A New Optimized Real-Time Disk Scheduling Algorithm

Nidhi
Madan Mohan Malaviya University of Technology,
Gorakhpur (UP)-273010, India

Dayashankar Singh
Madan Mohan Malaviya University of Technology,
Gorakhpur (UP), India

ABSTRACT
In this paper, a new approach of disk scheduling has been proposed to improve the throughput of modern storage device. After the invention of disk with movable head, researchers are making efforts continuously to improve the I/O performance by implementing many intelligent scheduling algorithms of disk accesses. Speeds of processor and memory capacity have been increased several times than the speed of the disk. Due to this difference in the speed of processor and the disk, I/O performance of disk has become an important bottleneck. Therefore, it is needed to develop some advanced methods for using disk more efficiently. Management of disk performance is an important aspect of operating system research and development. In this paper, a new disk scheduling algorithm has been developed and implemented to reduce the number of head movements. Experiment has been carried out to compare the performance of proposed algorithm with the performance of all six classical disk scheduling algorithms. Experiment shows that the numbers of head movements in existing classical disk scheduling algorithms such as FCFS, SSTF, SCAN, C-SCAN, LOOK and C-LOOK are high. But in proposed new optimized real-time disk scheduling algorithm, head movements are reduced and hence it maximizes the throughput for the modern storage devices.

Keywords
FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK, Scheduling Algorithm, Scheduler, Disk Head

1. INTRODUCTION
Since almost all the computer resources are needed to be scheduled before use, scheduling is one of the important fundamental function of operating system. Disk is one of the important resources of computer. For meeting the responsibilities for the disk driver entails having large disk bandwidth and fast access time. Speed of the processor and the memory capacity are increasing at very fast rate over 40% per year but the speed of disk is growing only 7% per year [6]. Because this rate is unlikely to change substantially in the incoming days, the I/O performance comes as the system bottleneck. Even it is difficult to improve the mechanical components, therefore it has been tried to use the disk more efficiently. The access time is the important factor. Generally the disk operates with a small function of the maximum bandwidth the disk have. It has been proved experimentally that more sophisticated disk head scheduling algorithms give higher throughput [9]. In the past work the researchers focused mainly on two types of work. One is the synthetic workloads in that disk requests are uniformly and randomly distributed across the whole disk. The other is traces which is more recent and in that all the requests to an actual are firstly recorded and then used as a testing ground for algorithms [2]. Many researchers made little attempts for developing algorithms which provides performance guarantees. The access time has two components: Seek Time and Rotational Latency. The Seek Time is the time for the disk arm to move the head to the tracks which are containing the desired sector. There is also an additional time called Rotational Latency which is the time for the disk to rotate the desired sector under the disk head. Disk bandwidth is the total no of bytes transferred, divided by the total time between first request for the service and completion of the last transfer [1].

In this paper efforts have been made to improve the access time as well as bandwidth by scheduling carefully all disk I/O requests in an optimized and good order. The request can be serviced immediately if the disk driver and controller are available and free. If the driver or controller is not free, new requests for the services will be placed in a queue of the pending requests. For a system with many processors called multiprogramming system, the disk queue may have many pending requests. After completing one request the operating system needs to select the next pending requests to service.

Experiment has been carried out by considering the same request queue for implementing the existing all six classical disk scheduling algorithms such as and newly developed disk scheduling algorithm. It has been analyzed that newly developed algorithm requires less number of head movements in order to service the disk I/O requests available in request queue.

2. BACKGROUND AND RELATED WORK
Many disk scheduling algorithms are there to schedule for servicing the disk I/O requests. We illustrate all these algorithms with a request queue (0-200): 38, 180, 130, 10, 50, 15, 190, 90, and 150. Head starts at 120.

2.1 First Come First Served Algorithm
The First Come First Served (FCFS) algorithm is simplest form of disk scheduling algorithm. This algorithm essentially fair, but generally does not give the fastest service compared to other disk scheduling algorithms.
Example, consider a disk queue with requests for I/O to blocks on tracks: Queue (0-200): 38, 180, 130, 10, 50, 15, 190, 90, and 150. Head starts at 120.
Initially the head is at track 120, it will first move from 120 to 38, then to 180, 130, 10, 50, 15, 190, 90, and finally to 150. The scheduling is represented in Figure: 1.1.

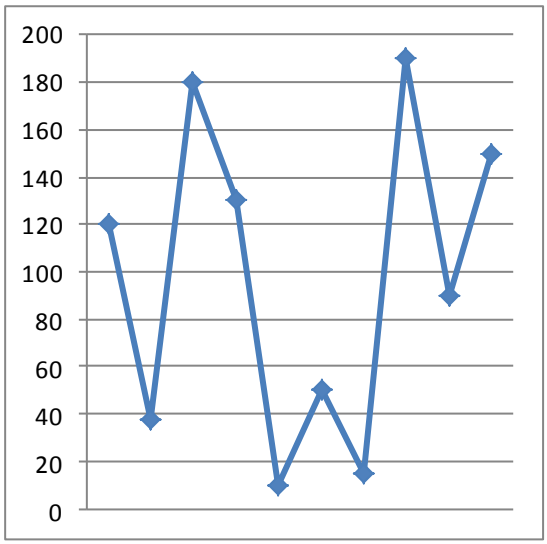


Fig-1.1: FCFS Disk Scheduling Algorithm

Total Head movements = (120-38) + (180-38) + (180-130) + (130-10) + (50-10) + (50-15) + (190-15) + (190-90) + (150-90) = **804**

The above calculation shows that if we use FCFS disk scheduling algorithm then the total head movements require to service the disk I/O requests are **804.**

2.2 Shortest-Seek-Time-First Algorithm

In the Shortest-Seek-Time-First(SSTF) disk scheduling algorithm head serves the request first that have minimum seek time from the current head position. As the seek time increases with the numbers of tracks traversed by the head, the SSTF algorithm first chooses the pending requests close to the current head position. SSTF scheduling is basically a form of shortest job first (SJF) scheduling and like SJF scheduling it is not very fair and may cause starvation of some requests.
Consider the same example: Queue: 38, 180, 130, 10, 50, 15, 190, 90, and 150. Initially the head is at 120.
Head is initially at position 120. The closest request to the initial head position is at track 130. Once we are at 130, the next closest request is at position 150 and so on from 150 to 180, 190, 90, 50, 38, 15 and then 10. The scheduling is represented in Figure 1.2.

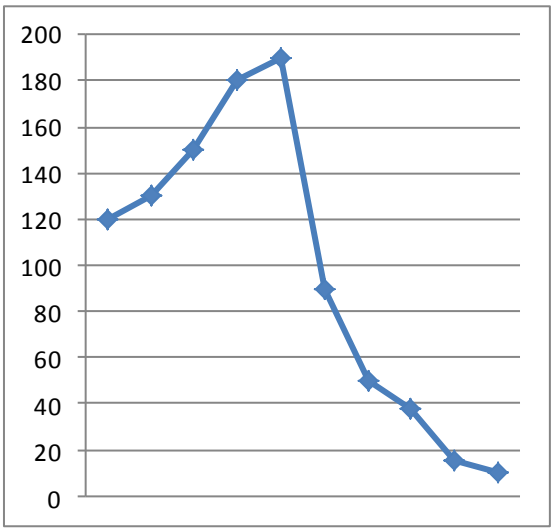


Fig-1.2: SSTF Disk Scheduling Algorithm

Total head Movements = (130-120) + (150-130) + (180-150) + (190-180) + (190-90) + (90-50) + (50-38) + (38-15) + (15-10) = **250**
The above calculation shows that if we use SSTF disk scheduling algorithm then the total head movements require to service the disk I/O requests are **250.**

2.3 SCAN Algorithm

In this algorithm the head starts at one end of the disk and moves towards the other end of the disk, servicing requests as it reaches each track, until it reach to the other end of the disk. After reaching the other end of the disk, the direction of head is reversed and servicing continues. It is some time called the Elevator algorithm because in this algorithm the disk arm behaves like an elevator.
Let as return to the same example. In this algorithm we need to know the direction of head in addition to the current position of the head. Let the disk arm is moving towards 0 and initial head position is 120. Queue: 38, 180, 130, 10, 50, 15, 190, 90 and 150.

Initially the head is at 120, the head will first serve 90 and then 50, 38, 15, 10. At track 0 direction of the arm will reverse and start moving towards the other end of the disk and services the requests at 130, 150, 180 and then 190. The scheduling is represented in Figure: 1.3.

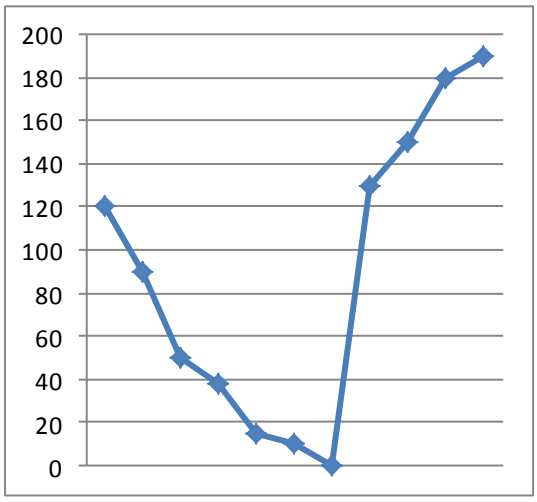


Fig-1.3: SCAN Disk Scheduling Algorithm

Total Head Movements = (120-90) + (90-50) + (50-38) + (38-15) + (15-10) + (10-0) + (130-0) + (150-130) + (180-150) + (190-180) = **310**

The above calculation shows that if we use SCAN disk scheduling algorithm then the total head movements require to service the disk I/O requests are **310**.

2.4 C-SCAN Algorithm

This algorithm provides more uniform waiting time than the SCAN algorithm. Like the SCAN algorithm, the disk head moves from one end of the disk to the other end of the disk in this algorithm, servicing requests along the way. When the head reaches the other end of the disk, it immediately returns back to the beginning of the disk without servicing any requests in the return. This algorithm treats the tracks as a circular list that warps around from the last track to the first track.

Consider the same example. Queue: 38, 180, 130, 10, 50, 15, 190, 90, 150. The current head position is at 120 and direction of disk head is towards the position 0. The scheduling is shown in Figure 1.4.

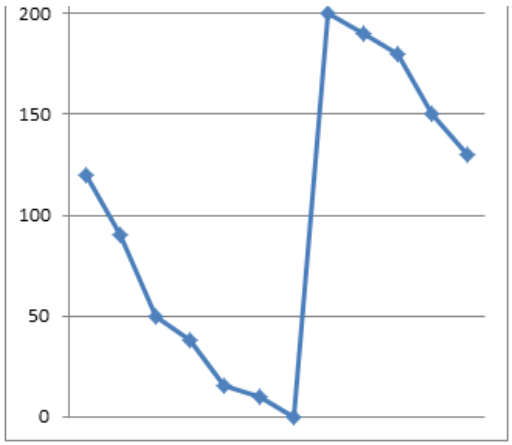


Fig-1.4: C-SCAN Disk Scheduling Algorithm

Total Head Movements = (120-90) + (90-50) + (50-38) + (38-15) + (15-10) + (10-0) + (200-0) + (200-190) + (190-180) + (180-150) + (150-130) = **390**

The above calculation shows that if we use C-SCAN disk scheduling algorithm then the total head movements require to service the disk I/O requests are **390**.

2.5 LOOK ALGORITHM

Similar to the SCAN, in LOOK algorithm the disk head sweeps across the surface of disk in both the direction performing reads and writes. However, unlike the SCAN algorithm in which the disk head visits the innermost and outermost tracks in each sweep, LOOK algorithm will reversed the direction of disk head when it reached to the last request in the current direction of the head.

Consider an example. Queue: 38, 180, 130, 10, 50, 15, 190, 90, 150. The initial head position is at 120 and the current direction of the disk head is towards 0. The scheduling is shown in Figure: 1.5.

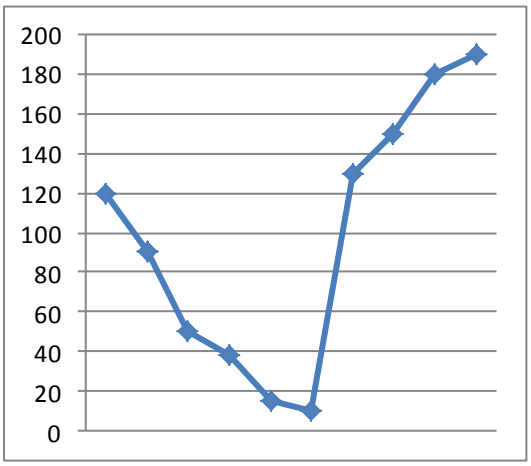


Fig-1.5: LOOK Disk Scheduling Algorithm

Total Head Movements = (120-90) + (90-50) + (50-38) + (38-15) + (15-10) + (130-10) + (150-130) + (180-150) + (190-180) = **290**

The above calculation shows that if we use LOOK disk scheduling algorithm then the total head movements require to service the disk I/O requests are **290.**

2.6 C-LOOK ALGORITHM

C-LOOK is a version of C-SCAN algorithm in which the arm goes only as far as last request in current direction and immediately reverses the direction of the disk head, without first going all the way to the end of the disk.

Consider the same example. Queue: 38, 180, 130, 10, 50, 15, 190, 90, 150. The head is initially at position 120 and current direction of head is towards 0.

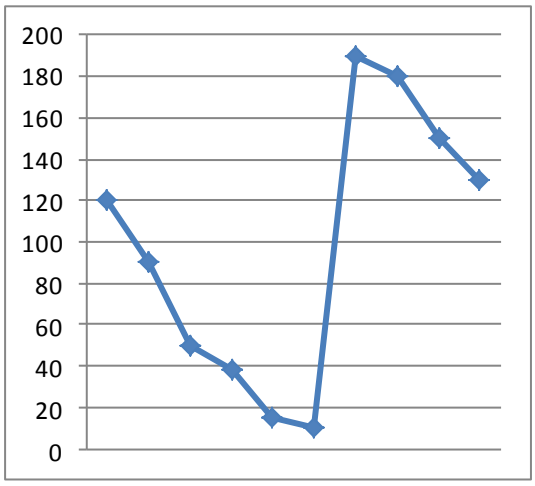


Fig-1.6: C-LOOK Disk Scheduling Algorithm

Total Head Movements = (120-90) + (90-50) + (50-38) + (38-15) + (15-10) + (190-10) + (190-180) + (180-150) + (150-130) = **350**

The above calculation shows that if we use C-LOOK disk scheduling algorithm then the total head movements require to service the disk I/O requests are **350.**

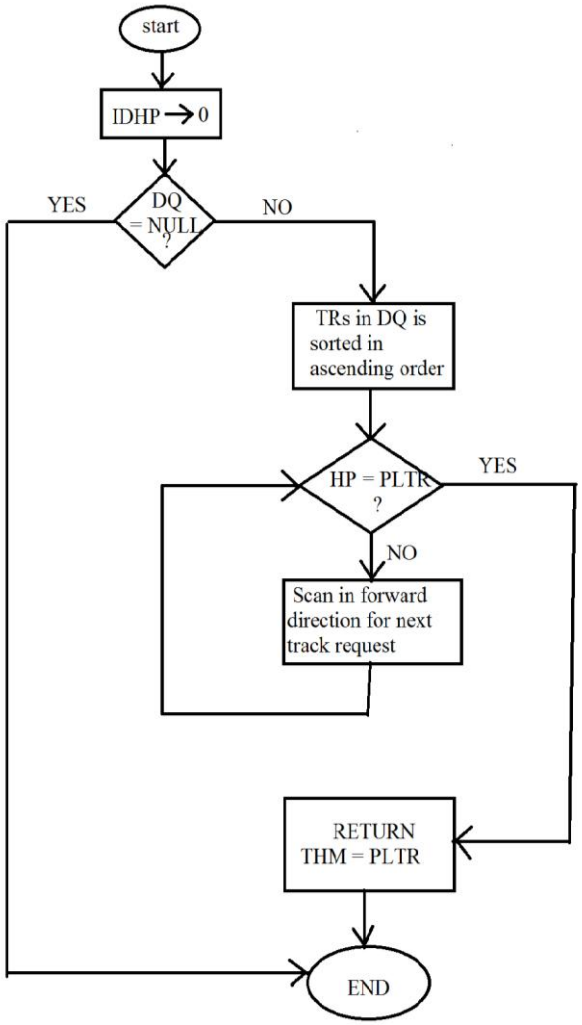
3. PROPOSED NEW OPTIMIZED REAL -TIME DISK SCHEDULING ALGORITHM

In this proposed algorithm, initially the disk head is at the position 0 and has the direction towards the position 200. It means initial head position and direction of head is always same. First we sort all the cylinders input blocks by using any sorting algorithm. Initially the head is at position 0 and sequentially moves and reached from this block to the highest input block number, servicing all the input request blocks in front of the head immediately.

Procedure/Algorithm

- Assume a[] is an array containing track numbers and x is the position of last input block.
- Inialize Head position is at 0. Assume h denotes the current head position.
- Sort input blocks of cylinder number in ascending order with the help of any sorting algorithm.
- Initially head position h is 0.
- for(i=0; i<=x; i++)
- Service the input request in front of head immediately.
- Total_head_movements = x;
- Return total_head_movements;

Flowchart



Graphical Representation of Proposed Algorithm

Assumptions in flowchart are given below:

IDHP = Initial Disk Head Position

DQ = Disk Queue with requests

TR = Track Request

HP = Head Position

PLTR = Position of Last Track Request

THM = Total Head Movements

Consider the same example. Queue: 38, 180, 130, 10, 50, 15, 190, 90, 150. According to the rule of proposed algorithm, current position of the disk head is at 0 and direction of the disk head is towards 200. If a request arrives in the queue just in front of the disk head, it will be serviced immediately. Initially the disk head is at position 0 and moves to 10 and then to 15, 38, 50, 90, 130, 150, 180, and then to 190. This scheduling is represented in Figure: 1.7.

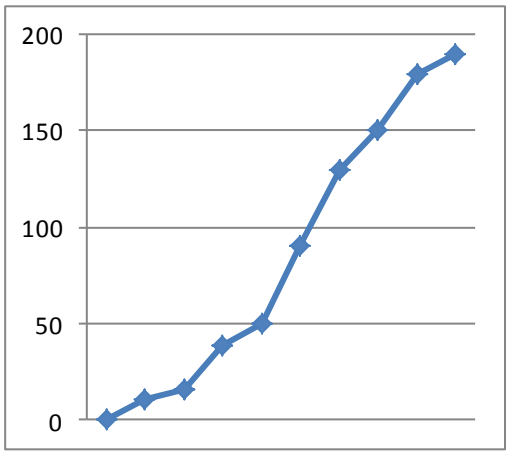


Fig- 1.7: New Optimized Real-Time Disk Scheduling Algorithm

Total Head Movements = (10-0) + (15-10) + (38-15) + (50-38) + (90-50) + (130-90) + (150-130) + (180-150) + (190-180) = **190**

Hence in new optimized real-time disk scheduling algorithm (proposed algorithm), total head movements required to service the disk I/O requests are **190**.

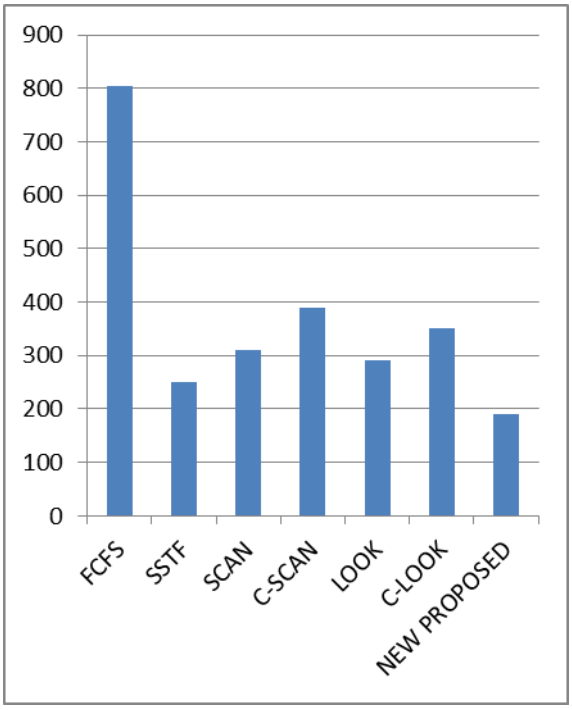
4 COMPARATIVE RESULT ANALYSIS

A comparative analysis of various classical existing disks scheduling algorithms and newly developed optimized real-time disk scheduling algorithm has been performed and result has been shown in Table 1.

Table 1: Comparative analysis of Results of Disk Scheduling Algorithms

S.N	Name Of Disk Scheduling Algorithm	Number Of Head Movements
1.	FCFS	804
2.	SSTF	250
3.	SCAN	310
4.	C-SCAN	390
5.	LOOK	290
6.	C-LOOK	350
7.	NEWLY Optimized Disk Scheduling Algorithm	190

Comparison Graph among Proposed and Existing Algorithms that shows performances:



It has been analyzed from above table that the newly developed optimized real-time disk scheduling algorithm requires minimum head movements in order to service the disk I/O requests. There are several advantages of using this proposed algorithm.

1. Total number of head movements are always less than or equal to the N, where N is the position of last track on the disk.
2. If all the input blocks are concentrated near the position 0 (means in the first half of the disk), then this proposed algorithm gives best result.
3. It is very simple algorithm. Calculation of total head movements is very simple and is always equal to the x, where x is the position of last input block.

5. CONCLUSIONS

In this paper, a new real-time optimized disk scheduling has been implemented which imposes almost no performance penalty over the non-real time optimal schedulers, when have sufficient slack time. With the help of above experiments and comparison of our proposed algorithm with existing algorithms, it is clear that the proposed algorithm reduces the total head movements. In this algorithm, sometimes the number of head movements is equal to SSTF or LOOK scheduling but it occurs very rarely. Worst case occurs when all the input blocks are concentrated near the position 200 or at the position 200. In this paper a lot of efforts have been done to improve the performance of disk I/O access, even there are tremendous scope for the improvement of disk I/O access.

6. REFERENCES

[1] Operating System Principles, Silberschatz, Peter Bare Galvin, Greg Gagne.

[2] Mordern operating system (2nd edition) Andrew S. Tanenbaum.

[3] Operating Systems: A Concept-based Approach(2E) D.M. Dhamdhere

[4] Sandipon Saha, Md. Nasim Akhter, Mohammod Abul Kashem, "A New Heuristic Disk Scheduling Algorithm" International Journal of Scientific &Technology Research Volume 2, Issue1,pp.49-53, January 2013

[5] Chang, R., Shih, W., And Chnag, R. "Real-time disk scheduling for multimedia application with deadline-modification scheme "International Journal of Time-critical Computing system 19(2000), 149-168.

[6] **C**. Reummler and J. Wilkes. An introduction to disk drive modeling. IEEE Computer, 27(3): 17-19, March 1994.

[7] Dees, B. Native Command queuing-advanced performance in desktop storage. Potentials, IEEE 24, 4(2005),4-7.

[8] Huang, Y., And Huang, J. Disk scheduling on multimedia storage servers. Computers, IEEE Transactions on 53, 1(2004), 77-82.

[9] Reddy, A.L.N., Wyllie, J., and Wijyaratne, K.B.R. disk scheduling in a multimedia I/O system. ACM Trans. Multimedia Comput. Commun. Appl. 1, 1(2005), 37-59.