

S-LOOK: A Preemptive Disk Scheduling Algorithm for Offline and Online Environments

Ammar Muqaddas*, Hanady Abdulsalam**, and Ayed Salman*

*Computer Engineering Department, Kuwait University, Kuwait

**Information Science Department, Kuwait University, Kuwait

E-mail: {ammar.mu, hanady.abdulsalam, ayed.salman}@ku.edu.kw

Abstract – We consider the problem of disk scheduling to enhance disk performance. We introduce a new algorithm (S-LOOK) for offline optimal performance, in terms of total seek time, and test it in online situations using a preemptive approach. In addition, we build a simulator for disk requests generation (DiskSims) to experimentally test our proposed algorithm and provide a comprehensive comparison with popular disk scheduling algorithms. Experimental results demonstrate that our algorithms achieve promising behavior.

Key words – scheduling algorithms, scheduling simulator.

I. Introduction

Computer systems performance is highly affected by the scheduling algorithm they use to perform their I/O operations. A good scheduling algorithm should be fast and fair. In other words, algorithms should be able to serve as many I/O requests as possible in a give period of time, while not causing starvation for some requests. The main two concerns of disk scheduling algorithms are, therefore, to reduce the total time needed to serve a number of requests (access time) and the average time between the arrival of requests and them being served (waiting time). The access time for any I/O consists of the time needed for the disk arm to move the head to the requested track (seek time) in addition to the time required for the disk to rotate to the specified sector on track (rotational latency). The dominant factor of the access time is the seek time, hence, rotational latency is usually ignored by scheduling algorithms.

One more important concern of scheduling algorithms is the ability to handle online situations, where additional requests arrive while the head is seeking to serve current requests. In such situations, a preemptive approach can be used to take a quick decision of whether the head should change its movement direction or not. This decision should be taken in favor of reducing the total seek time while not affecting the algorithm fairness by increasing the average waiting time. Clearly, this is not a concern of offline situations where it is assumed that no requests arrive while serving other requests.

Common conventional disk scheduling algorithms include FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK, and V(R)[1,2]. Improving these scheduling algorithm attracts a great deal of attention for researchers. According to the literature, the best competing of the conventional scheduling algorithms are the SSTF and the LOOK algorithms [2].

The main contribution of this paper is to introduce a new disk scheduling algorithm S-LOOK (Shortest LOOK). S-LOOK algorithm extends the conventional LOOK algorithm. It decides which way the disk head

should travel first. The algorithm is able to handle both offline and online situations. S-LOOK is mainly designed to improve on desktop PCs performance. It, however, does not fail to work with servers.

Theoretical verification and practical experiments show that the S-LOOK algorithm is optimal in offline cases and performs better than the common conventional algorithms in terms of total seek time in many online cases while still improving the average waiting time for jobs.

As an additional contribution, we implement a disk schedule simulator (DiskSims) [3], by which we evaluate our proposed algorithm. DiskSims is designed to generate random disk requests and then calculates the total seek time and the average waiting time for our algorithm as well as other common scheduling algorithms. We then provide a comparison between the results obtained from all mentioned algorithms, and show that S-LOOK achieves better performance than conventional algorithms in many cases.

The remainder of the paper is organized as follows: Section II states some background. Section III introduces the S-LOOK algorithm. Section IV explains the experimental settings. Section V discusses experimental results. Section VI presents related work. The final section draws our conclusions.

II. Background

A. SCAN algorithm

The algorithm states that the head moves from its current position towards one direction and serves all requests in its way until it reaches the last cylinder in this direction, then it switches the direction to starts serving the requests along all cylinders till it reaches the other end, and so on.

B. LOOK algorithm

The LOOK algorithm is an extension of the SCAN algorithm except that the head does not scan the disk from end to end. It, instead, scans the disk in one direction till it reaches the far-end scheduled request in this direction, and then it switches to the other direction and seeks while serving requests till it reached the far-end request in the other direction, and so on.

III.S-LOOK Algorithm

The Shortest LOOK (S-LOOK) algorithm is an extension of the LOOK algorithm to handle the cases where the disk head is located between the far-end requests. The algorithm is designed to make a decision of

which direction should be served first instead of only continuing to seek in the same direction before the new requests have arrived. Since the seek time is directly proportional to the seek distance, our goal is to minimize the seek distance, and hence, reduce the seek time.

A. Offline scheduling

For simplicity, we first consider the offline case at which we assume no new requests arrive in the duration between deciding which direction the head should move and finish serving all the requests in queue. The offline situation, hence, represents the case that requests arrive together as a group.

Consider the following example in Fig. 1 where the horizontal line represents the disk cylinders in incrementing order from 0 to 1000 (left to right), the arrow shows the current disk head position, and A and B are the distances from the head to the far-end requests at the left and right respectively.

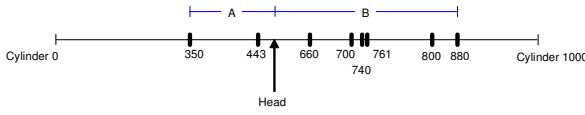


Fig. 1 Head is between the two far-end requests

From the figure, the head has two options to serve requests:

- Option 1: to start serving the requests at the right first till the far-end request at cylinder 880 is reached, and then switch to travel left till cylinder 350, or
- Option 2: to go left first till cylinder 350 and then switch to travel right till cylinder 880.

Option 1 results in a total seek distance of $2B+A$ whereas the total seek distance for option 2 is $2A+B$. Given that $A < B$, it is clear that option 2 has shorter total seek distance. The minimum seek distance to cover the total distance is, therefore, reached when the disk head seeks towards its closer far-end request first, then switches to serve the other direction.

We now consider the case where all the requests are at the right or at the left of the head. See Figures 2 and 3. Obviously, both cases have only one option, which is seeking in the requests direction. This results in a seeking distance equal to the value of the larger distance. From this observation, the minimum seek distance, $seekdis_{min}$, for serving requests using S-LOOK algorithm is

$$seekdis_{min} = \begin{cases} A+B+\min(A,B) & \text{If head is between far-end requests} \\ \max(A,B) & \text{Otherwise} \end{cases}$$

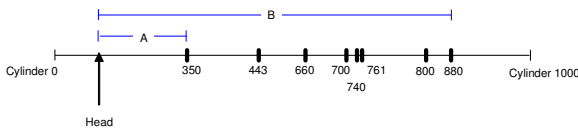


Fig. 2 Requests are at the right of the head

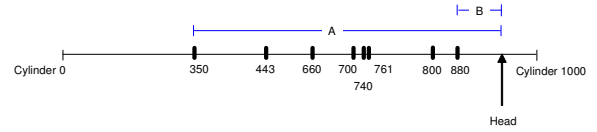


Fig. 3 Requests are at the left of the head

B. Online scheduling

We now consider the online case, where requests can arrive at any time while the head is seeking to serve other requests. The challenge in this case is that the algorithm should always keep track of new requests, which might result in changing the distances to the far-end requests. In addition, the algorithm should balance between the direction switch and the total seek time since too many head switches may cause large overhead. Consider again the case in Figure 1. Following option 2, suppose that the head moves towards the request on cylinder 443 and serves it, then, according to our algorithm, it continues to move towards cylinder 350. Mean while, assume two new requests at cylinders 550 and 790 arrive. See Figure 4. Note that the new requests do not affect the distances A and B since they appeared to be within the margins of the two far-end requests. Hence, the head does not change its direction and continues to serve further requests in the same direction.

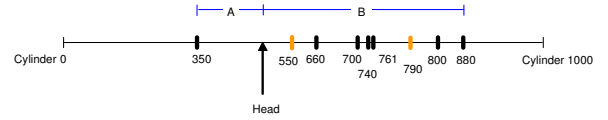


Fig. 4 Requests with no effect the current plan

Now assume that a new request arrives at cylinder 12, Figure 5. The distance A has, obviously, got affected. Depending on the new value of A whether it's still less than B or not, the scheduler decides to flip the head direction or remain in the same direction. However, depending only on the distances should not always be the case. This is because too many changes in direction might cause traveling longer distances.

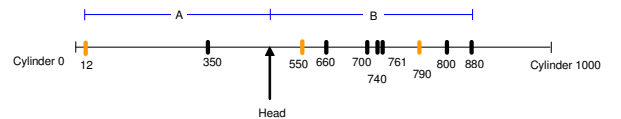


Fig. 5 Requests that affects the current plan

In order for the algorithm to consider switching the disk head or not, we introduce a formula to calculate the gain of switching the head direction. Assume that $time_{wasted}$ represents the time duration since the head has started moving. Upon the arrival of any new requests, the distances from the head position to the far-end requests are recalculated. If the distances relationship changes, then the head switches direction only if the inequality $\min(A,B)+time_{wasted} < \max(A,B)$ holds. The procedure followed by our algorithm is explained in Algorithm 1.

Input:

- cyl_{dr}_{max} : maximum cylinder number of requests in current queue
- cyl_{dr}_{min} : minimum cylinder number of requests in current queue
- cyl_{dr}_n : cylinder number of request n
- $head_{pos}$: cylinder number of current head position
- $WTF(wasted\ time\ factor)$: constant specified by the operating system
- $head_{dir}$: represents the head direction

```

= 1 if head is moving towards higher cylinder numbers
= -1 if head is moving towards lower cylinder numbers
= 0 if the head is not moving
A:  $head_{pos} - cyl_{dr_{min}}$ 
B:  $head_{pos} - cyl_{dr_{max}}$ 
time_wasted : time since the head started moving

main {
Update the queue for new requests
if there exists new requests with  $cyl_{dr_n}$  NOT between  $cyl_{dr_{min}}$  and  $cyl_{dr_{max}}$  then
    Update  $cyl_{dr_{min}}$  and/or  $cyl_{dr_{max}}$ 
    Update A and/or B
    UpdateHeadDirection()
end
if there exists a request "n" with  $cyl_{dr_n} = head_{pos}$  then
    service request n
else
    if  $head_{dir} \neq 0$  then
        Move the head one cylinder in the direction indicated by  $head_{dir}$ 
        Update  $head_{pos}$ 
         $time_{wasted} = time_{wasted} + WTF$ 
    end
end
}
Function UpdateHeadDirection() {
if there exists requests pending in the queue then
    if  $\min(A,B) + time_{wasted} < \max(A,B)$  and head is NOT moving
    towards the far-end request of  $\min(A,B)$  then
        if  $head_{dir} = 0$  then
            Update  $head_{dir}$  to the direction of the far-end request of
             $\min(A,B)$ 
        else
             $head_{dir} = - head_{dir}$ 
        end
    end
else
    Reset  $head_{dir}$  to 0
end
}

```

Algorithm 1: S-LOOK algorithm for online and offline scheduling

IV. Implementation And Experimental Settings

A. Testing criteria

Since S-LOOK is designed to minimize the total seek time, the main criteria by which we evaluate it is the total seek time. We, also, measure the average waiting time to show that S-LOOK does also improve the average waiting time. We validate the performance of S-LOOK its performance with SSTF, SCAN, C-SCAN, LOOK, and C-LOOK. FCFS is ignored in our comparisons because of its bad performance.

B. DiskSims Simulator

We conduct our experiments on a system with Windows XP, 2.0 GHz processor and 1.5 GB RAM. We implement a simulator using C++ language. Our simulation test bed is a virtual 80Gb hard disk. The disk's physical geometry consists of exactly 34327 cylinders, with an average of 600 blocks per track, a sector size of 512 bytes and 8 heads. The disk is assumed to be formatted with a blocksize of 4kb. Those parameters are inspired from the IBM-DTLA-305XXX series.

C. Experimental settings

We define three levels of desktop requests' loads; low, medium, and high based on observation experiments using DiskMon [4], a disk monitor software tool, which monitors disk access and produces a log of all requests done during a period of time. We monitor the desktop for

three distinct periods (low, medium, and high), each lasting for about 5 minutes. For the low and medium load periods, we run a number of applications that can be considered as the average case scenarios, while for the high load period, we test the maximum number of applications that could be run. During the low usage period, one application, a word document, and 3 web sites were opened. During the medium usage period, one online video was streaming, a word document was edited, one application was launched, and many sites were browsed while downloading a file of size 1.67MB. During the high usage period, around 20 different applications were running simultaneously, many documents were edited and saved, a web server was running, and many internet sites were browsed while downloading two files from the internet and copying two large folders of 900Mb from the network. Our results are as follows: around 10 requests/second for low usage, 45 requests/second for medium usage, and 150 r/s for high usage. We, therefore, define our intervals as 1 to 20 r/s for low load period, 21 to 70 r/s for medium load period, and 71 to 150 requests for high load period. We, accordingly, conduct different batches of simulation runs to cover all the defined arrival rates, each with 20 runs. Each run generates 1000 random disk requests, that is different than any other run. Our recorded results are the averages of the 20 runs. For all our experiments WTF is set to 1 and timewasted is reset to zero every time the queue is empty. We, in addition, conduct more experiments for loads of more than 150 r/s to show that S-LOOK does not fail to work for servers and also to use comparable loads of requests with the research work in literature [5]. We, therefore, run S-LOOK with loads of 260 r/s, 360 r/s, and 610 r/s.

V. Results and Discussion

The results we got for running our simulator for the different desktop and server loads are shown in Table 1 and Table 2 respectively, with times in milliseconds. Note that the highlighted values represent the minimum values among each column. As it can be seen from the tables, the desktop results are shown for four different arrival rates, three of which represent the averages of the low, medium, and high ranges for desktop usage. The forth value (150 requests/second) represent the highest arrival for desktops. The desktop results demonstrate that our algorithm S-LOOK records both minimum total seek time and average waiting time. As it can be seen from the tables, S-LOOK performs better than LOOK (the second wining algorithm always) in terms of total seek time for the three desktop ranges of loads, while also, recording better average waiting time results. For the low load results, the total seek time for S-LOOK is less than LOOK by around 20 milliseconds, while for medium and high loads, the total seek time of S-LOOK is less than LOOK by around 65 milliseconds. Even though the average waiting time improvement does not seem significant, it still affects the overall performance. For example, for the low load desktop results in Table 1, the average waiting time of S-LOOK is less than LOOK by

only around 0.2 millisecond, which does not seem much. However, for a queue of 1000 request this reduction in average waiting time results in 2.4% improvement. The recorded server arrival rates, as shown in the Table 2, are 260 r/s, 360 r/s, and 610 r/s. Clearly, S-LOOK records comparable results to LOOK.

TABLE 1

TOTAL SEEK TIMES AND AVERAGE WAITING TIMES FOR DESKTOP LOADS

Algorithm	10 r/s	45 r/s	110 r/s	150 r/s
<i>Total Seek Time (milliseconds)</i>				
SLOOK	6927.216	6646.363	5845.06	5222.619
LOOK	6946.801	6712.89	5909.165	5259.176
SSTF	6984.046	6826.461	6079.481	5366.449
CLOOK	6984.922	6866.033	6148.16	5452.207
SCAN	12187.605	10134.904	7161.565	5833.203
C-SCAN	17508.906	12895.608	7955.461	6180.447
<i>Average Waiting Time (milliseconds)</i>				
SLOOK	7.207	8.067	10.249	11.801
LOOK	7.253	8.268	10.583	12.125
SSTF	7.35	8.7	11.77	13.612
CLOOK	7.358	8.895	12.748	15.472
SCAN	13.03	13.671	14.868	15.56
C-SCAN	19.578	20.596	21.99	23.055

TABLE 2

TOTAL SEEK TIMES AND AVERAGE WAITING TIMES FOR SERVER LOADS

Algorithm	260 r/s	360 r/s	610 r/s
<i>Total Seek Time (milliseconds)</i>			
SLOOK	3608.762	2648.141	1530.377
LOOK	3613.51	2646.479	1531.161
SSTF	3638.819	2654.967	1533.43
CLOOK	3661.448	2662.724	1550.859
SCAN	3666.28	2654.373	1533.422
C-SCAN	3708.366	2669.696	1551.684
<i>Average Waiting Time (milliseconds)</i>			
SLOOK	16.159	20.081	40.436
LOOK	16.22	20.093	40.486
SSTF	18.2	22.68	45.412
CLOOK	22.272	28.783	58.302
SCAN	18.222	21.768	41.695
C-SCAN	26.83	32.105	60.334

VI. Related Work

As mentioned in the introduction section, the field of scheduling algorithms is rich with many research. Recent algorithms [6–10] usually extend (or based on) one of the conventional algorithms. The GSR algorithm (global seek-optimizing real time disk scheduling algorithm) [6] is based on SCAN and EDF algorithms, in addition to the FCFS algorithm. Other extends to EDF algorithm to deal with multimedia servers include the MS-EDF algorithm [7], as well as the HB-EDF algorithm [8] that work for mixed workload environment. Povzner et al. [9] propose a scheduling algorithm that uses the SSTF and EDF in forming its architecture. Zarandioon and Thomasian [10] propose a methodology of reducing the computational cost of the SPTF algorithm by developing the implementation of the algorithm. They also propose an efficient implementation of LOOK-ahead algorithm.

Conclusion

In this work, we proposed a disk scheduling algorithm, S-LOOK, that reduces the total seek time, and implement a simulator (DiskSims) to test it. It extends the conventional LOOK algorithm in deciding which way to move the head first in the case the disk head is located in between the two far-end requests. Our algorithm is optimal in terms of total seek time for offline situations. It can also handle online situations with performance better than many conventional algorithms. S-LOOK is mainly designed for desktop usage. It can also be used for servers at which it converges to LOOK. Experimental results demonstrate that S-LOOK is able to perform better than other conventional algorithms in terms of total seek time, while still improving the average waiting time for jobs in queue. Results also show that for high arrival rates of requests, S-LOOK converges to LOOK, which is one of the best algorithms that work under heavy loads.

References

- [1] R. Geist and S. Daniel, "A continuum of disk scheduling algorithms," *ACM Transactions on Computer Systems*, vol. 5, pp. 77–92, 1987.
- [2] M. Javed and I. Khan, "Simulation and performance comparison of four disk scheduling algorithms," in *TENCON 2000. Proceedings*, vol. 2, 2000, pp. 10–15 vol.2.
- [3] DiskSims Simulation Tool. [Online]. Available: <http://cpe.kuniv.edu/users/ammarm/disksims.htm>
- [4] Disk Monitor Software Tool (DiskMon). [Online]. Available: <http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx>
- [5] A. Riska, E. Riedel, and S. Iren, "Adaptive disk scheduling for overload management," in *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, Sept. 2004, pp. 176–185.
- [6] H.-P. Chang, R.-I. Chang, W.-K. Shih, and R.-C. Chang, "Gsr: A global seek-optimizing real-time disk-scheduling algorithm," *Journal of System Software*, vol. 80, no. 2, pp. 198–215, 2007.
- [7] Y.-F. Huang and J.-M. Huang, "Disk scheduling on multimedia storage servers," *IEEE Transactions on Computers*, vol. 53, no. 1, pp. 77–82, 2004.
- [8] T. Kim, J. Park, S. Ahn, K. Koh, Y. Won, and H. Bahn, "A heuristic-based real-time disk scheduling algorithm for mixed-media workload," in *IMSA'06: Proceedings of the 24th IASTED international conference on Internet and multimedia systems and applications*. Anaheim, CA, USA: ACTA Press, 2006, pp. 165–170.
- [9] A. Povzner, T. Kaldewey, S. Brandt, R. Golding, T. M. Wong, and C. Maltzahn, "Efficient guaranteed disk request scheduling with fahrrad," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 4, pp. 13–25, 2008.
- [10] S. Zarandioon and A. Thomasian, "Optimization of online disk scheduling algorithms," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 4, pp. 42–46, 2006.