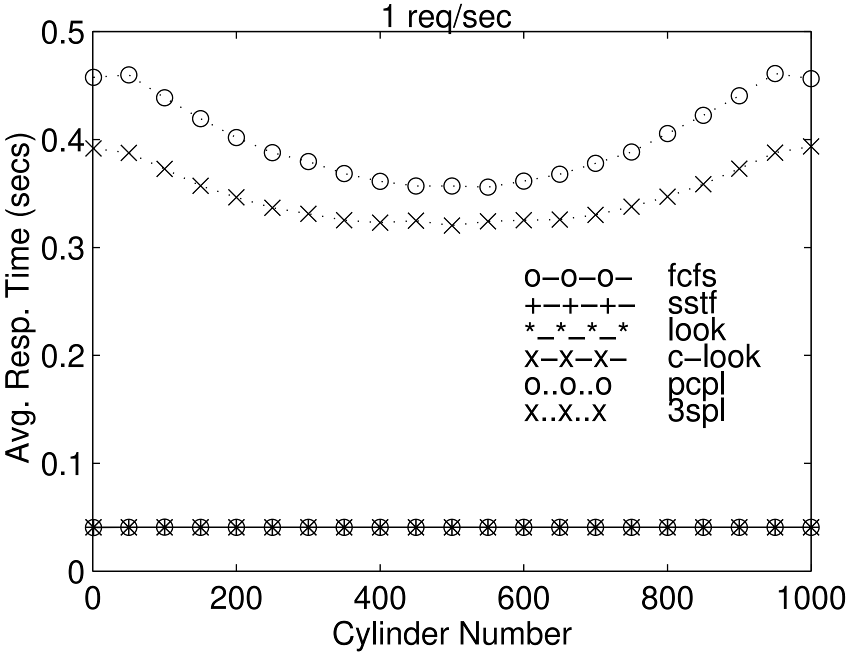


Fig. 7.10a

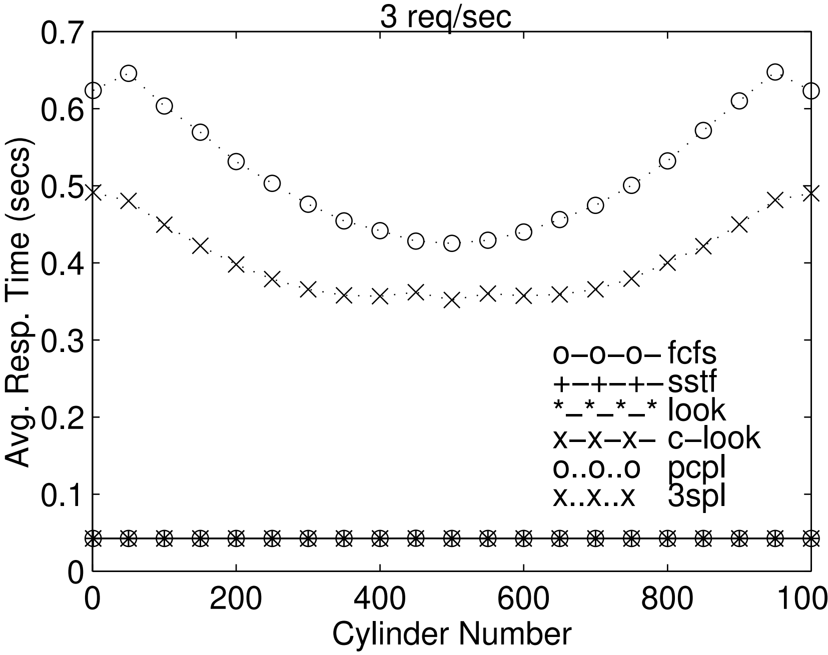


Fig. 7.10b

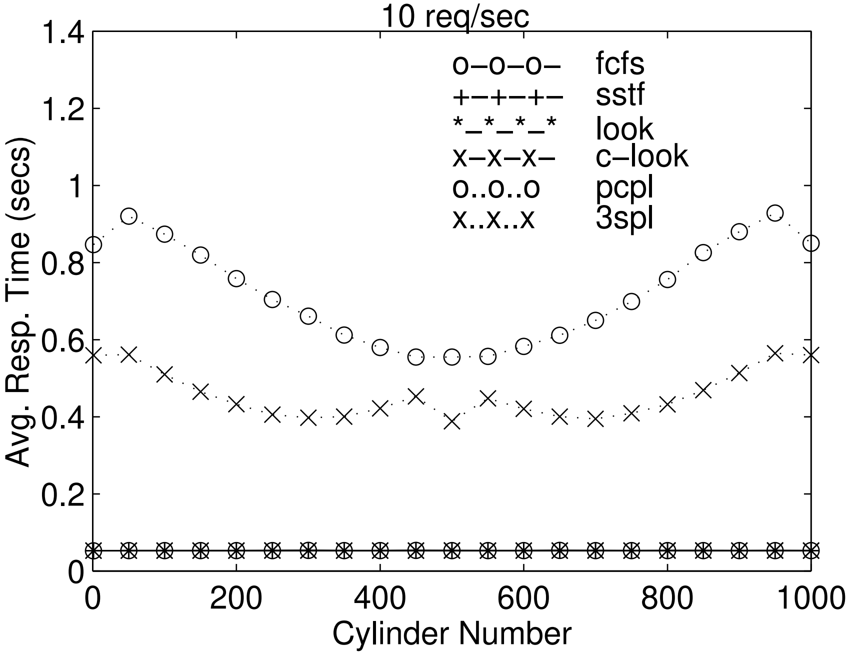


Fig. 7.10c

STOP Algorithms

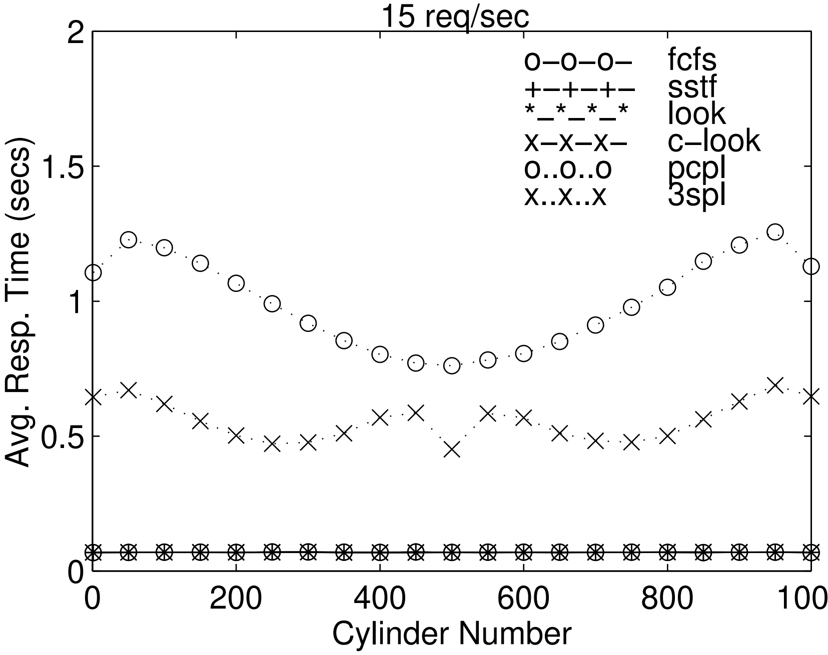


Fig. 7.10d

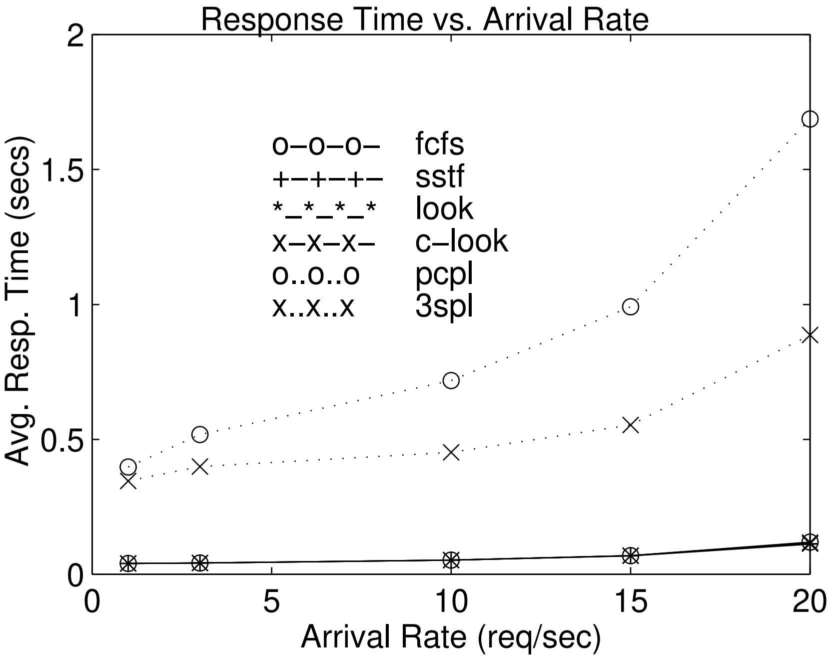


Fig. 7.10e

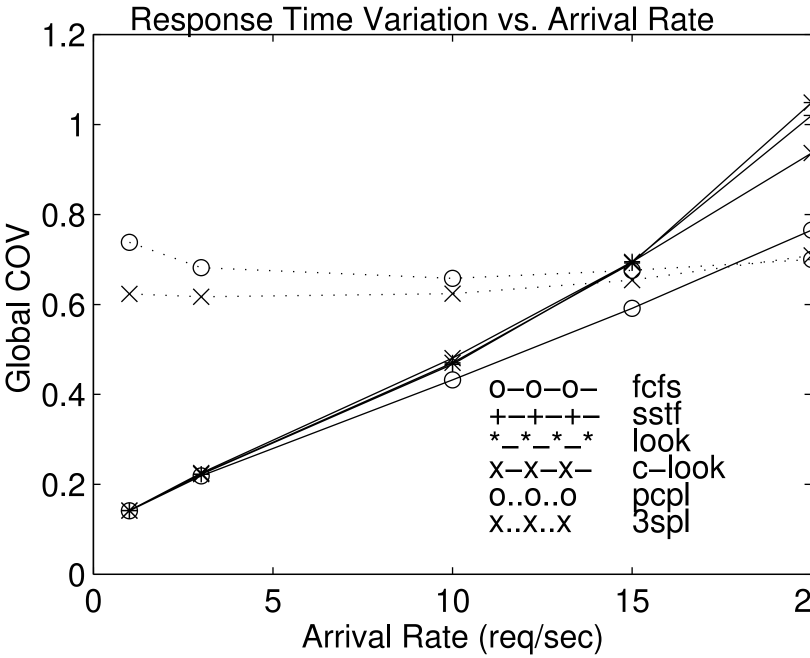


Fig. 7.10f

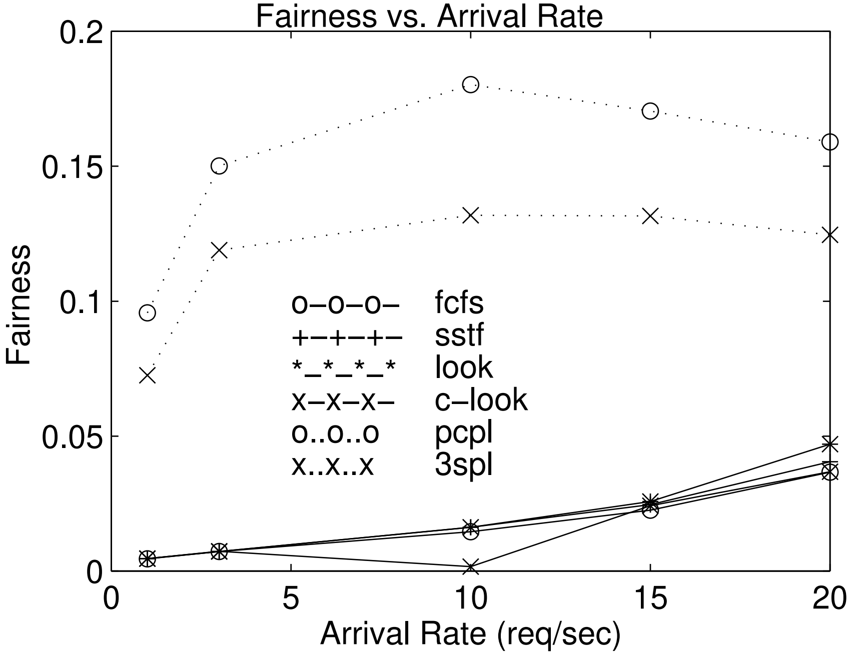


Fig. 7.10g

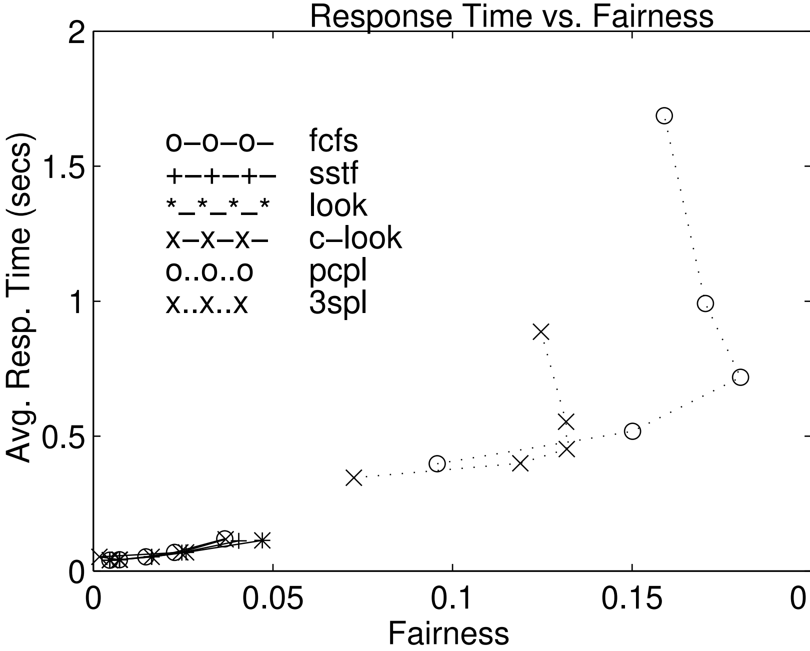


Fig. 7.10h

STOP Algorithms

arriving for cylinders on the way, will be delayed as they will be served only in the next sweep. However, the outermost cylinders do not suffer much, as they are mostly ahead of the chosen cylinder to be served. Hence, the response time is maximum, not for the outermost cylinders, but for slightly lower ones (Figure 7.10a - 7.10d). Similarly, requests for the inner cylinders suffer when the sweep direction is outside–in. The central cylinders are not affected much in either case and hence have a lower response time.

Moving on to 3SPL, we observe this is also not fair for any arrival rate (Figures 7.10g, 7.10h). This is again because of reasons similar to those given for PCPL except that, here there are three points to rewind to, instead of two, as in PCPL. When the sweep direction is inside–out, the outer cylinders and the ones just before the middle cylinder, suffer (Figure 7.10c, 7.10d). This is because they need to rewind sweeping nearly half the disk surface, when there are no requests ahead. As in PCPL, as the head moves towards the chosen cylinder to be served, requests arriving for cylinders on the way, will be served only in the next sweep. Similarly, when the sweep direction is outside-in, the inner cylinders and the ones just after the middle cylinder, suffer. The cylinders in the neighbouhood of $N/4$ and $3N/4$ have lower response times as they rewind sweeping approximately one-fourths of the disk surface, for both sweep directions. Cylinders very close to the central cylinder also have a low response time as they mostly rewind to the middle cylinder.

DYNAMIC Scheduling

The results of the experiments evaluating the performance of the algorithms are shown in Figures 7.11a through 7.11h. From these figures, it is observed that FCFS and CLOOK behave similarly for the entire loading range. This is due to the same reasons as given for GATE scheduling. Since LOOK and SSTF are also expected to behave in the similar fashion (like in the STOP scheduling case), their results are not shown in the graphs.

The NSCP scheduler is observed to be fair at low and medium arrival rates (Figures 7.11a, 7.11b), but with a significantly high mean response time. This is due to the same reasons as explained in Section 4.1. Under heavy loads, it shows some amount of unfairness (Figure 7.11d). The innermost cylinders have a higher mean response time

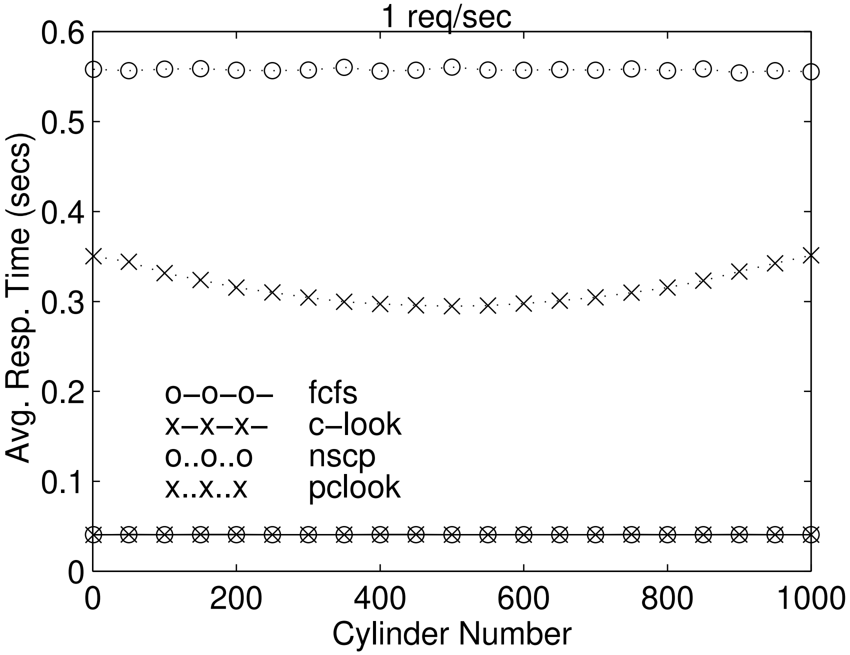


Fig. 7.11a

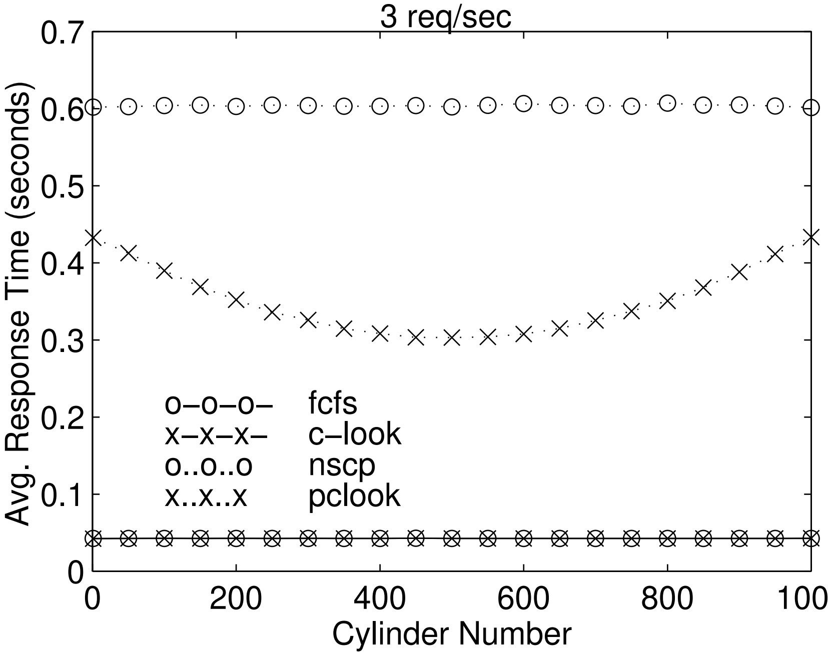


Fig. 7.11b

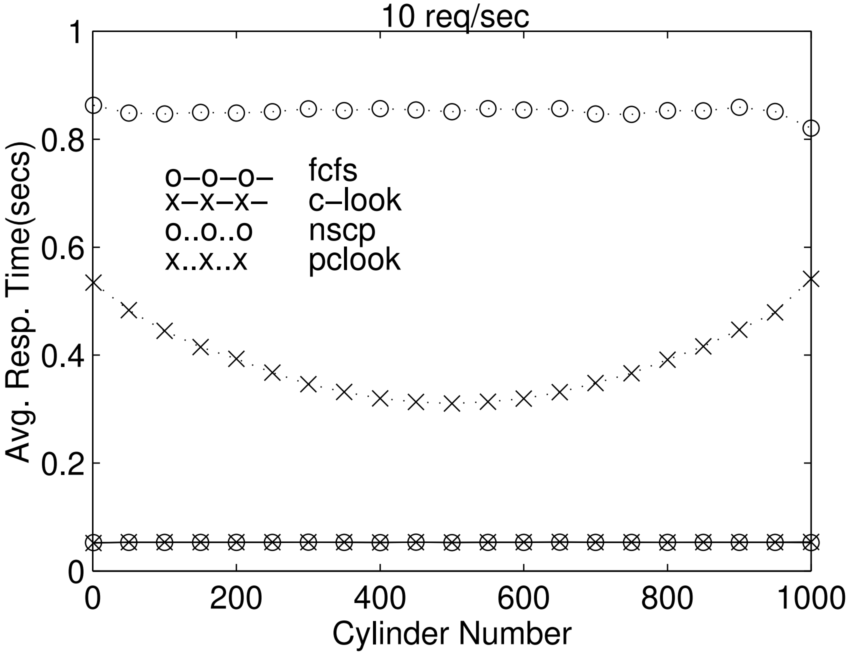


Fig. 7.11c

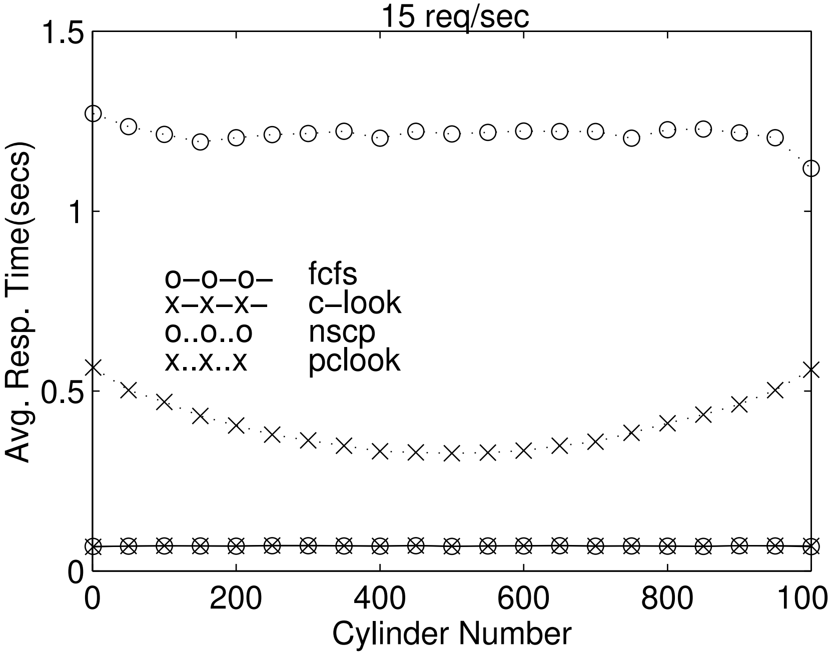


Fig. 7.11d

DYNAMIC Algorithms

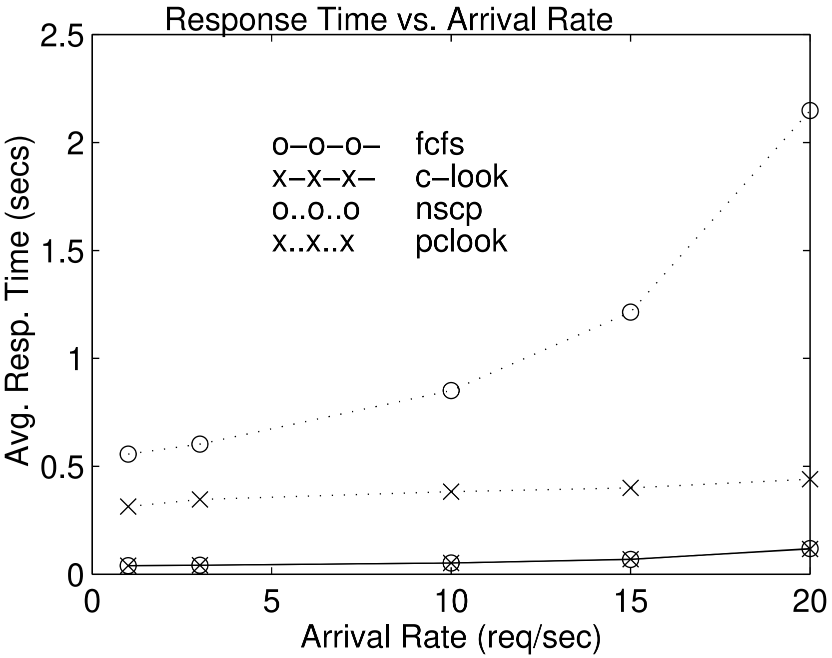


Fig. 7.11e

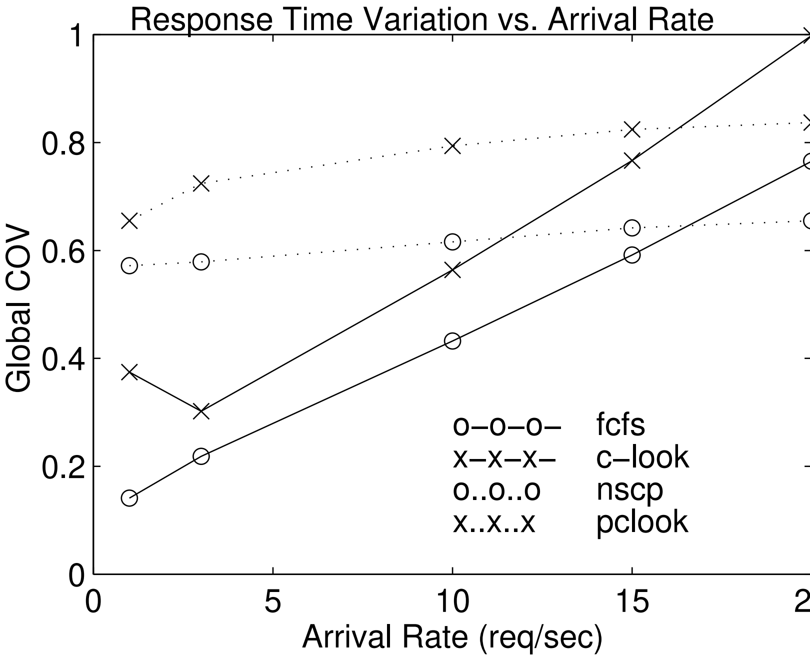


Fig. 7.11f

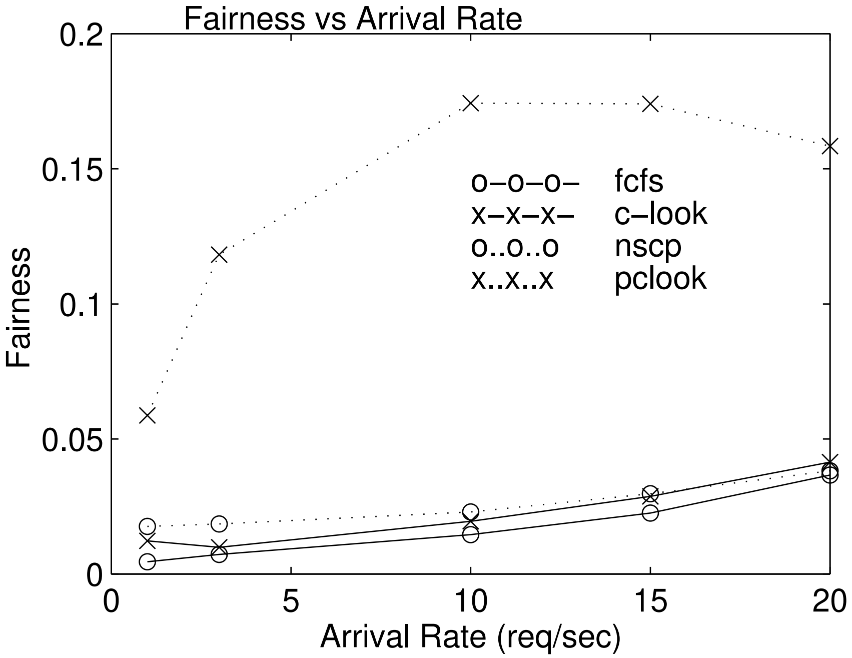


Fig. 7.11g

DYNAMIC Algorithms

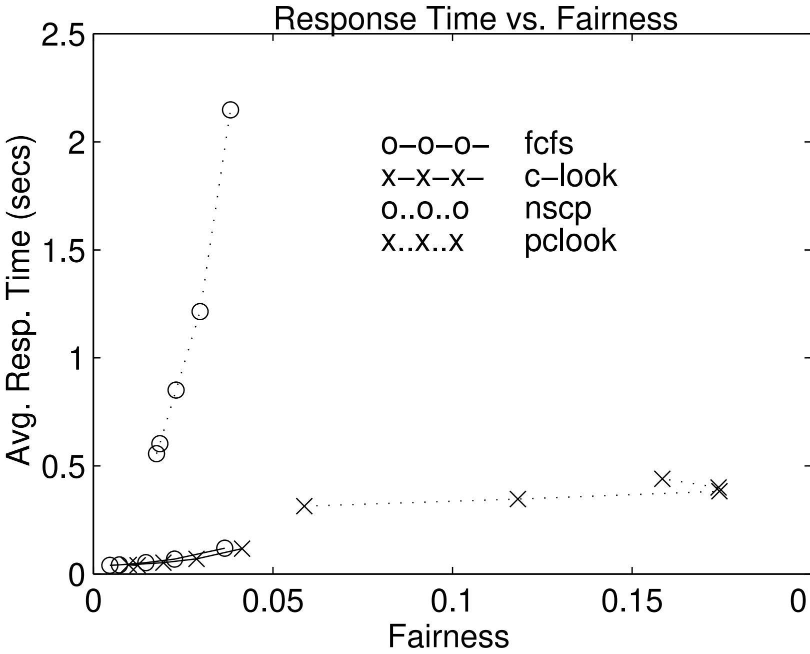


Fig. 7.11h

than the outermost ones. This is due to the following reason: Due to the kind of skew in the data request distribution, at any instant of time, the queue will contain requests only for a set of neighbouring cylinders. A request for any cylinder will arrive to find the disk head serving one of its neighbouring cylinders. For the innermost cylinders, most of their neighbours are ahead of them. Hence, these requests will typically have to wait for the disk head move to the outer edge and then rewind to the inner one before being served. This results in higher cylinder response times for these cylinders. Similarly, requests for the outermost cylinders typically arrive when the disk head is serving some cylinder behind these cylinders. Therefore, these requests will be served in the current sweep. Hence, they have a smaller mean response time. Requests for other cylinders will have equal probability of finding the head serving cylinders ahead of it as well as behind it. Due to this reason, the response time distribution is flat apart from the outermost and innermost cylinders.

The PCLOOK scheduler is found to be unfair over the entire loading range, being most unfair for medium arrival rates (7.11g). This is due to the same reasons as given for PLOOK of GATE scheduling.

It is observed that the fair algorithms of Chapter 5, PLOOK, 3SPL and PCLOOK, remain fair if the data request distribution is spatially skewed (except PCLOOK for the double spiked skewed distribution). However, if the distribution becomes temporally skewed, they no longer remain fair. This was mainly because the algorithms were constructed keeping in mind, the uniform distribution of data requests. For such skewed distributions, the classical algorithms themselves were found to be fair with low mean response times. This indicates the need for developing *adaptive scheduling algorithms*, which would use some kind of a feedback system to determine the kind of skew in the data request distribution and alter the scheduling policy accordingly.

7.3 Simulation Model

As described in Chapter 3, an open simulation model for the disk system was used for all the experiments. In the open model, the request leave the disk system as soon as they are served. A more realistic model for a multiuser environment would be the *closed model* where there are a number of users, specified by *NumUsers*, submitting requests. On completion of the service of a request, the users "think" for some time before submitting the next request. The think time distribution is exponential, with the mean value specified by *ThinkTime*. As in the open model, requests wait for service in a single queue. The data requests for each user is confined to a few consecutive cylinders, specified by the *Range* parameter and within this range, they are uniformly distributed. The parameters and their values for the closed model are given in Table 7.2.

Parameter	Values
$NumCyl$	1000
$Alpha$	30 msec.
$Beta$	0.5 msec./cylinder
$Access$	8 msec.
$NumUser$	200
$ThinkTime$	10 - 120 sec.
$Range$	5 cylinders

Table 7.2: Parameters for Closed Model

Using this model, the improved fair algorithm for each of the scheduling granularities was simulated. Also, for comparison, simulations for FCFS and one other classical algorithm (CLOOK) were done. The results of these experiments evaluating the performance of these algorithms are shown in Figures 7.12a - 7.12h (for GATE scheduling), Figures 7.13a - 7.13h (for STOP scheduling) and Figures 7.14a - 7.14h (for DYNAMIC scheduling). Note that now, we plot think time instead of arrival rate. Figure 7.15 shows the arrival rate as a function of think time.

The values for this graph remains nearly the same for all the algorithms simulated. The value of the think time used for the simulations correspond to arrival rates which

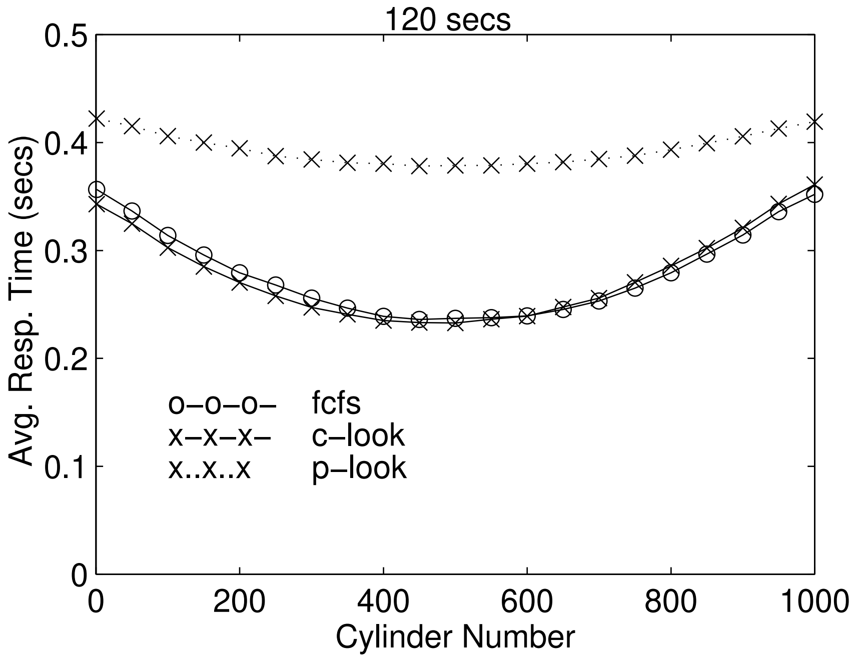


Fig. 7.12a

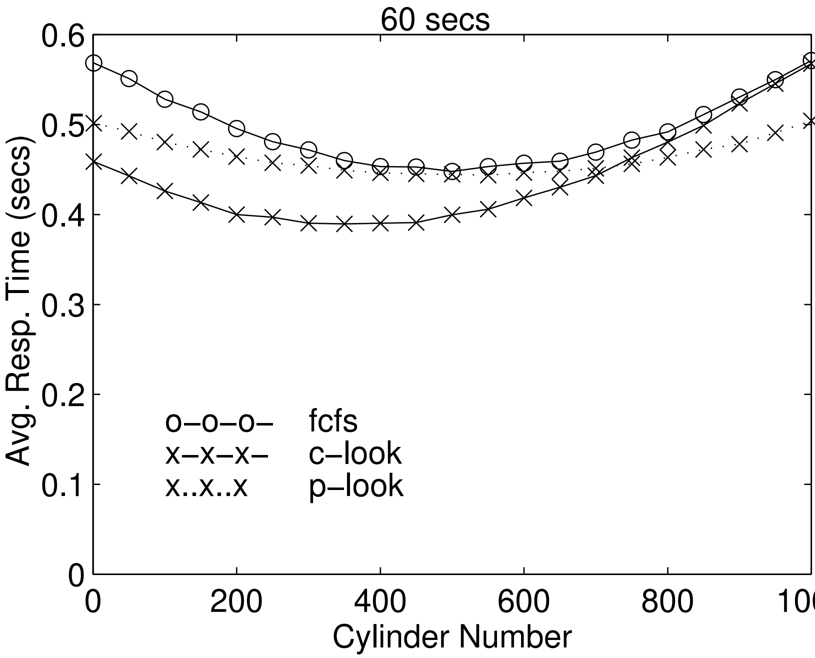


Fig. 7.12b

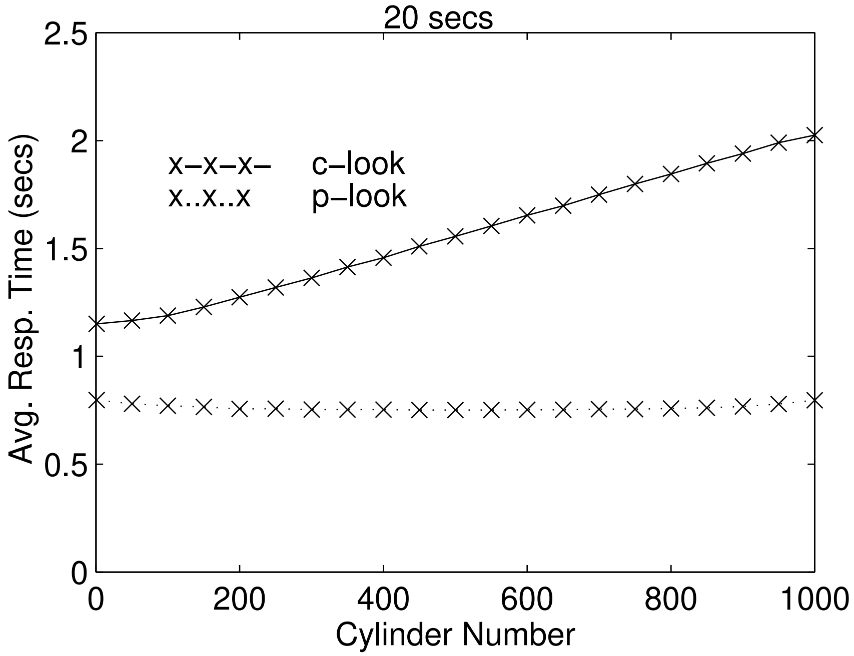


Fig. 7.12c

GATE Algorithms

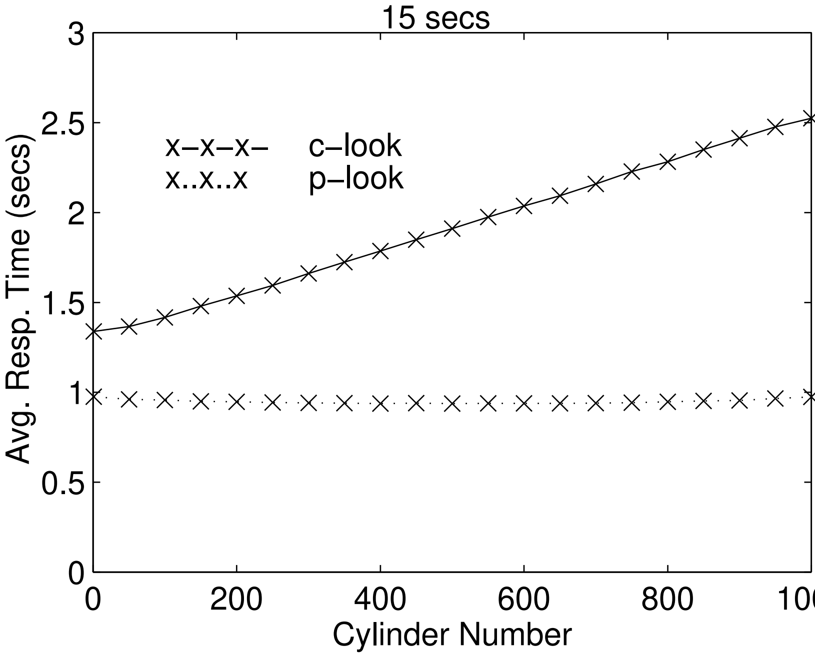


Fig. 7.12d

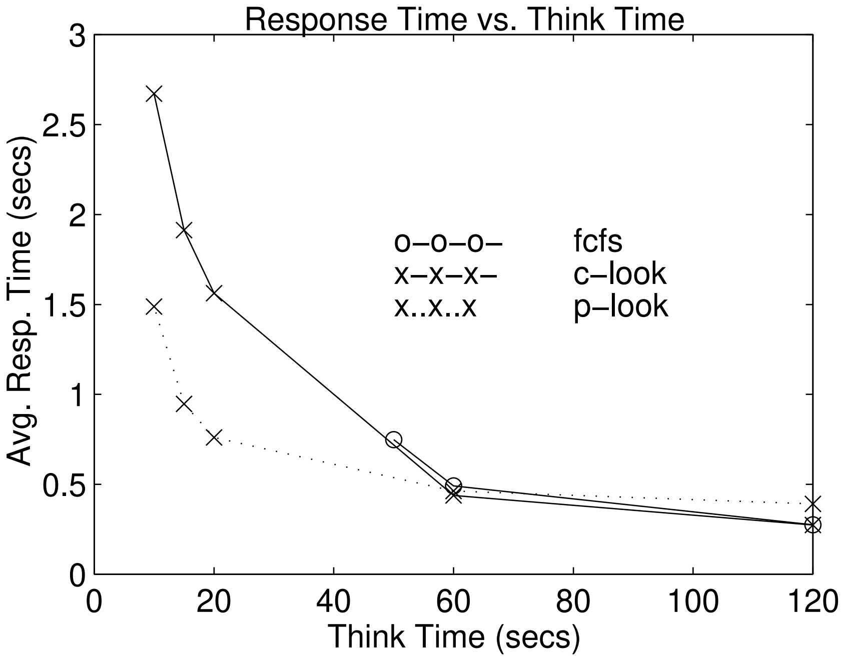


Fig. 7.12e

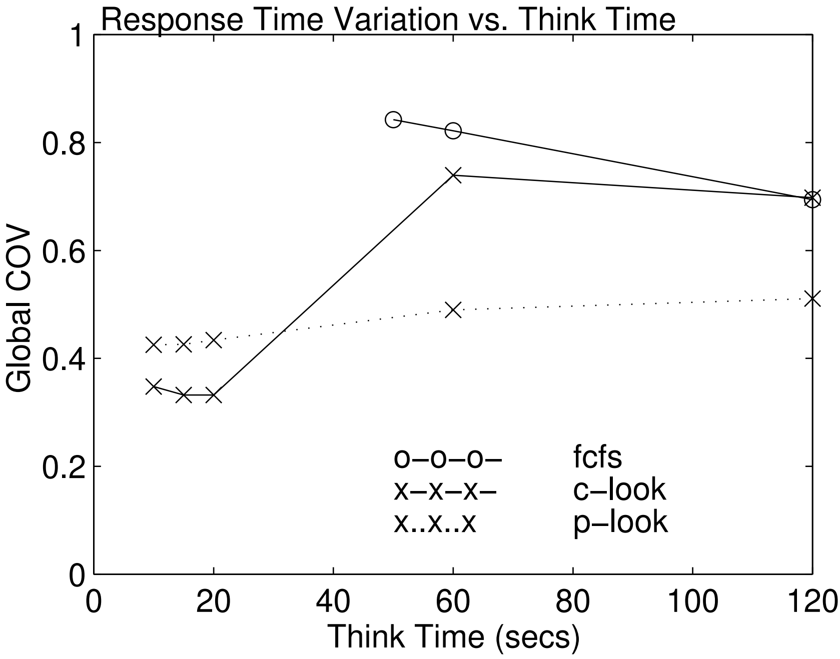


Fig. 7.12f

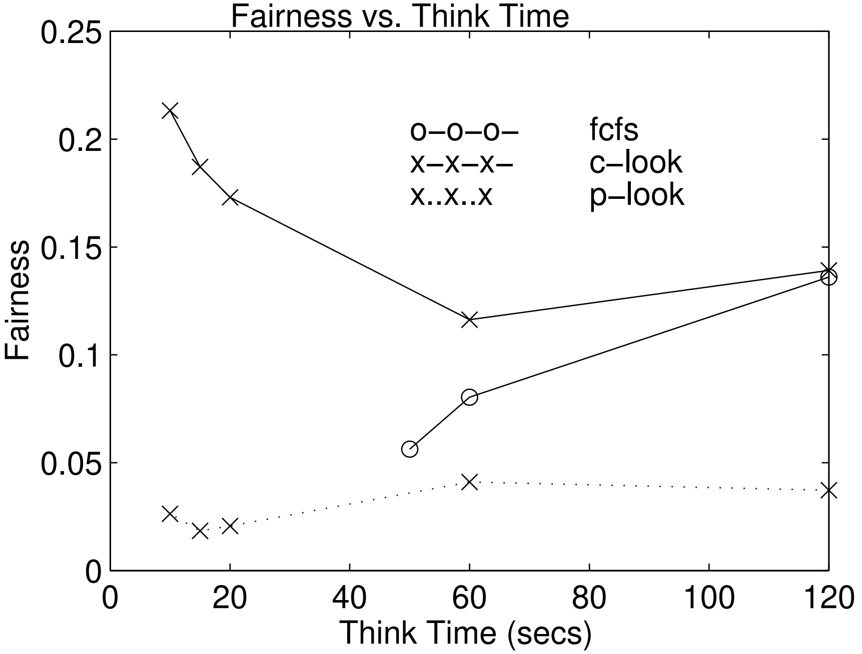


Fig. 7.12g

GATE Algorithms

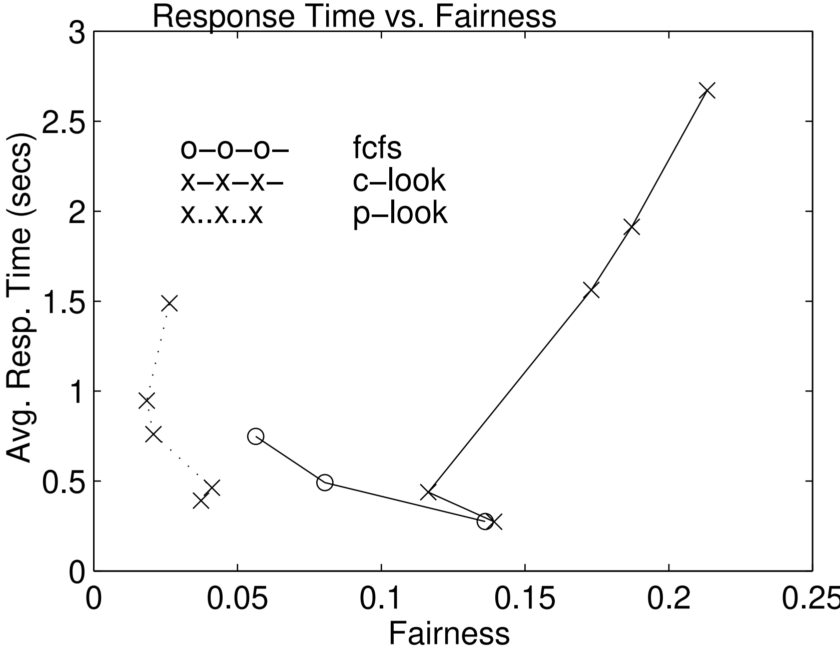


Fig. 7.12h

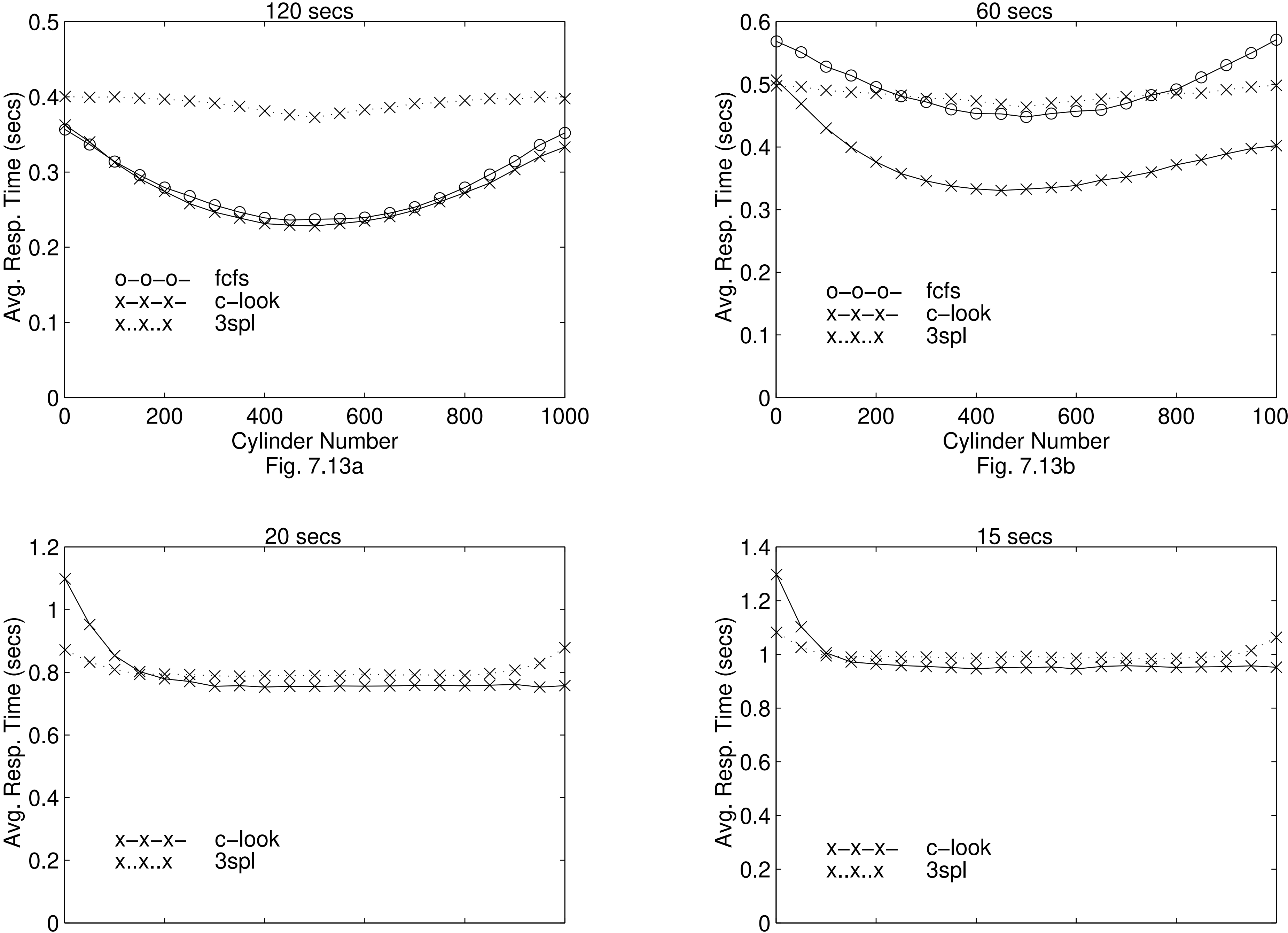


Fig. 7.13a

Fig. 7.13b

Fig. 7.13c

Fig. 7.13d

STOP Algorithms

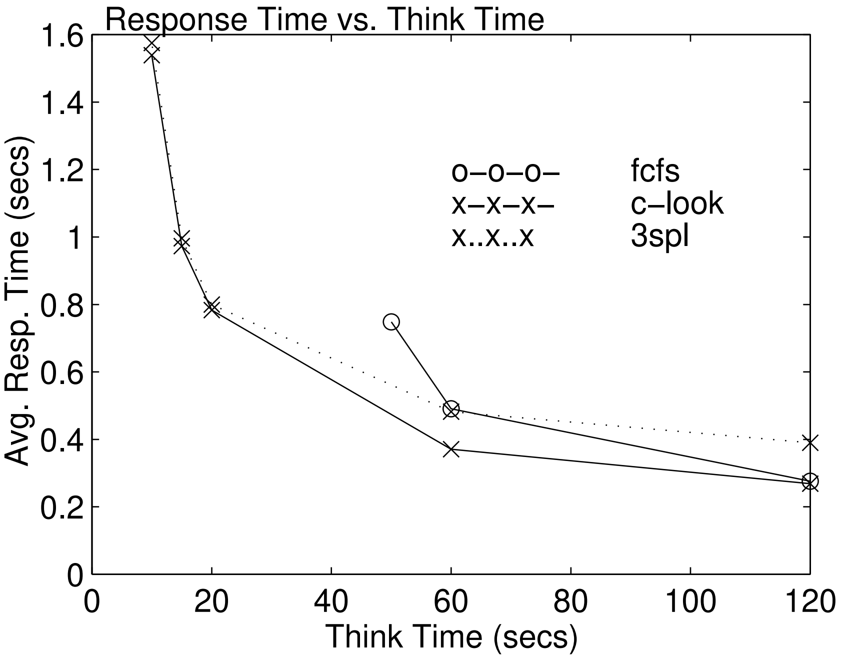


Fig. 7.13e

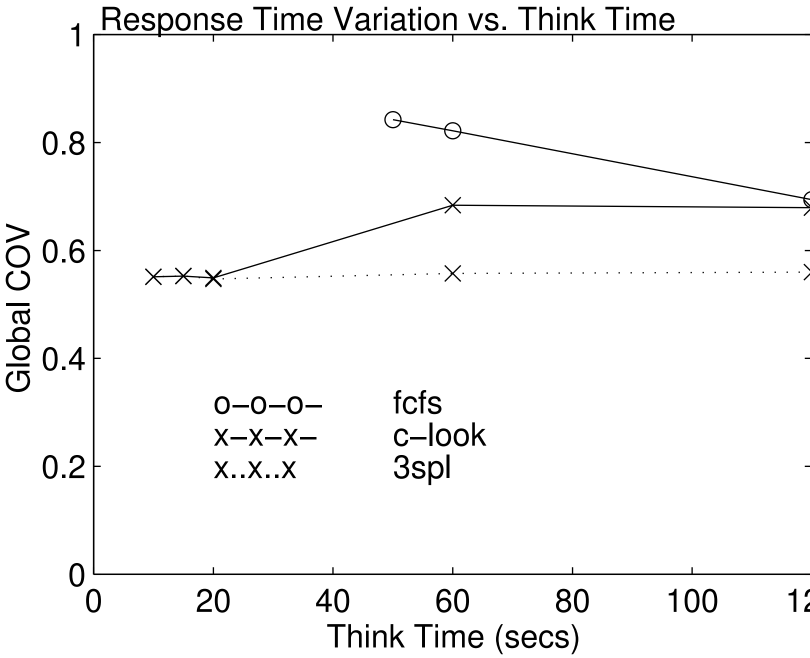


Fig. 7.13f

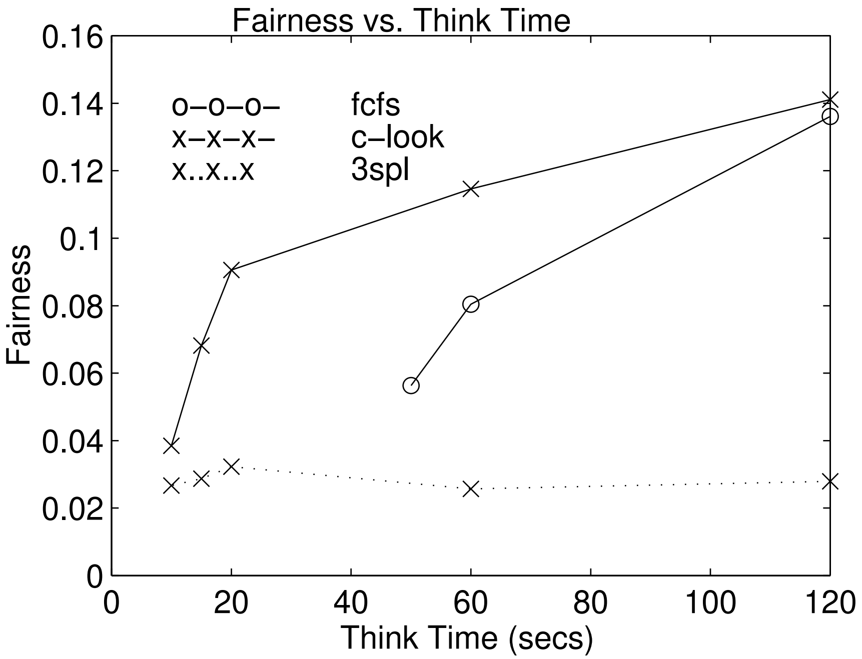


Fig. 7.13g

STOP Algorithms

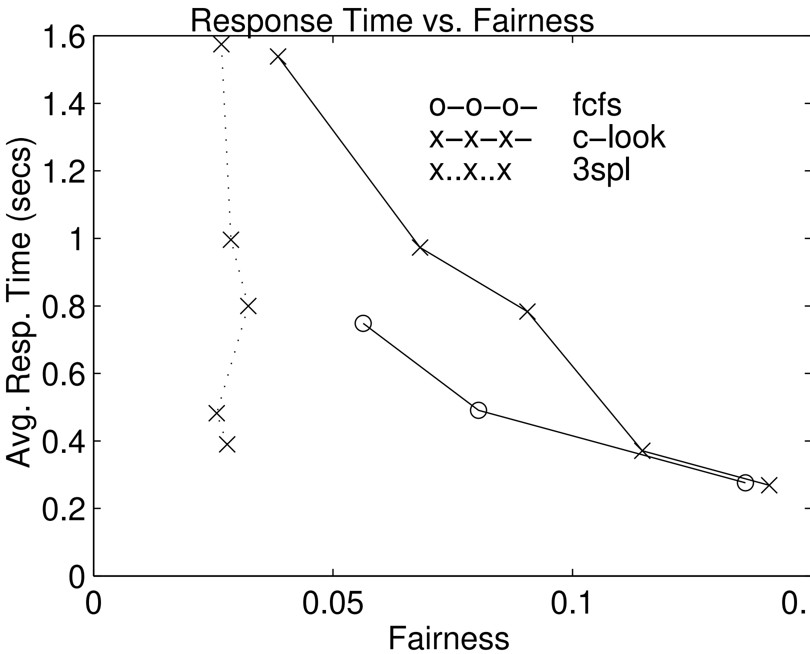


Fig. 7.13h

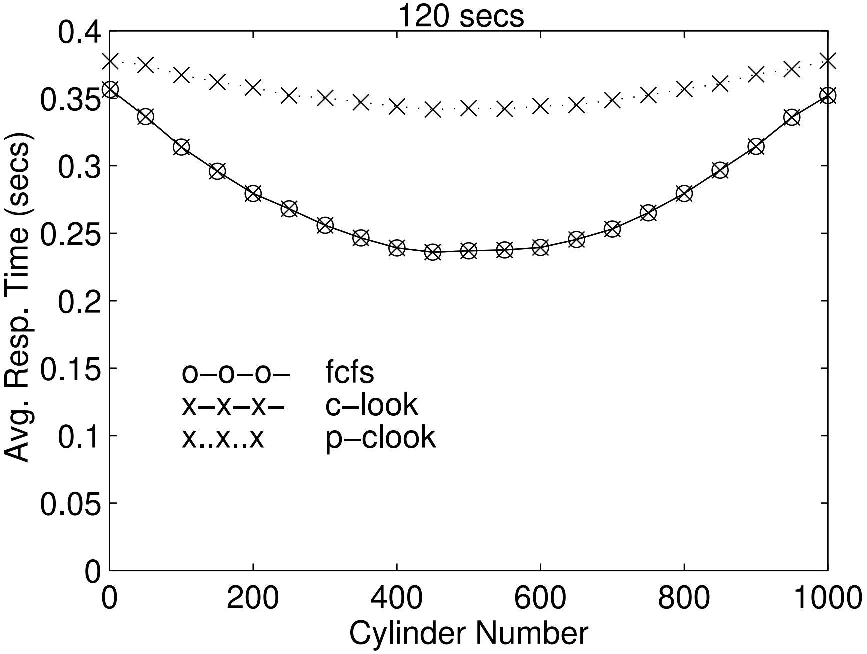


Fig. 7.14a

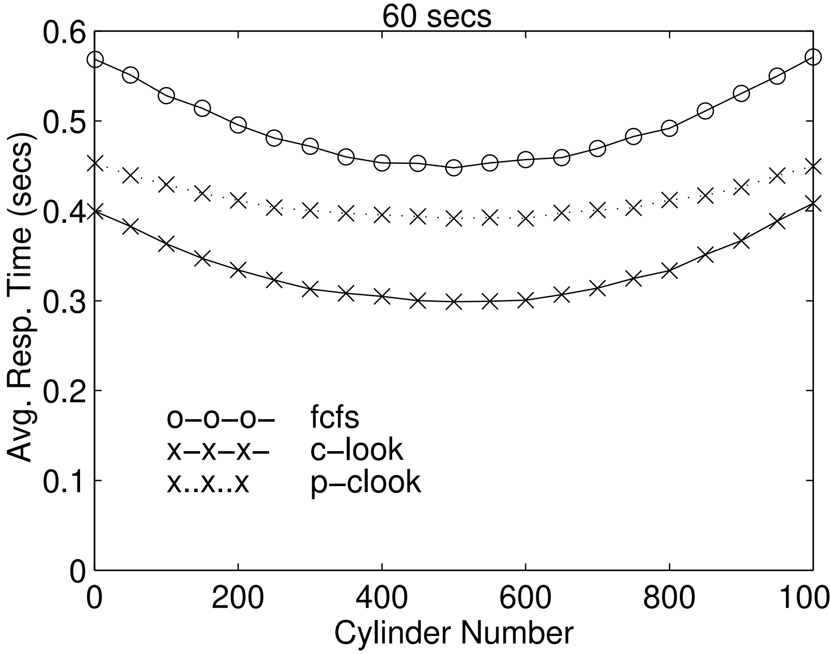


Fig. 7.14b

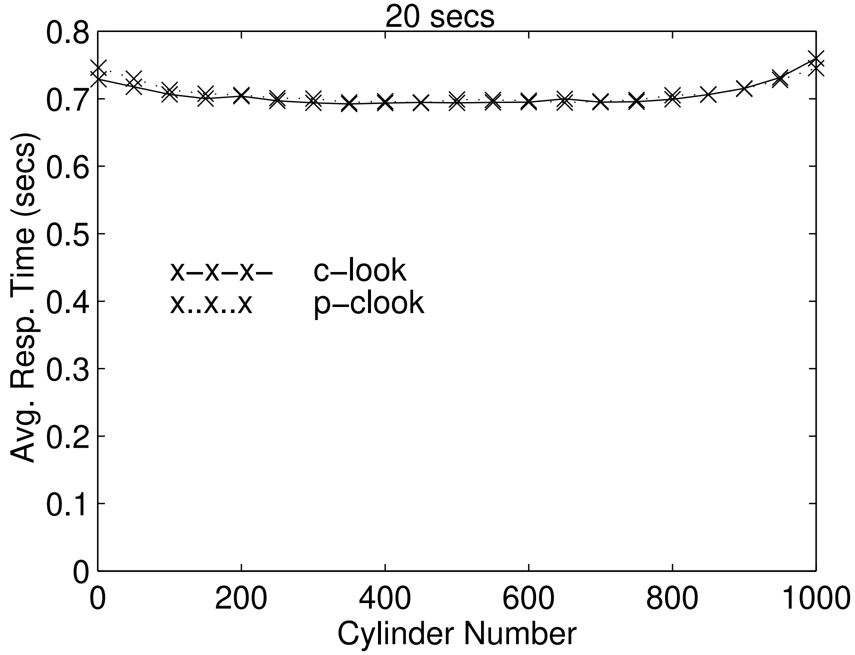


Fig. 7.14c

DYNAMIC Algorithms

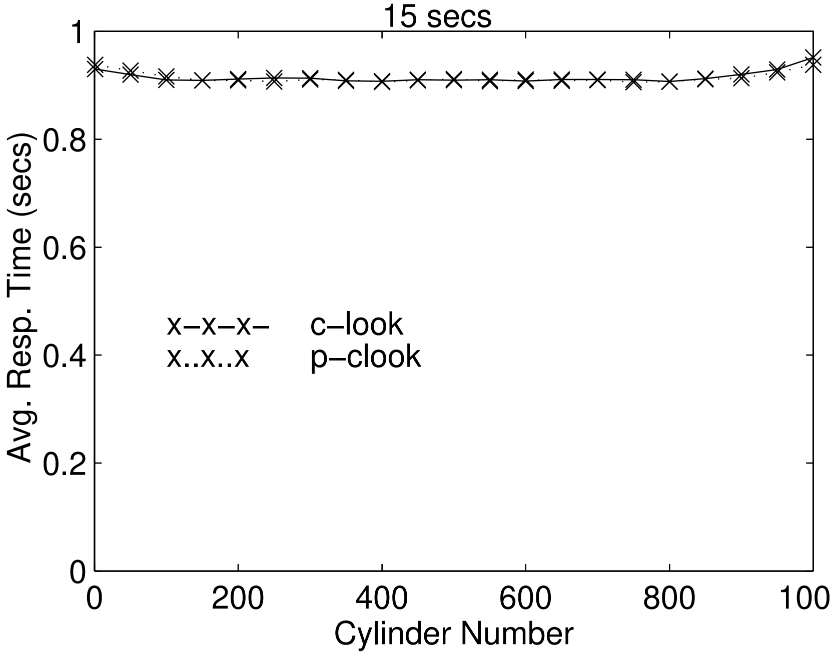


Fig. 7.14c

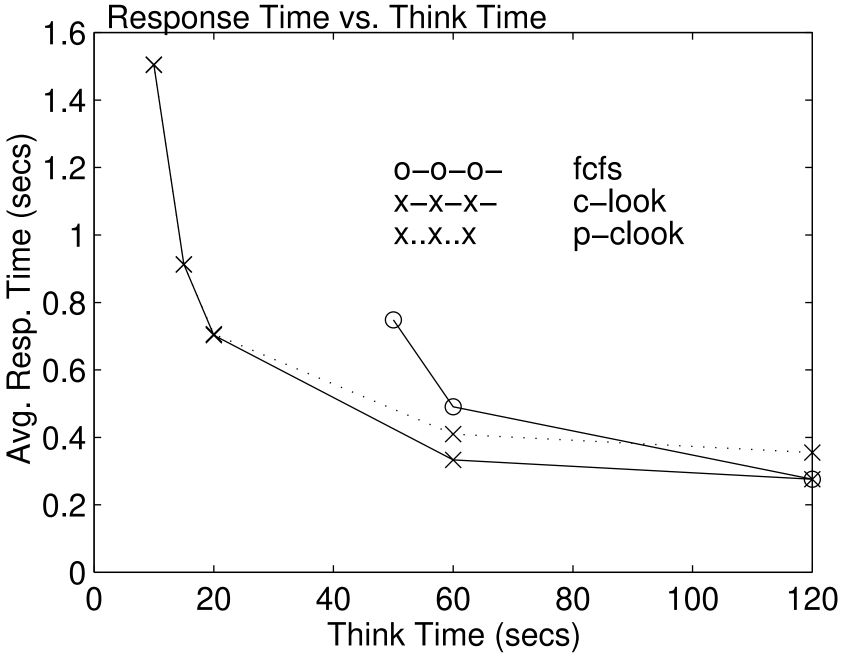


Fig. 7.14e

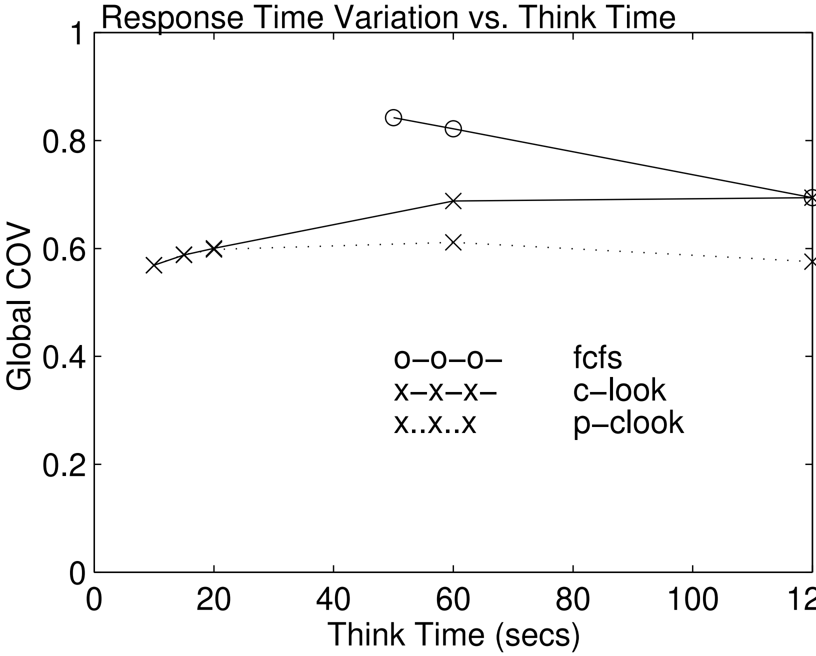


Fig. 7.14f

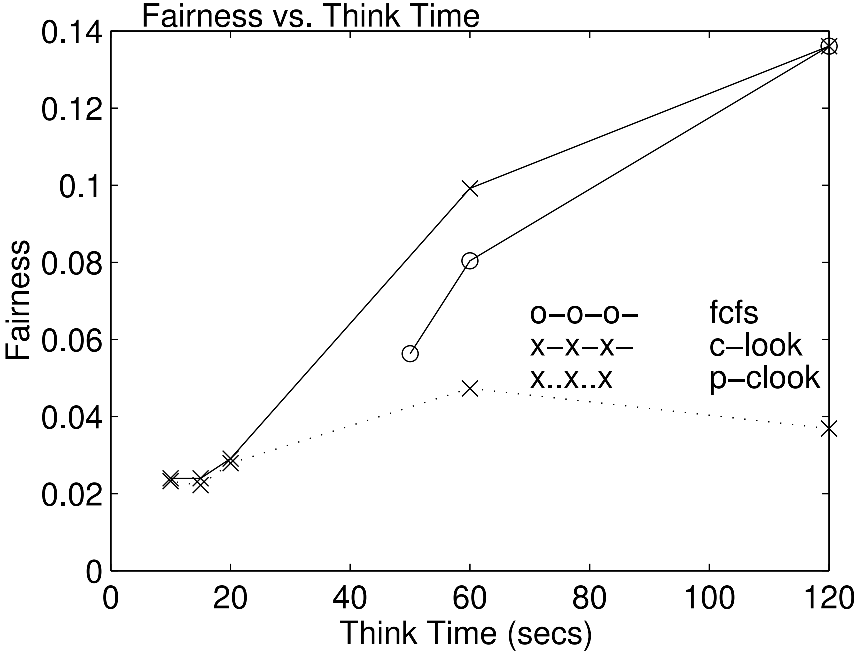


Fig. 7.14g

DYNAMIC Algorithms

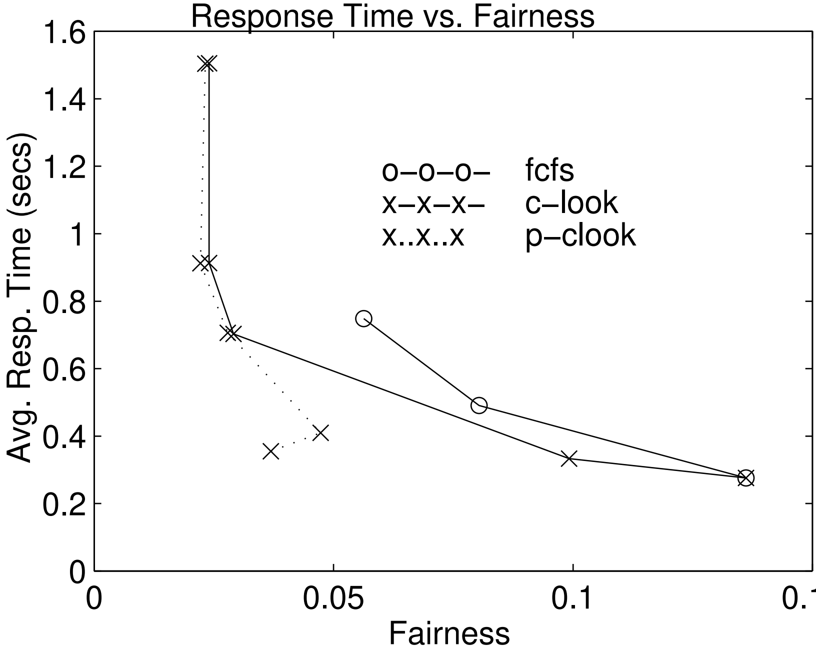


Fig. 7.14h

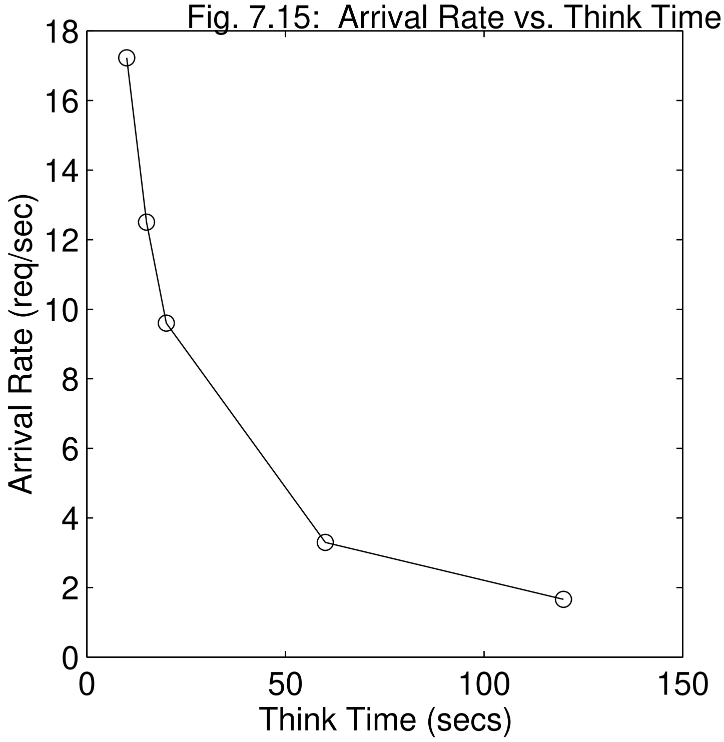


Fig. 7.15: Arrival Rate vs. Think Time

are close to the ones used for the earlier experiments, using the open model. This was done so that the figures for the closed model could be directly compared to the ones of the open model.

From these figures, and the ones in Chapter 5, it is observed that the qualitative performance of the algorithms do not change significantly from that for the open model.

7.4 Arrival Process

So far, in all the earlier experiments, we have looked at the the disk system independently, attempting to improve its performance. However, it may be more reasonable to include the CPU in the model and then study the performance. The new model is described as follows: The system consists of a CPU unit and an I/O unit. Each system request consists of a CPU request and an I/O request to be served in that order. This is pictorially shown in Figure 7.16. The arrival process for the overall system is Poisson. The CPU service times are exponentially distributed and the requests are served in the CPU unit using Round-Robin scheduling. The parameters and their values used in the simulation are

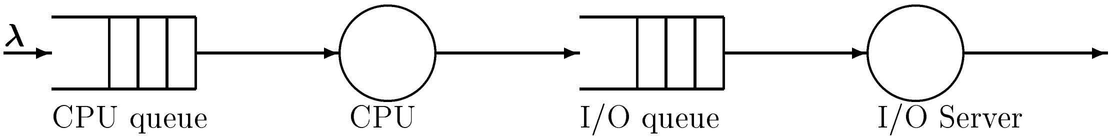


Figure 6.1: System Model

listed in Table 7.3. Due to this variation in the system model, the arrival process to the disk system may change from the Poisson stream used in earlier simulations.

Parameter	Values
$ServiceRate$	25 requests/sec.
$QuantumSize$	0.4 msec.
$ContextSwitchTime$	0.004 msec.

Table 7.3: CPU Parameters

Thus, the input process to the I/O subsystem is the output process of the CPU unit. This is the same as the output process of a $M/M/1$ queue where the server uses Round-Robin scheduling. This queue was simulated to determine its output distribution. Using techniques such as correlation plots, histograms and P–P plots, it was found out that the distribution for the interarrival times of the output process was close to exponential. It had the the same mean as that of the input stream. Hence this model is equivalent to the model of Chapter 3 and therefore will perform similarly.

Another variation of the arrival process that was tried, was to use the *erlang(k)* (where $k = 2$) distribution for the interrarrival times of the requests. This will result in smaller interarrival times being less probable as compared to those which are exponentially distributed. The density function for erlang(2) distribution is given by

$$f(x) = \mu^2 x e^{-\mu x} \tag{7.1}$$

The performance of all the three improved algorithms of Chapter 5 (PLOOK, 3SPL and PCLOOK) remain qualitatively similar to those presented in Chapter 5.

Chapter 8

Conclusions

In this report, the problem of ensuring fair service in disk schedulers was addressed. We defined a new fairness metric, the coefficient of variation of the cylinder mean response times. A scheduler with fairness value of 0.0 is defined to be completely fair. For this metric, a detailed simulation study was made of classical disk scheduling algorithms such as FCFS, SSTF, POLL, LOOK, CPOLL, and CLOOK, for the GATE, STOP and DYNAMIC scheduling granularities. An open model with Poisson arrival process was used for simulating the disk system. The data requests were uniformly distributed across the entire disk. Our experiments showed that, for all the scheduling granularities, the classical algorithms fail to provide adequate fairness over the entire loading range. To address this problem, a new scheduling algorithm was proposed for each scheduling granularity. These were PPOLL, PCPL and NSCP for GATE, STOP and DYNAMIC scheduling, respectively. The experimental results showed these algorithms to be completely fair over the entire loading range in that their fairness value never exceeded 0.05. The new algorithms did not reduce the global variation in user response times, but instead ensured that this variation was evenly distributed.

It was observed that the fairness for the above new algorithms was achieved at the expense of a significant increase in the mean response times. For GATE scheduling, the increase was significant only for lighter loads. However, for STOP and DYNAMIC scheduling, there was an increase for the entire loading range, the increase being more for lighter loads. To address this problem, improved algorithms were constructed for each granularity. These were PLOOK, 3SPL and PCLOOK for GATE, STOP and DYNAMIC scheduling, respectively. They reduce the mean response times at the expense of a slight deterioration in fairness values. However, the fairness value still remains less than 0.05, which was considered to be satisfactory.

More fundamentally, it was proved that in a disk system, fairness cannot be achieved by redistributing the cylinder mean response times of an algorithm which minimizes the overall mean response time. It was shown that, an optimal fair algorithm is the one which makes the cylinder mean response times of all cylinders equal to that of the extreme cylinders of an algorithm which minimizes the overall mean response time. Thus, a lower bound on the mean response time for fair algorithms was determined. It was found that the mean response time of the improved algorithms, PLOOK, 3SPL and PCLOOK came close to this bound.

The robustness of the fair algorithms was examined by varying the system parameters, model and workload. The simulations were done with the disk parameters of a recent disk ($HPC2200$). A closed model for the disk system was used to realistically model the multiuser environment. These variations did not effect the performance of the fair algorithms qualitatively. The arrival process for the disk system was also varied by using a model which included the CPU. Other distributions like $erlang$, were also used to generate the interarrival times of the requests. Again, the fair algorithms remained fair. Instead of uniform distribution for the data requests, simulations were done with skewed distributions for data requests. The algorithms remained fair for the spatially skewed disk accessses (for example, the single and the double spiked skew), but became unfair when the data requests exhibited temporal locality (as in the moving gaussian distribution). For these kind of skewed distributions, the classical algorithms themselves were found to be fair. This indicated the need for developing adaptive scheduling algorithms which would alter the scheduling policy based on feedback information about the data request distribution.

We now discuss fairness with respect to disk arrays.

Fairness In Disk Arrays (RAID)

So far, the fairness issue has been discussed keeping in mind a single disk module. In the recent years, Redundant Arrays of Inexpensive Disks (RAID) have become commercially popular. This is due to the need for high performance secondary storage systems to keep in pace with the development of faster microprocessors and larger primary memory systems.

Disk arrays consists of multiple independent disks working as a large high performance logical disk. The two major concepts used in redundant disk arrays are: data striping for improved performance and data redundancy for improved reliability. There are various RAID organisations which implement these concepts in different ways ([Chen94]).

We now discuss whether implementing the fair algorithms of Chapter 5 in each disk of the disk array will achieve overall fairness. The overall fairness of a disk array can be defined as follows: Since some RAID organisations treat read and write requests differently, a disk array is considered to be fair if all users have approximately the same average read waiting times independent of where on the disk array their files are located. The average write waiting times should also be nearly the same for all users. However, the mean read waiting time need not be the same as the mean write waiting time. Waiting times are used instead of response times, since the requests size is not the same always and we are interested in the delay caused before the actual transfer of the data begins. Note that unlike the request modeling in single disk module, here the requests are modeled explicitly as read or write. Also, their sizes are taken into account. This is because the number of disk accesses for reads maybe different from that for writes (due to data redundancy). And, the accesss pattern also depends on the size of the requests (due to data striping). This was not the case in single disk module.

The data requests are assumed to be uniformly distributed over the entire array of disks. Also the requests have equal probability of being reads and writes. The important RAID organisations and their fairness is discussed separately below.

Nonredundant (RAID Level 0)

A nonredundant disk array does not employ redundancy and implements block level data striping. Due to the absence of redundancy, read and write requests are treated similarly. Small requests will need to access only a small number of disks whereas larger ones may access all the disks in the array. Since independent disks can be accessed parallelly, we can say that the disk array is fair if the cylinders mean waiting times for all cylinders and all disks are approximately the same. Hence, if all disks are fair independently and the mean waiting time for all the disks are nearly the same, then the disk array can be considered to be fair. Since all the disks will be accessed equally and similarly, the mean waiting time for all disks will be nearly the same. And if the fair algorithms of Chapter 5 are implemented on each disk, then all the disks will be independently fair. Hence, the disk array will show overall fairness under such conditions.

Mirrored (RAID Level 1)

This organisation uses twice as many block interleaved disks as a nonredundant disk array. Whenever data is written to a disk, the same data is also written to a redundant disk. When data is read, it is retrieved from the disk with the shorter waiting delay. Here, if every pair, consisting of a disk and its duplicate, shows fairness, then the whole array will be fair. To compute the fairness for a pair of disks, the cylinder waiting time should be averaged over both the disks. Due to the uniform distribution of accesses, this will lead to a uniform distribution of data requests for every disk. Hence, if the fair algorithms of Chapter 5 are implemented on each disk, then as explained for the nonredundant disk array, this disk array will also show overall fairness.

Bit-Interleaved Parity (RAID Level 3)

This is a bit-interleaved parity disk array where data is conceptually interleaved bit-wise over all the disks and a single parity disk is added to recover from any single disk failure. Due to bit-interleaving, all requests, regardless of their size, access all the data disks in the array. The parity disk is involved only in the write requests. All data disks will be accessed equally and similarly. If the algorithms of Chapter 5 are implemented on each disk, all of them will be fair independently. All the data disks will also have the same mean waiting time. The mean waiting time for the parity disk may be different from that of the data disk and will depend upon the write requests distribution. Since a write request consists of data disk access followed by parity disk access, the mean waiting time for writes may not be equal to that for reads. Since this is not necessary for overall fairness of a disk array, and all cylinder mean read waiting times will be nearly the same and all cylinder mean write waiting times will also be approximately equal, the disk array will show overall fairness.

Block-Interleaved Parity (RAID Level 4)

This is similar to the bit-interleaved parity disk array except that the data is interleaved across disks in blocks instead of bits. As in bit-interleaved parity disk array, this will also show overall fairness for the same reasons as explained above.

Block-Interleaved Distributed Parity (RAID Level 5)

It is similar to block-interleaved parity disk array except that the parity is now uniformly distributed over all the disks. Hence all the disks will be accessed similarly. The presence of parity in all the disks will result in a non-uniform distribution of the requests for each disk. Since the parity blocks are accessed more, the data request distribution will be qualitatively similar to a single spike distribution, for which the fair algorithms of Chapter 5 remain fair (Section 7.2.1). Hence if those algorithms are implemented on all disks, they will all be independently fair. Also, as all of them will have nearly equal mean waiting time, the disk array will show overall fairness.

Bibliography

[Bach86] Bach, M., *The Design of the UNIX Operating System*, Prentice Hall, 1986.

[Chen94] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., Patterson, D. A., "RAID: High-Performance, Reliable Secondary Storage", *ACM Computing Surveys*, 26(2), June 1994.

[Coff73] Coffman, E., and Denning, P., *Operating Systems Theory*, Prentice-Hall, 1973.

[Coff82] Coffman, E., and Hofri, M., "On the Expected Performance of Scanning Disks", *SIAM J. Computing*, 11(1), February 1982.

[Coff86a] Coffman, E., "Queuing Models for Secondary Storage Devices", *Proc. IFIP Congr. 1986*, North Holland Pub. Co., Amsterdam.

[Coff86b] Coffman, E., and Hofri, M., "A Continuous Polling System with Constant Service Times", *IEEE Trans. on Information Theory*, 32(4), July 1986.

[Coff87] Coffman, E., and Gilbert, E., "Polling and Greedy Servers on a line", *Queueing Systems: Theory and Applications*, 2(2), 1987.

[Dani83] Daniel, S., and Geist, R., "V-SCAN: An Adaptive Disk Scheduling Algorithm", *Proc. of IEEE Int. Symp. on Computing Systems Organization*, March 1983.

[Deme90] Demers, A., Keshav, S., Shenker, S., "Analysis and Simulation of a Fair Queuing Algorithm", *Internetworking Research and Experience*, June 1993.

[Denn67] Denning, P., "Effects of Scheduling on File Memory Operations", *Proc. AFIPS SJCC*, 31(1967).

[Fran69] Frank, H., "Analysis and Optimization of Disk Storage Devices for Time-Sharing Systems", *Journal of the ACM*, 16(4), October 1969.

[Geis87a] Geist, R., and Daniel, S., "A Continuum of Disk Scheduling Algorithms", *ACM Trans. on Computer Systems*, February 1987.

[Geis87b] Geist, R., Reynolds, R., and Pittard, E., "Disk Scheduling in System V", *Proc. of ACM Sigmetrics Conference*, May 1987.

[Gopa88] Gopalakrishnan, R., "An Analytic Model for UNIX Systems", *AT & T Technical Journal*, September-October 1988.

[Gotl73] Gotlieb, C., and MacEwen, G., "Performance of Movable-Head Disk Storage Devices", *Journal of the ACM*, 20(4), October 1973.

[Gree92] Greenberg, A. G. and Madras N., "How Fair is Fair Queuing?", *Journal of ACM*, 39(3), July 1992.

[Henr84] Henry, G., "The UNIX System: The Fair Share Scheduler", *AT & T Bell Systems Technical Journal*, 63(8), October 1984.

[Hofr80] Hofri, M., "Disk Scheduling: FCFS Vs. SSTF Revisited", *Comm. of ACM*, 23(11), November 1980.

[IEEE94] *IEEE Computer, Special Issue – The I/O Subsystem: A Candidate for Improvement*, March 1994.

[King90] King, R. P., "Disk Arm Movement in Anticipation of Future Requests", *ACM Trans. on Computer Systems*, 8(3), 1990.

[Klei75] Kleinrock, L., *Queueing Systems, Vol. 1*, John Wiley & Sons, 1975.

[Law77] Law, A. M., Carson, J. S., "A Sequential Procedure for Determining the Length of a Steady State Simulation", *Tech. Report, Dept. of Industrial Engineering, University of Wisconsin, Madison*, 1977.

[Mert70] Merten, A., “Some Quantitative Techniques for File Organization”, *Ph.D. Thesis*, Univ. of Wisconsin, 1970.

[Oney75] Oney, C., “Queueing Analysis of the Scan Policy for Moving-Head Disks”, *Journal of the ACM*, 29(3), July 1975.

[Raj93] Kumar, R., “Fairness in Disk Scheduling”, *Master’s Thesis, Dept. of Computer Science and Automation, Indian Institute of Science*, January 1993.

[Rama92] Ramakrishnan, K. K., Biswas, P., Karedia, R., “Analysis of File I/O Traces in Commercial Computing Environments”, *Performance Evaluation Review*, 20(1), June 1992.

[Ruem94] Ruemmler, C., Wilkes, J., “An Introduction to Disk Drive Modelling”, *IEEE Computer*, March 1994.

[Seam66] Seaman, P., Link, R., and Wilson, T., “An Analysis of Auxiliary Storage Activity”, *IBM Systems Journal*, 5(3), 1966.

[Shoh92] Shoham, R., and Yechiali, U., “Elevator-Type Polling Systems”, *Proc. of ACM Sigmetrics Conference*, June 1992.

[Silb92] Silberschatz, A., Peterson, J., and Gavin, P., *Operating System Concepts*, 3rd ed., Addison-Wesley, 1992.

[Swar80] Swartz, G., “Polling in a LOOP system”, *Journal of the ACM*, 27(1), January 1980.

[Tane87] Tanenbaum, A., *Operating Systems: Design and Implementation*, Prentice Hall, 1987.

[Teor72] Teorey, T., and Pinkerton, T., “A Comparative Analysis of Disk Scheduling Policies”, *Comm. of the ACM*, 15(3), March 1972.

[Triv82] Trivedi, K., *Probability & Statistics with Reliability, Queueing, and Computer Science Applications*, Prentice Hall, 1982.

[Wein68] Weingarten, A., “The Analytical Design of Real-Time Disk Systems”, *Proc. IFIP Congr. 1968*, North Holland Pub. Co., Amsterdam, pp. 131-137.

[Wilh76] Wilhelm, N., “An Anomaly in Disk Scheduling: A Comparison of FCFS and SSTF Using an Empirical Model for Disk Accesses”, *Comm. of the ACM*, 19(1), January 1976.

[Wilh77] Wilhelm, N., “A General Model for the Performance of Disk Systems”, *Journal of the ACM*, 24(1), January 1977.

[Wort94] Worthington, L., Ganger, G., and Patt, Y., “Scheduling Algorithms for Modern Disk Drives”, *Proc. of ACM Sigmetrics Conference*, June 1994.

Appendix A

Analytical Results

We describe below a mathematical model which is then used to analyse the algorithms, NSCP of DYNAMIC scheduling and PPOLL of GATE scheduling.

The set of discrete cylinder addresses is replaced by the set of real numbers in the interval $[0, N]$ where N is the total number of cylinders. As in the simulation, we assume a Poisson stream of arrivals and uniform distribution of data requests. Let the arrival rate be λ. The probability that any arrival requests for cylinders in the interval $(x, x + \Delta x)$ is given by $\Delta x / N$. We use the same parameters α, β and $Access$ (referred to here as A) for startup, seek-distance multiplier and rotational latency plus transfer time respectively. Since the arrival of requests forms a Poisson stream, the probability density function for the number of requests (n) in time interval $(0, t)$ is given by

$$f_n(t) = \frac{[\lambda t]^n}{n!} e^{-\lambda t} \tag{A.1}$$

with mean value as λt. We assume that if n requests are served in the crossing of the disk head, then the crossing time is given by $nA + N\beta$.

Our objective here is to find an expression for the mean waiting time for requests as a function of the cylinder number, x.

A.1 Non-Stop Circular Poll (NSCP)

From the description of NSCP in Chapter 4, we know that in this algorithm the disk head serves unidirectionally and continues to poll the disk even when there are no requests in the system.

Let us assume that the service crossings of the disk head are from left to right. Let $y_n(x)$ denote the random variable whose value is the time taken by the disk head to move a distance x from position 0 on its nth crossing. The last time prior to the nth service crossing that the disk head served the cylinder interval $(x, x + \Delta x)$ was in the $(n-1)$th service crossing. Thus given $y_{n+1}(x)$, $y_n(N)$ and $y_n(x)$, the mean number of requests serviced in $(x, x + \Delta x)$ on the $(n+1)$th crossing is $\lambda[y_n(N) - y_n(x) + y_{n+1}(x)]\Delta x/N$. Since each such request requires A time units for service and the disk head requires $\beta\Delta x$ time units to cross the interval, we have,

$$E[y_{n+1}(x+\Delta x)|y_{n+1}(x), y_n(N), y_n(x)] = y_{n+1}(x)+\beta\Delta x+\lambda[y_{n+1}(x)+y_n(N)-y_n(x)+R]A\Delta x/N \tag{A.2}$$

where R is the time taken for the disk head to rewind from the outer disk end to the inner one. Hence $R = \alpha + N\beta$.

Let $E[y_i(x)] \equiv \bar{y}_i(x)$. Removing the conditioning in A.2 we get,

$$\bar{y}_{n+1}(x + \Delta x) = \bar{y}_{n+1}(x) + \beta\Delta x + \lambda A[\bar{y}_{n+1}(x) + \bar{y}_n(N) - \bar{y}_n(x) + R]\Delta x/N \tag{A.3}$$

Consider the limit $\bar{y}(x) \equiv lim_{n\to\infty}\bar{y}_n(x)$. This limit exists as long as the system remains in statistical equilibrium. Taking limits in Equation A.3, we get,

$$\bar{y}(x + \Delta x) = \bar{y}(x) + \beta\Delta x + \lambda A[\bar{y}(N) + R]\Delta x/N \tag{A.4}$$

Dividing by Δx and taking limit $\Delta x \to 0$, we get,

$$\frac{d\bar{y}(x)}{dx} = \beta + \lambda A[\bar{y}(N) + R]/N \tag{A.5}$$

Solving the above we get,

$$\bar{y}(x) = [\beta + \lambda A[\bar{y}(N) + R]/N]x \tag{A.6}$$

from which we get,

$$\bar{y}(N) = \frac{N\beta + R\lambda A}{1 - \lambda A} \tag{A.7}$$

and

$$\bar{y}(x) = \frac{N\beta + R\lambda A}{N(1 - \lambda A)}x \tag{A.8}$$

Using the above expression for $\bar{y}(x)$, we get an expression for the mean waiting times for cylinder x. Consider the system at equilibrium. Suppose the head is at position v and moving right (serving requests). Let $w_r(x|v)$ denote the expected time required for the head to move from the current position v to position x. Then,

$$w_r(x|v) = \begin{cases} \bar{y}(x) - \bar{y}(v) & \text{if } x \geq v \\ \bar{y}(N) - \bar{y}(v) + R + \bar{y}(x) & \text{otherwise} \end{cases} \tag{A.9}$$

We now want to compute

$$w_r(x) = \int_0^N w_r(x|v) dP_r(v) \tag{A.10}$$

where $P_r(v)$ is the probability that the disk head is in the interval $(0, v)$ given that it is moving right. Its value will be the fraction of the time the head spends moving right that the head spends in the interval $(0, v)$. Hence,

$$P_r(v) = \frac{\bar{y}(v)}{\bar{y}(N)} \tag{A.11}$$

and the corresponding density function is,

$$p_r(v) = \frac{dP_r(v)}{dv} = \frac{d\bar{y}(v)/dv}{\bar{y}(N)} \tag{A.12}$$

On substituting into Equation A.10, we get

$$w_r(x) = \frac{1}{2}\bar{y}(N) + R(1 - x/N) \tag{A.13}$$

Now consider the system when the head is moving left (rewinding). Let $w_l(x|v)$ denote the corresponding time to move from position v to position x. Then,

$$w_l(x|v) = \frac{v}{N}R + \bar{y}(x) \tag{A.14}$$

Proceeding as before, we get an expression for $w_l(x)$,

$$w_l(x) = \frac{1}{2}R + \bar{y}(N)\frac{x}{N} \tag{A.15}$$

The mean waiting time for cylinder x

$$w(x) = p_{right}w_r(x) + p_{left}w_l(x) \tag{A.16}$$

where p_{right} is the probability that a random observer finds the disk head moving right and p_{left} is the probability that it finds the disk head moving left. On the average the time spent by the head in moving right from position 0 to position N is $\bar{y}(N)$ and in moving left from position N to 0 is R. Hence

$$p_{right} = \frac{\bar{y}(N)}{\bar{y}(N) + R} \tag{A.17}$$

and

$$p_{left} = \frac{R}{\bar{y}(N) + R} \tag{A.18}$$

On substituting these in Equation A.16, we get

$$w(x) = \frac{\frac{1}{2}[\bar{y}(N)]^2 + R\bar{y}(N) + \frac{1}{2}R^2}{\bar{y}(N) + R} \tag{A.19}$$

which is independent of x and hence the mean waiting time is the same for all cylinders.

Figure A.1 compares the mean response times determined from the analysis above with the ones obtained from the experimental results. From the graph, it is observed that under light and medium loads, the experimental response time is nearly the same as the analytical one. For high arrival rates, the analytical response time is lower than the experimental one. This discrepancy may be due to the following reasons: The analysis assumes exhaustive service policy whereas the simulation experiments implement single service policy. Also, replacement of the discrete set of cylinders by real numbers may introduce some errors.

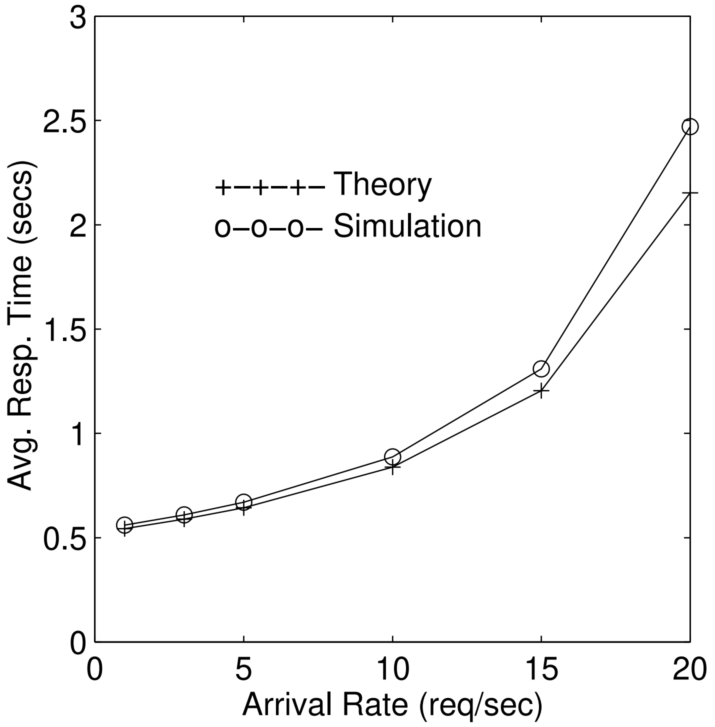


Figure A.1: Comparison of Simulation and Analytical Results for NSCP

A.2 Park Poll (PPOLL)

We now compute the waiting times for PPOLL. As described in Chapter 4, at the end of serving a group of requests, the disk head moves on to the extreme cylinder in the current direction where a new group is made which is served as the head sweeps the disk in the opposite direction.

Let us assume that the odd-numbered crossings of disk head are left to right. Hence $y_{2n+1}(x)$ denotes the random variable whose value is the time taken for the head to move a distance x from position 0 in the $(n+1)$th left to right crossing. Similarly, $y_{2n}(x)$ corresponds to the time taken for the head to move a distance of x from position N in the nth right to left crossing. The last time prior to the $(2n+1)$th crossing the interval $(x, x+\Delta x)$ received service was in the $2n$th crossing which served the group made at the beginning of the $2n$th crossing. Similarly, the requests that are served in the $(2n+1)$th crossing are from the group made at the beginning of the crossing. Thus, given $y_{2n}(N)$, the mean number of requests serviced in $(x, x+\Delta x)$ on the $(2n+1)$th crossing are $\lambda y_{2n}(N)\Delta x/N$. Hence,

$$E[y_{2n+1}(x+\Delta x)|y_{2n+1}(x), y_{2n}(N)] = y_{2n+1}(x) + \beta x + \lambda y_{2N}(N)A\Delta x/N \tag{A.20}$$

Proceeding as in NSCP, we get

$$\bar{y}_(x) = [\beta + \lambda A\bar{y}(N)]x \tag{A.21}$$

from which we get

$$\bar{y}(N) = \frac{N\beta}{1-\lambda A} \tag{A.22}$$

and

$$\bar{y}(x) = \frac{x\beta}{1-\lambda A} \tag{A.23}$$

To determine the waiting time for cylinder x, consider the system at equilibrium. Suppose the head is at position v and moving right. Let $w_r(x|v)$ denote the the waiting time for requests for position x which arrived when the disk head was at position v. As these requests will be served only after the next group is formed,

$$w_r(x|v) = \bar{y}(N) - \bar{y}(v) + \bar{y}(N-x) \tag{A.24}$$

Integrating as before, we get

$$w_r(x) = \frac{\beta}{1 - \lambda A}[\frac{3}{2}N - x] \tag{A.25}$$

Similarly, corresponding to the right to left crossings,

$$w_l(x|v) = \bar{y}(N) - \bar{y}(N - v) + \bar{y}(x) \tag{A.26}$$

and

$$w_l(x) = \frac{\beta}{1 - \lambda A}[\frac{N}{2} + x] \tag{A.27}$$

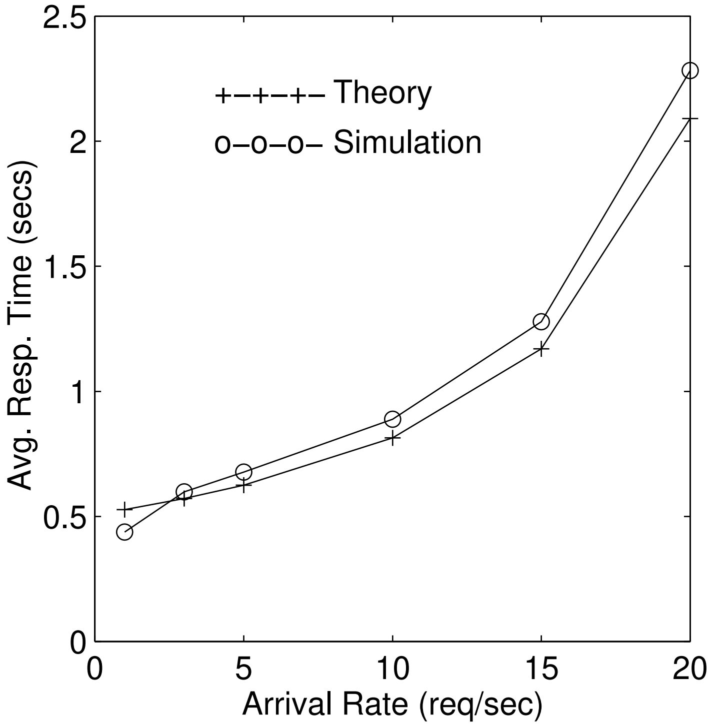


Figure A.2: Comparison of Simulation and Analytical Results for PPOLL

The mean waiting time for cylinder x is given by

$$w(x) = p_{right} w_r(x) + p_{left} w_l(x) \tag{A.28}$$

where p_{right} and p_{left} are as defined previously. Due to the symmetry of the algorithm, the probability that the disk head is found traveling in either direction is equal($\frac{1}{2}$). Thus the overall waiting time,

$$w(x) = \frac{N\beta}{1 - \lambda A} \tag{A.29}$$

which is again independent of x and hence the algorithm should be fair.

Figure A.2 compares the mean response times determined from the analysis above with the ones obtained from the experimental results. From the graph, it is observed that for low arrival rates, the experimental response time is lower than the analytical one. This is because, the analysis does not model the parking of the head when it is idle. Parking at the extreme cylinders lowers the mean response time and this effect is dominant at low arrival rates. This is because at such light loads, the head is often found to be idle. As explained for NSCP, the discrepancy between the mean response times determined using simulation and analysis may be due to the assumptions made while doing the analysis.

Appendix B

M/M/1 Queue with Round-Robin Scheduling

Our aim here is to determine the output process of a $M/M/1$ queue where the server uses Round-Robin scheduling to serve requests. The quantum size is constant and the context switch time is negligible when compared to the quantum size. It is assumed that when a request completes service, the next quantum starts at min*(completion time of request, end of quantum)*. We claim that the output process of a queue as the one described above is bounded by the Poisson Process (limit as the context switch time tends to zero).

Proof:
It is known that the output process of a $M/M/1$ queue with FCFS scheduler is Poisson. Let $RR(c)$ represent a Round-Robin scheduler with constant quantum size and context switch time, c. We first look at a sample path of a $M/M/1$ queue with $RR(0)$ scheduler.

Consider the system at a point when there are two requests, T_1 and T_2 in the system (T_i arrived before T_j iff $i < j$). In Figure B.1 the *number of requests in the system* is plotted against *time* (which is in units of quantum size). The following time instants are described below:

0 : T_1 is scheduled.

1 : T_1 has not completed service at the end of the quantum. Since there are other

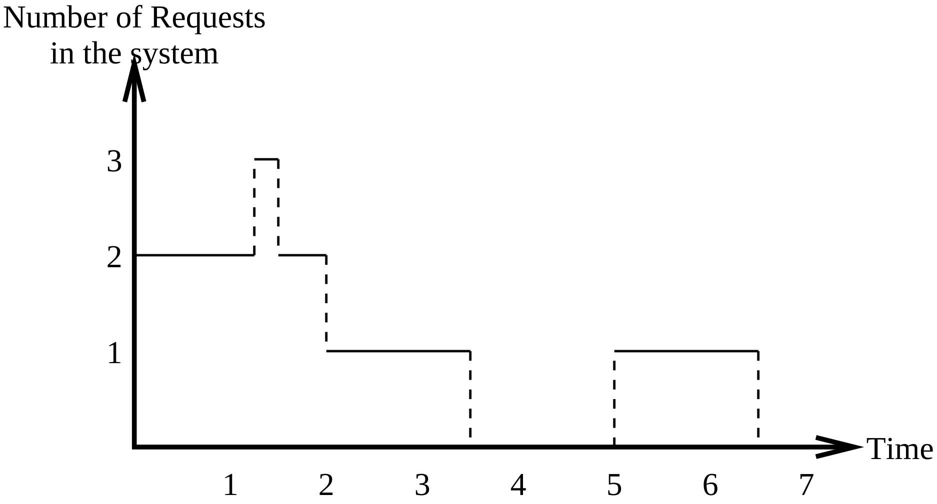


Figure B.1: Round-Robin Scheduling

requests waiting in the queue, T_2 is scheduled next.

1.25 : T_3 arrives to the system and finds the server busy. It joins the queue waiting for service.

1.5 : T_2 completes before the quantum gets over. Due to the assumption mentioned above, the new quantum begins here and T_3 is now scheduled for service.

2 : T_3 completes before the quantum gets over. The new quantum begins here and T_1 is scheduled.

3 : T_1 does not complete at the end of the quantum, and since there are no other requests in the queue, T_1 is re-scheduled.

3.5 : T_1 completes in the middle of the quantum. There are no more requests in the system and hence, the server becomes idle.

5 : T_4 arrives to find the server idle and is scheduled immediately.

6 : T_4 does not complete at the end of the quantum, and since there are no other requests in the queue, T_4 is re-scheduled.

6.5 : T_4 completes service before the end of the quantum.

Note that we have assumed that requests complete after half the quantum is over. However, this can be generalized to any value between 0 and 1. The busy period graph for the above sample path is shown in Figure B.2.

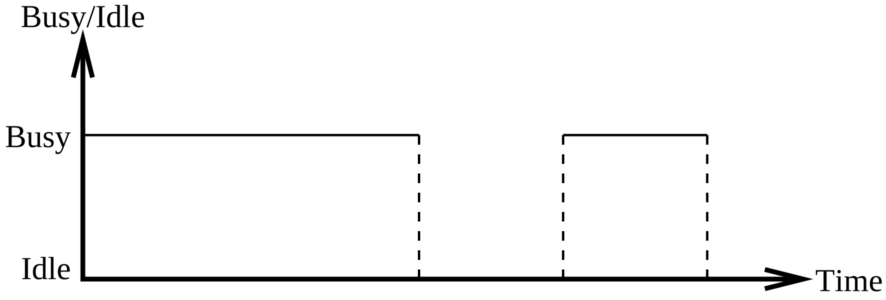


Figure B.2: Busy Periods

Now, let us try to interpret Figure B.2 using a $M/M/1$ queue with FCFS scheduler. If FCFS was used to schedule requests, the requests would be served in the order T_1, T_2, T_3 and T_4. The corresponding plot of *number of requests in the system* vs. *time* is shown in Figure B.3.

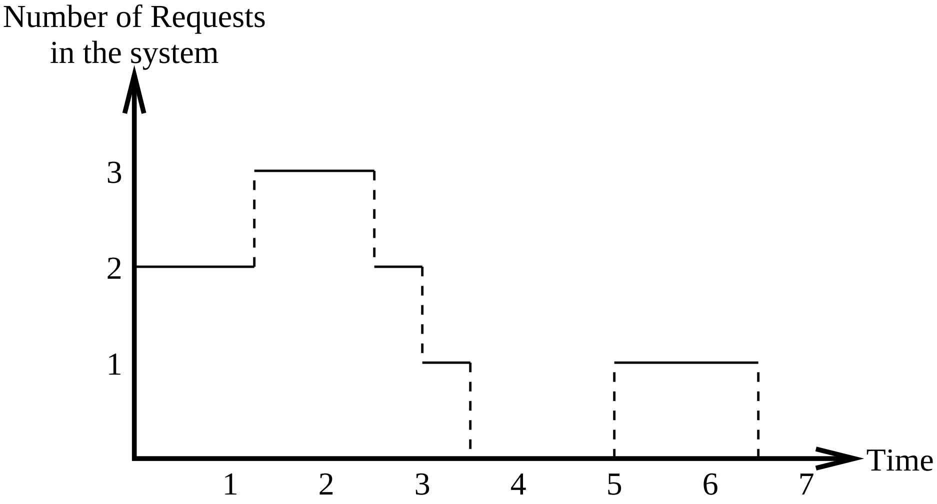


Figure B.3: FCFS Scheduling

However, the busy period graph will be the same as Figure B.2. This is expected as busy periods correspond to the unbroken periods when the server is busy. It should not be affected by the order of scheduling, but only by the moments of arrivals and the processing times. Now, since the busy period distribution is the same for both the queues, namely $M/M/1$ with FCFS and $M/M/1$ with $RR(0)$ schedulers, the output process of both the queues will also be the same. Since we know that the output process for $M/M/1$ queue with FCFS scheduler is Poisson, it will also be Poisson for the $RR(0)$ case.

Now we consider the generalized case of Round-Robin when the context switch time is *not* zero. The queue under consideration is a $M/M/1$ queue with $RR(c)$ scheduler, where $c > 0$. Let s be the service time for a request and q be the quantum size. Due to the context switching, the *effective service time*, s' will be

$$\begin{aligned} s' &= s + [s/q]c \\ &= s + \frac{s+x}{q}c \end{aligned}$$

where $[s/q]$ is the smallest integer greater than or equal to s/q
and $x = q - s\%q$ (where % is the modulus operator).
Therefore,

$$\begin{aligned} s' &= s(1 + \frac{c}{q}) + \frac{c}{q}(q - s\%q) \\ &= ks + c - \frac{c}{q}s\%q \end{aligned}$$

where $k = (1 + \frac{c}{q})$.

Since s is exponentially distributed, $ks + c$ is also exponentially distributed. As c is small compared to q, the third factor is negligible. Hence, $M/M/1$ queue with $RR(c)$ scheduler and mean service time λ_0 is nearly equivalent to a $M/M/1$ queue with $RR(0)$ scheduler with mean service time, $k\lambda_0 + c$. Hence, the output process of $M/M/1$ queue with $RR(c)$ scheduler is close to a Poisson process. It becomes exactly Poisson as the context switch time becomes zero.