OPTIMIZED TWO HEAD DISK SCHEDULING ALGORITHM (OTHDSA)

Karishma Singh
Deptt. of CSE
MMMUT
Gorakhpur (UP), India
Email:
singhkarishma.880@gmail.com

Nidhi
Deptt. of CSE
MMMUT
Gorakhpur (UP), India
Email:
nidhimmm332@gmail.com

Divya Rastogi
Deptt. of CSE
MMMUT
Gorakhpur(UP),India
Email:
divya.rastogi92@gmail.com

Dayashankar Singh
Assistant Professor
(CSE)
MMMUT
Gorakhpur (UP), India
Email:
dss_mec@yahoo.co.in

***Abstract*-In this paper, we proposed a new disk scheduling algorithm for improving the performance of modern storage devices in terms of throughput. Since the invention of disk with movable heads, many researchers tried to improve the I/O performance by using intelligent algorithms for disk scheduling. Memory capacity and speed of the processor are increasing several times than the speed of disk. This disparity shows that this cannot supply data to processor as fast as it requires. As soon as data is received by the processor from disk, data is processed immediately and then processor waits for next data from disk. Hence disk I/O performance becomes a bottleneck. Some advanced methods and scheduling algorithms are required for increasing disk I/O performance and using the disk efficiently. We can improve the performance of disk by reducing total number of head movements needed to serve requests. In this paper, we proposed and implemented an optimized two head disk scheduling algorithm (OTHDSA) which reduces total number of head movements. Experimental analysis has been carried out and it is observed that this proposed algorithm requires less number of head movements than many existing disk scheduling algorithms and hence it maximizes the throughput.**

Keywords- FCFS, SSTF, SCAN, C-SCAN LOOK, C-LOOK Disk Scheduling algorithm.

I. INTRODUCTION

One of the important fundamental functions of operating system is scheduling and almost all the computer resources are needed to be scheduled before their use for increasing the performance in terms of throughput. Modern storage devices like disk is one of the fundamental resource of computer and for meeting this responsibility the disk drivers entails having large disk bandwidth and fast access time[10]. The processor's speed and the memory capacity are increasing dramatically at fast rate over 14% per year but the disk speed is growing by only 7% per year[8] and this rate is unlikely to change dramatically in the near future, the disk I/O performance comes as performance bottleneck for the system. In terms of disk I/O performance, access time is the important factor. Generally disk operates with a small function of maximum disk bandwidth. It has been proved experimentally that if we use more intelligent and sophisticated disk scheduling algorithms then they give higher throughput [8].

In the past work related to scheduling of disks the researchers mainly focused on two types of works. One is the synthetic workload in which the disk requests are randomly and uniformly distributed and in the other all the requests are first recorded and then used for testing ground for algorithm in order to provide the performance guarantee [2]. The access time contains mainly two components, one is seek time and other is rotational latency. The seek time is the time needed for the disk head to move to the tracks containing the desired sector whereas the rotational latency is the additional time required by the head to rotate to the desired sector on that track. We can define the disk bandwidth as the ratio of total number of bytes transferred to the total time between the first request for the service and the completion of last transfer [1].

A new disk model MHPR (Multi-head Parallelism and Redundancy) is presented by Yang in which there are more heads in one arm assembly which introduce redundancy and parallelism in disk [4]. In that model, the access time for a single request is reduced and hence the response for the request becomes fast. In any disk scheduling algorithm, if the disk driver and controller are free then the request can be serviced immediately else new requests will be placed in a queue of pending requests. In a multiprogramming system, the queue may have many pending requests from which the operating system selects one request for its completion and then it needs to select another request from the queue after the completion of previous request. In this paper attempts have been made for improving the access time and the bandwidth by using a new optimized disk scheduling algorithm which uses two heads. We performed some experiments by considering a request queue and compared the result of our proposed algorithm with other existing disk scheduling algorithm. It has been analyzed that newly proposed algorithm requires less

2327-0659/15 $31.00 © 2015 IEEE
DOI 10.1109/ACCT.2015.70

number of total head movements in order to serve the I/O requests. [10, 12, 13]

II. BACKGROUND AND RELATED WORK

There exist many disk scheduling algorithms. In this chapter we briefly introduce those algorithms. We take a dataset of I/O requests as an example (as shown below) and results have been compared with the newly proposed disk scheduling algorithm to show that proposed algorithm requires the minimum seek time comparative to others and hence give better performance.

Example 1: Consider a disk queue with I/O requests to blocks on tracks: Queue (0-200): 60, 90, 40, 20, 110, 70, 150, 130, 160, 120. Head starts at 80.

Example 2: Consider a disk queue with I/O requests to blocks on tracks: Queue (0-200): 40, 60, 20, 70, 120, 160, 130, 180, 80, 190. Head starts at 80.

A. *First Come First Served (FCFS) Algorithm*

FCFS algorithm is the simplest disk scheduling algorithm. This algorithm is essentially fair but generally does not give faster service in comparison to other algorithms. [1, 2, 3]

If we solve example 1 and example 2 using FCFS algorithm then total head movements are 420 and 530 respectively.

B. *Shortest-Seek-Time-First (SSTF) Algorithm*

In the SSTF algorithm the head serves the request first that have minimum seek time from the current position of head. As the seek time increases with the number of tracks traversed by the head, the SSTF algorithm first services the pending request close to the current head position. [1, 2, 3]

If we solve example 1 and example 2 using SSTF algorithm then total head movements are 220 and 230 respectively.

C. *SCAN Algorithm*

In SCAN algorithm the head starts at one end of the disk and move towards the other end, servicing I/O request as it reaches the desired track, until it reaches to the other end of the disk. When it reaches to the other end of the disk, the direction of head is reversed and continues servicing. Sometimes it is also called Elevator Algorithm because the disk arm in this algorithm behaves like an elevator. [1, 2, 3]

If we solve example 1 and example 2 using SCAN algorithm then total head movements are 300 and 300 respectively.

D. *C-SCAN Algorithm*

C-SCAN algorithm gives more uniform waiting time than the SCAN algorithm. Like the SCAN algorithm, the head of the disk moves from one end of the disk to another end in this algorithm, servicing request along the way. When the disk head moves to another end of the disk, the head immediately returns back to the beginning of the disk without servicing any request in the return journey. This algorithm treats the tracks as a circular list that wraps around from the first track to the last track. [1, 2, 3]

If we solve example 1 and example 2 using C-SCAN algorithm then total head movements are 390 and 390 respectively.

E. *LOOK Algorithm*

Similar to the SCAN algorithm, in LOOK algorithm the head sweeps across the surface of the disk in both directions, performing read & writes operations. However, unlike the SCAN algorithm, LOOK algorithm will reversed the direction of the head when it reached to the last requested track in the current direction of the disk head. [1, 2, 3]

If we solve example 1 and example 2 using LOOK algorithm then total head movements are 220 and 280 respectively.

F. *C-LOOK Algorithm*

C-LOOK algorithm is a variation of C-SCAN algorithm. In C-LOOK algorithm the disk head goes only as far as last request in the current direction & then immediately reverse its direction, without going all the way to the end of the disk. [1, 2, 3]

If we solve example 1 and example 2 using C-LOOK algorithm then total head movements are 270 and 330 respectively.

III. PROPOSED OPTIMIZED TWO HEAD DISK SCHEDULING ALGORITHM (OTHDSA)

In this proposed algorithm, we have again considered two heads and one head is at position 0(first track position) and other at position 200(last track position). One head always moves in right direction from position 0 to 200 and we call this head as left head (HL). Other head always moves towards left direction from position 200 to 0 and we call this head as right head (HR). Firstly, we sort all the requests blocks in ascending order by using any sorting algorithm. In this way, we find the max and min from the sorted list of input requests. To decide that which head moves first, we do some calculations. We find the difference between the position of left head (0) and min value and between the position of right head (200) and max value. We compare these two values and we move towards the direction with lesser value of difference. When that head completes its task (serves all requests less than or equal to 100 if it is left head or serves all requests greater than and equal to 100 if it is right head), it has two options: it can continue or it can stop moving and second head starts moving in other direction. Now we do some calculations to select one from the two cases. We calculate which head will

travel lesser distance in serving the remaining requests and if moving head need to travel less distance then it continues moving otherwise it stops moving and second head starts moving. [4, 6, 7]

A. *Procedure/Algorithm*

```
1. INPUT: n (number of requests), array a [] of size n
having track position of each request.
2. Left_head← 0
Right_head←200
3.  Sort array a[] in ascending order.
4. x ←(a[0]-left_head)
Y←(right_head-a[n-1])
5. If(x<=y) goto step 6 else goto step 7
6. i←0
While (a[i]<=100)
left_head←a[i]
serve_track(left_head)
i←i+1
x←(a[n-1]-a[i-1])
y←(200-a[i])
if(x<=y) goto step 6.1 else goto step 6.2
6.1 while (i<=n-1)
left_head=a[i]
serve_track(left_head)
i←i+1
6.2 j←n-1
While (j>=i)
right_head←a[j]
serve_track(right_head)
j←j-1
while(a[i]>=100)
right_head←a[i]
while (j<=i)
left_head←a[j]
serve_track(left_head);
j←j+1
7.2while(i>=0)
right_head=a[i];
serve_track(right_head);
i←i-1
8. seek_time←((left_head-0)+(200-right_head))
9. Returnseek_time
```

B. *FLOWCHART*

The flowchart of newly developed algorithm has been drawn as below in figure 1.

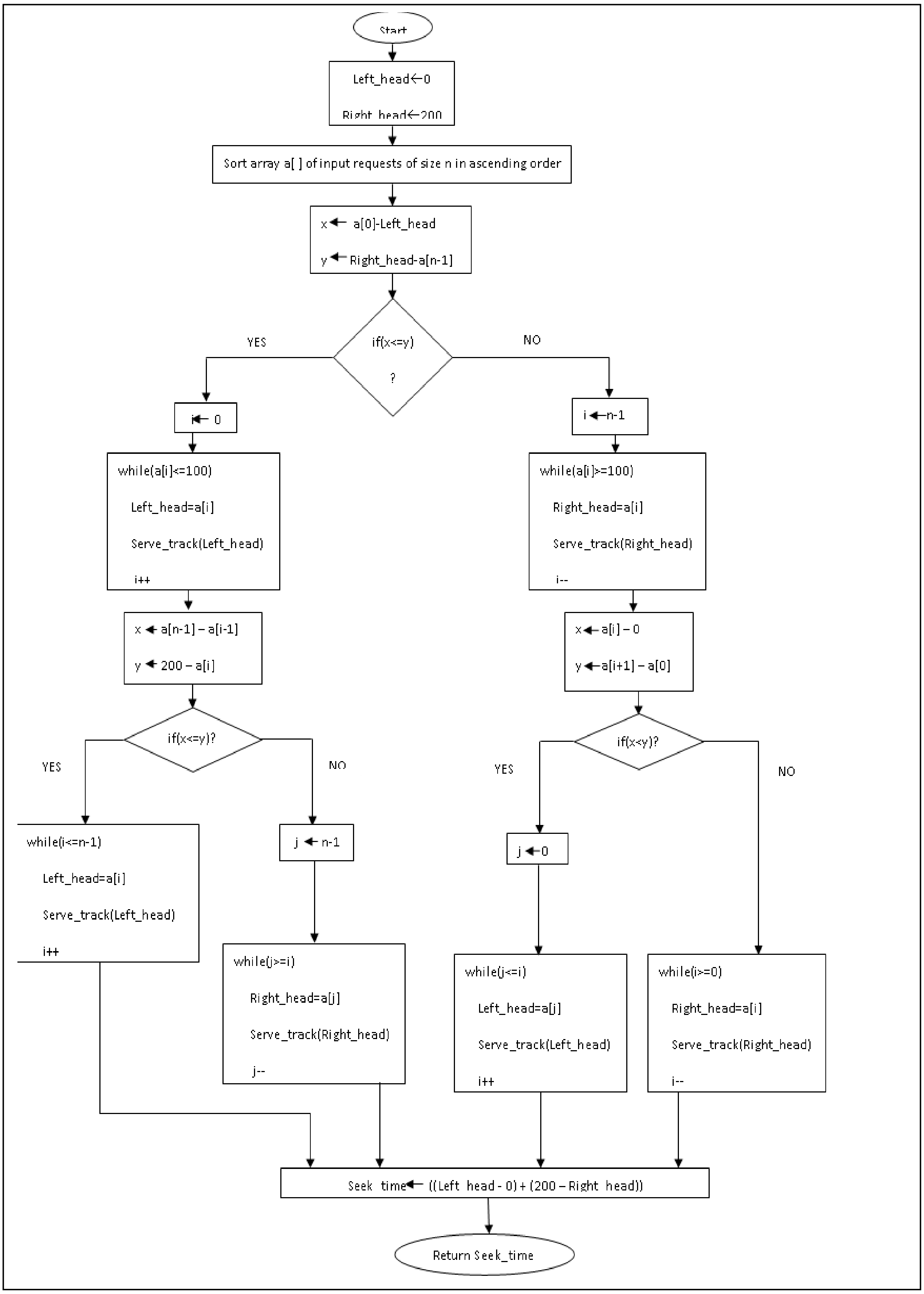


Figure 1. Flow chart for Proposed optimized Two Head Disk Scheduling Algorithm

Consider the same examples:

Example 1: Consider a disk queue with I/O requests to blocks on tracks: Queue (0-200): 60, 90, 40, 20, 110, 70, 150, 130, 160, 120. Head starts at 80.

Example 2: Consider a disk queue wih I/O requests to blocks on tracks: Queue (0-200): 40, 60, 20, 70, 120, 160, 130, 180, 80, 190. Head starts at 80.

According to the rule of proposed algorithm, initial head position of Left_head is 0 and Right_head is at 200. The left head moves towards the right direction from 0 to 200 and right head moves towards the left direction from 200 to 0.

Firstly we sort our input requests in ascending order. For deciding which head moves first we subtract position of left head(0) with minimum of input requests and compare the result with the subtraction of right head(200) position and maximum of input requests, whichever comes smaller moves first.

Considering example 1:

(20-0) < (200-160), hence left head moves first and serves requests till 90th track position. Now, decision is made whether left head continues serving the requests or right starts serving the requests. This decision is based on, whether left head will travel less distance or right head. In our case left head will have to travel 70 tracks while right head will have to travel 90 tracks to complete the I/O access requests. Hence left head continues serving the requests and moves till 160th track.

In this way, seek time comes out to be 160. Head Movements and implementation output is shown in figure 2 and figure 3.

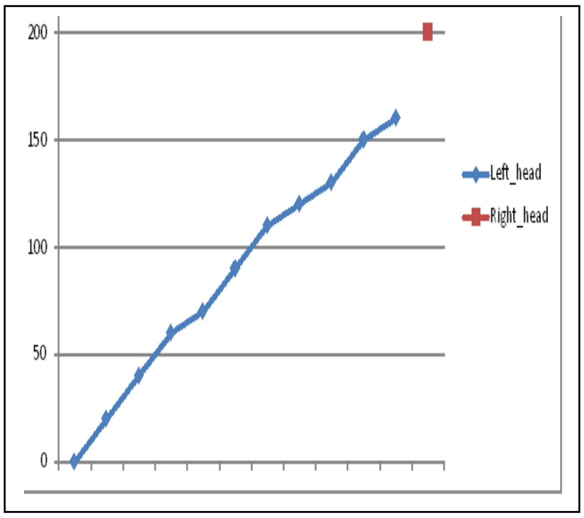
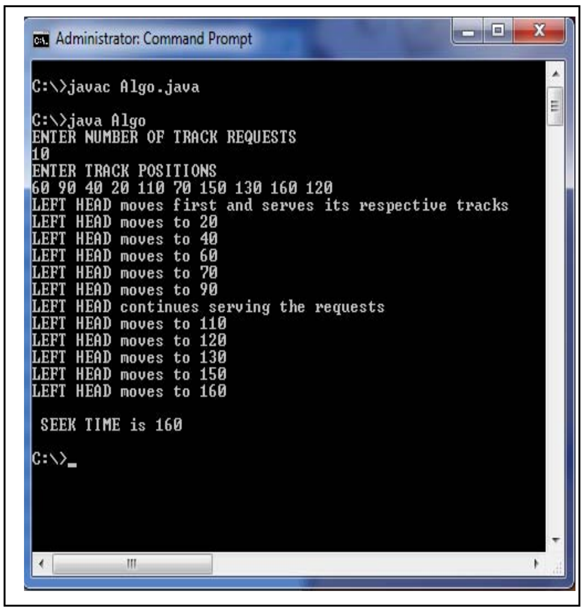


Figure 2. New Proposed Optimized Two Head Disk Scheduling Algorithm Example 1



```
C:\>javac Algo.java

C:\>java Algo
ENTER NUMBER OF TRACK REQUESTS
10
ENTER TRACK POSITIONS
60 90 40 20 110 70 150 130 160 120
LEFT HEAD moves first and serves its respective tracks
LEFT HEAD moves to 20
LEFT HEAD moves to 40
LEFT HEAD moves to 60
LEFT HEAD moves to 70
LEFT HEAD moves to 90
LEFT HEAD continues serving the requests
LEFT HEAD moves to 110
LEFT HEAD moves to 120
LEFT HEAD moves to 130
LEFT HEAD moves to 150
LEFT HEAD moves to 160

 SEEK TIME is 160

C:\>_
```

Figure 3. Implementation Output for Example 1

Considering example 2:

(20-0) > (200-190), hence right head moves first and serves requests till 120th track position. Now, decision is made whether right head continues serving the requests or left starts serving the requests. This decision is based on, whether left head will travel less distance or right head. In our case left head will have to travel 80 tracks while right head will have to travel 100 tracks to complete the I/O access requests, as such left head starts serving the requests and moves till 80th track.

In this way, seek time comes out to be 160. Head Movements and implementation output is shown in figure 4 and figure 5.

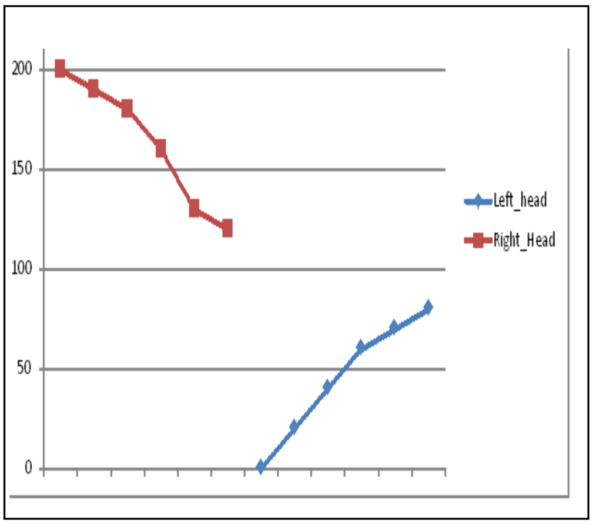
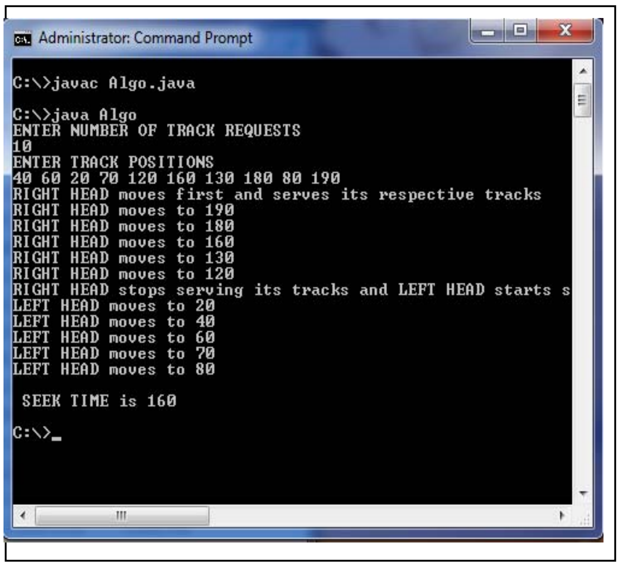


Figure 4.New Proposed Optimized Two Head Disk Scheduling Algorithm Example 2



```
C:\>javac Algo.java

C:\>java Algo
ENTER NUMBER OF TRACK REQUESTS
10
ENTER TRACK POSITIONS
40 60 20 70 120 160 130 180 80 190
RIGHT HEAD moves first and serves its respective tracks
RIGHT HEAD moves to 190
RIGHT HEAD moves to 180
RIGHT HEAD moves to 160
RIGHT HEAD moves to 130
RIGHT HEAD moves to 120
RIGHT HEAD stops serving its tracks and LEFT HEAD starts s
LEFT HEAD moves to 20
LEFT HEAD moves to 40
LEFT HEAD moves to 60
LEFT HEAD moves to 70
LEFT HEAD moves to 80

 SEEK TIME is 160

C:\>_
```

Figure 5. Implementation Output for Example 1

IV. RESULT AND ANALYSIS

A comparative analysis among various existing disk scheduling algorithms and the newly developed Optimized Two Head Disk Scheduling Algorithm has been performed and result has been shown in Table 1.

TABLE 1: Comparative Analysis of disk scheduling algorithms in terms of total number of head movements

S.N.	Name of Disk Scheduling Algorithm	Total Number of Head Movements in Example 1	Total Number of Head Movements in Example 2
1.	FCFS	420	530
2.	SSTF	220	230
3.	SCAN	300	300
4.	C-SCAN	390	390
5.	LOOK	220	280
6.	C-LOOK	270	330
7.	Optimize Two Head Disk Scheduling Algorithm(OTHDSA)	160	160

A. Comparison Graph between Proposed and Existing Disk Scheduling Algorithms:

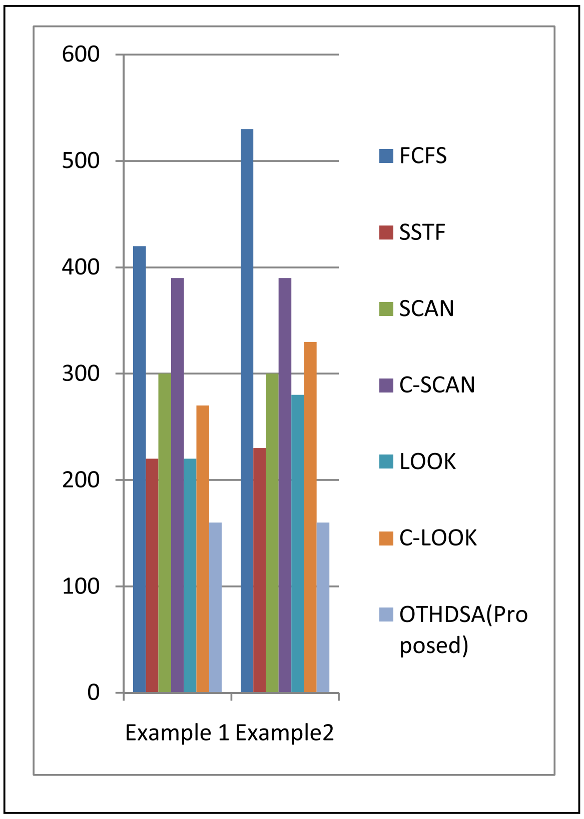


Figure 6.Comparison graph

We can analyse from the above table and comparison graph that new proposed optimized two head disk scheduling algorithm requires minimum head movements in order to service the disk I/O requests. There are several advantages of this proposed algorithm. If all the input blocks are concentrated in the start and end of the disk (means near the position 0 and 200), then this proposed algorithm gives the best result.

V. CONCLUSIONS

In this paper, we proposed and implemented a new disk scheduling algorithm with two heads which imposes almost no performance penalty over the non-real time optimal scheduler, when we have sufficient slack time. With the help of experiments and comparison results, it is clear that proposed algorithm reduces the total number of head movements and always gives less number of head movements in comparison to other scheduling algorithms. The worst case occurs when one input block is near the position 0 and other is near the position 200 and other requests are uniformly distributed between 0 and 200. But the total number of head movements can never be more than the number of tracks in the disk (if disk has 200 tracks then total number of head movements can never be more than 200).

VI. FUTURE WORK

In this paper, we made lot of efforts to improve the performance of disk I/O access; even there are tremendous scopes for the further improvement of disk I/O access by developing innovative disk scheduling algorithm or even enhancing the existing disk scheduling algorithms

REFERENCES

[1] Silberschatz, Peter Bare Galvin, Greg Gagne, Operating System Principles, 9th edition 2014

[2] Andrew S. Tanenbaum , Modern operating system, 2nd edition, 2001

[3] D.M. Dhamdhere, Operating Systems: A Concept-based Approach, second edition, 2006

[4] Yang Hu, MHPR: Multi-head Parallelism and Redundancy Disk Model, IEEE International Conference on Networking, Architecture and Storage, ISBN: 978-0-7695-3741-2, pp 315 – 322, 9-11 July, 2009, Zhang JiaJie, China.

[5] Dees B., Native Command queuing-advanced performance in desktop storage, Potentials, IEEE, ISSN: 0278-6648, Vol. 24, Issue: 4, pp 4-7, 2005

[6] Sandipon Saha, Md. NasimAkhter, MohammodAbulKashem, A New Heuristic Disk Scheduling Algorithm, International Journal of Scientific &Technology Research, ISSN 2277-8616, Volume 2- Issue 1, pp 49-53, January 2013

[7] Huang Y. and Huang J. , Disk scheduling on multimedia storage servers, IEEE Transactions on Computers, Volume 53 , Issue 1, Page 77-82, 2014

[8] Reddy A.L.N., Wyllie, Jim and Wijyaratne, K.B.R., Disk scheduling in a multimedia I/O system, ACM Transactions on Multimedia Computing, Communications, and Applications. Volume 1 Issue 1, Pages 37-59 2005

[9] C. Reummler and J. Wilkes, An introduction to disk drive modeling, IEEE Computer, Vol. 27, Issue: 3, Pages 17-19, March 2010

[10] Nidhi and Dayashankar Singh, A New Optimized Real-Time Disk Scheduling Algorithm, International Journal of Computer Applications (0975-8887), Volume 93, No-18, May 2014.

[11] R. K. Yadav, A. K. Mishra, N. Prakash and H.Sharma, An Improved Round Robin Scheduling Algorithm for CPU scheduling, International Journal on Computer Science and Engineering (IJCSE), Vol. 02, No. 04, Pages: 1064-1066., 2010

[12] Yaashuwanth C and Dr.R. Ramesh, Design of Real Time scheduler, Simulator and Development of Modified Round Robin Architecture, International Journal of Computer Science and Network Security, vol. 10, No.3, 2010

[13] S. K. Panda, S. K. Bhoi, "An Effective Round Robin Algorithm using Min-Max Dispersion Measure". International Journal on Computer Science and Engineering (IJCSE), ISSN: 0975-3397 Vol. 4 No. 01.

[14] Tarek Helmy, AbdelkaderDekdouk, "Burst Round Robin as a Proportional-Share Scheduling Algorithm", In Proceedings of The fourth IEEE-GCC Conference on Towards Techno-Industrial Innovations, pp. 424-428, Bahrain