OPERATING SYSTEMS-CSE 2005

DISK I/O PERFORMANCE OPTIMISATION TECHNIQUES

PARITOSH PANDEY **17BEC0308**

ANKIT KUMAR **17BEC0591**

TARUNYA TRIVEEDI **17BEC0784**

IISHAAN KAUSHIK **17BCE0662**

ABSTRACT:

Input/output (I/O) scheduling is a term used to describe the method computer operating systems decide the order that block I/O operations will be submitted to storage volumes. I/O Scheduling is sometimes called 'disk scheduling'.

I/O schedulers can have many purposes depending on the goal of the I/O scheduler, some common goals are:

- To minimize time wasted by hard disk seeks. - To prioritize a certain processes' I/O requests.
- To give a share of the disk bandwidth to each running process.
- To guarantee that certain requests will be issued before a particular deadline.

INTRODUCTION:

There are many different types of I/O disk schedulers present in the today's scenario, therefore there arises a need to find the best among them for our day to day purposes ,so that we can extract the maximum capabilities of our devices without any loss of data, wastage of time and least effort .Our team's project is predominantly based on Linux kernel, since Linux is the richest OS in terms of I/O schedulers, and it includes many variations such as noop, anticipatory, deadline, and CFQ schedulers.

Speaking of write performance, you would usually find your IO either very sequential – such as writing video blocks one after the other. Or rather fairly random – such as user driven DB updates in places you can't really expect them. Tuning the latter is a rather harder task as the input would be rather random.

Especially when tuning for IO, just use more memory. If, for instance, you need to write data blocks to the disk, perhaps you can buffer them as much as you can – and engage in disk writes only when really necessary or when it is more convenient to do so.
Same goes for random write IO – buffer your requests and serve them to your DB in large chunks – let the layer underneath handle the multiple IOs more efficiently. E.g. you have many IO write requests, instead of serializing them to the disk in the order they have arrived, cache many of them, and let the operating system queue them in the most efficient way.
If your application is actually a database – there are numerous parameters you can configure to get the database to work much better. Work them all with your loyal DBA for maximum performance.

SYSTEM IMPLEMENTATION:

FCFS:

First Come First Served (FCFS): This algorithm simply service the request in FCFS basis, in which the earliest arriving request is serviced first if the load becomes heavy, FCFS can be long waiting times

FIFO is an acronym for first in, first out, a method for organizing and manipulating a data buffer, where the oldest (first) entry, or 'head' of the queue, is processed first. It is analogous to processing a queue with first-come, first-served (FCFS) behaviour: where the people leave the queue in the order in which they arrive.

-Serves I/O requests with least number of CPU cycles.
- Best for flash drives since there is no seeking penalty.
- Good data throughput on db systems

SSTF:

This algorithm service the request with least seek time from the current head position before moving the head to service other request. This algorithm gives substantial improvement in performance compared to FCFS

This is a direct improvement upon a first-come first-served (FCFS) algorithm. The drive maintains an incoming buffer of requests, and tied with each request is a cylinder number of the request. Lower cylinder numbers indicate that the cylinder is closer to the spindle, while higher numbers indicate the cylinder is farther away. The shortest seek first algorithm determines which request is closest to the current position of the head, and then services that request next.

The shortest seek first algorithm has the direct benefit of simplicity and is clearly advantageous in comparison to the FIFO method, in that overall arm movement is reduced, resulting in lower average response time

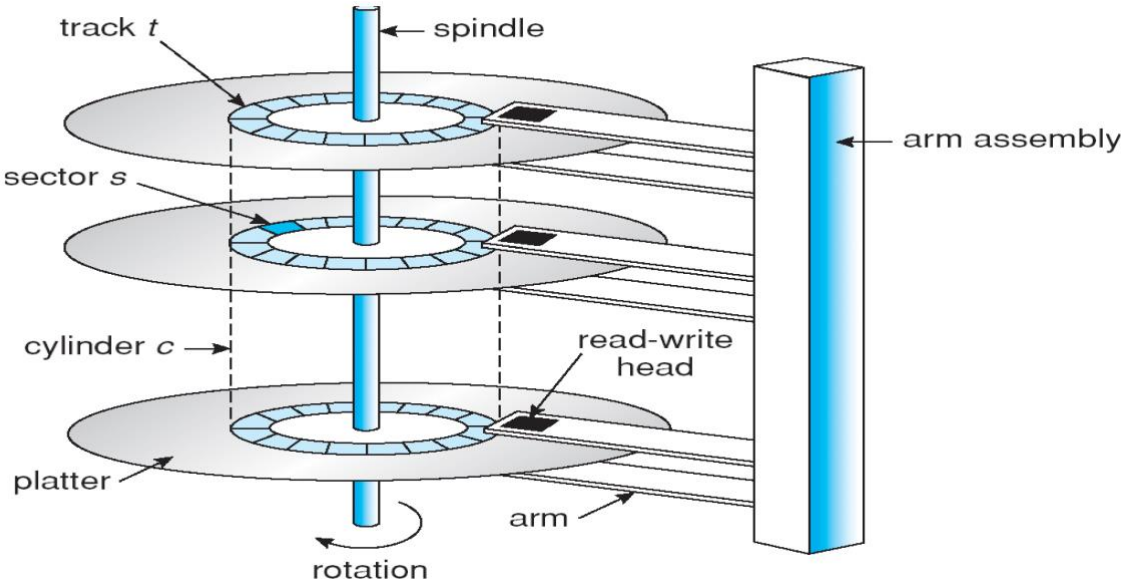
SCAN:

The elevator algorithm (also SCAN) is a disk scheduling algorithm to determine the motion of the disk's arm and head in servicing read and write requests.

This algorithm is named after the behavior of a building elevator, where the elevator continues to travel in its current direction (up or down) until empty, stopping only to let individuals off or to pick up new individuals heading in the same direction.

From an implementation perspective, the drive maintains a buffer of pending read/write requests, along with the associated cylinder number of the request. (Lower cylinder numbers generally indicate that the cylinder is closer to the spindle, and higher numbers indicate the cylinder is farther away.)

BASIC ARCHITECTURE:



IMPLEMENTATION:

The Code required under the following disk I/O Schedulers are as follows:

Program Code for the above schedulers:

```
#include<stdio.h> #include<math.h>

void fcfs(int noq, int qu[10], int st)

{  int i,s=0;

for(i=0;i<noq;i++)

 {   s=s+abs(st-

qu[i]);   st=qu[i];

  }  printf("\n Total seek time

:%d",s);

} void sstf(int noq, int qu[10], int st, int

visit[10])

{     int  min,s=0,p,i;

while(1)                {

min=999;

for(i=0;i<noq;i++)

if (visit[i] == 0)

   {

    if(min > abs(st - qu[i]))

     {
```

```
      min = abs(st-qu[i]);
p = i;
       }
     }
 if(min == 999)
break;
visit[p]=1;  s=s
+  min;      st  =
qu[p];
  }  printf("\n Total seek time is:
%d",s);
 } void scan(int noq, int qu[10], int st, int
ch)
{  int i,j,s=0;
for(i=0;i<noq;i++)
 {     if(st  <  qu[i])    {
for(j=i-1; j>= 0;j--)
  {    s=s+abs(st -
qu[j]);    st = qu[j];

   }   if(ch
== 3)
   {    s = s + abs(st
- 0);    st = 0;
```

```
   }   for(j = 1;j <
noq;j++)
  {    s= s + abs(st -
qu[j]);    st = qu[j];
   }   break;  } } printf("\n Total
seek time : %d",s);
} int main() {    int n,qu[20],st,i,j,t,noq,ch,visit[20];
printf("\n Enter the maximum number of cylinders : ");
scanf("%d",&n);  printf("enter number of queue
elements"); scanf("%d",&noq);
printf("\n Enter the work queue"); for(i=0;i<noq;i++)
{  scanf("%d",&qu[i]);
  visit[i] = 0;   }  printf("\n Enter the disk head
starting position: \n");  scanf("%d",&st);  while(1)
 {   printf("\n\n\t\t MENU \n");
printf("\n\n\t\t 1. FCFS \n");
printf("\n\n\t\t 2. SSTF \n");
printf("\n\n\t\t 3. SCAN \n");
printf("\n\n\t\t 4. EXIT \n");
printf("\nEnter your choice: ");
scanf("%d",&ch);   if(ch > 2)
   {
   For(i=0; I <noq;i++)
 For(j=i+1;j<noq;j++)
```

```
 if(qu[i]>qu[j])   {
 t=qu[i];
   qu[i] = qu[j];
qu[j] = t;    }
   }
switch(ch)
    {
    case 1: printf("\n FCFS \n");
printf("\n*****\n");
fcfs(noq,qu,st);            break;

    case 2: printf("\n SSTF \n");
printf("\n*****\n");
sstf(noq,qu,st,visit);
break;     case 3: printf("\n
SCAN \n");
printf("\n*****\n");
scan(noq,qu,st,ch);
break;     case 4: exit(0);
 }
 }
}
```

ANALYSIS BY EXAMPLES:

- Suppose a disk drive has 5000 cylinders, numbered 0 to 4999. Consider a disk queue with requests for i/o to blocks on cylinder : 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 . Assume that disk head is currently at cylinder 143.

  FFCS-786.77

  SSTF-193.88

  SCAN-1085.44

- Suppose a disk drive has 100 cylinders, numbered 0 to 99. Consider a disk queue with requests for i/o to blocks on cylinder : 33, 72, 47, 8, 99, 74, 52, 75. Assume that disk head is currently at cylinder 63.

  FFCS-36.75

  SSTF-21.25

  SCAN-20.25

- Suppose a disk drive has 200 cylinders, numbered 0 to 199. Consider a disk queue with requests for i/o to blocks on cylinder : 98, 183, 37, 122, 14, 124, 65, 67. Assume that disk head is currently at cylinder 53.

  FFCS-80

  SSTF-29.5

  SCAN-29.5

SELECTION OF ALGORITHMS

To determine a particular algorithm, predetermined workload and the performance of each algorithm for that workload is to be determined

The choice of algorithm depends on expected performance and on implementation complexity

SSTF is common and has a natural appeal. SCAN gives better performance than FCFS and SSTF. The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary

- FIFO -First in First Out- Fairest of them all
- SSTF -Shortest service time first -High utilization, small queues
- SCAN -Back and forth over disk -Better service distribution

OBSERVATIONS

The FCFS performs operations in order requested. No reordering of work queue since it processed disk requests according to its arrival. There is no starvation and all the requests are serviced but it doesn't provide fastest service

The Shortest Seek Time First (SSTF) selects the disk I/O request that requires the least movement of the disk access arm from its current position regardless of direction. It also reduces the seek time compared to FCFS but in this algorithm, I/O requests at the edges of the disk surface may get starved

The SCAN algorithm go from the outside to the inside servicing requests and then back from the outside to the inside servicing requests. It also reduces variance compared to SSTF

OPTIMISED ALGORITHM

Assuming that the disk controller and disk drive are busy doing something. The request that can't be serviced by the hardware as soon as arrived is placed in the queue of pending requests. Assuming that the requests are in the random order. Now apply any sorting method to sort the requests in the ascending order and then obtain the midpoint request from the sorted queue. Once we know the midpoint request, then we will compare the current head pointer with the midpoint request. If the current head pointer is less than the midpoint request the we will service the requests one by one from initial request until we reach the last request in the sorted list. Else we will service the requests one by one from the last request until we reach the initial request in the sorted list. In either case the scanning should be done from the current head pointer. Finally we will calculate the total number of head movement and average seek time

- Declaration and Initialization
- A [ ] // the list of pending requests which are waiting to be serviced. It is of type int
- N // Total number of pending waiting requests in A [ ]. Initially it is 0 and is of type int
- CHP // Current Head Position
- THM // Total number of Head Movement. Initially it is 0 and is of type int
- AST // Average Seek Time. It is of type float
- MPR // Mid Point Request obtained from the sorted array. It is of type int
- low // indicates the index of the starting element in A [ ]
- high // indicates the index of the last element in A [ ]
- Use any sorting method to sort the pending requests in the unsorted array A [ ].
- After sorting the array, obtain the Mid Point Request from the sorted array A [ ] MPR = (low + high) / 2 // this will be needed for our future comparisons

- Read the CHP CHP = Initial disk head position
- for i = 0 to N-1
- do { if (CHP < MPR) { Scanning will start from A [i] up to the last element A [i-1] from the CHP
- // Calculate Total no. of Head Movement and Average Seek Time THM = CHP + | A i-1 – A i | AST = THM / N }
- // end if else if (CHP > MPR) { Scanning will start from last element A [i-1] to A [i] from the CHP
- // Calculate Total no. of Head Movement and Average Seek Time THM = CHP + | A i-1 – A I | AST = THM / N } // else if break; }
- // end for
- Stop the algorithm

RESULTS AND COMPARISON

- **CASE I:** **CASE II:** **CASE III:**

FFCS-36.5	FFCS-80	FFCS-786.77
SSTF-21.25	SSTF-29.5	SSTF-193.88
SCAN-20.25	SCAN-29.5	SCAN-1085.44
SORT QUEUE-15.75	SORT -24.87	SORT-193

CONCLUSION

The performance of disk scheduling algorithm depends heavily on the total number of head movement, seek time and rotational latency. With the classical approach of disk scheduling algorithm, few algorithms like SSTF will be the most efficient algorithm compared to FCFS, SCAN disk scheduling algorithm with respect to these parameters

Compared to the classical approach of disk scheduling algorithm, our results and calculations show that our proposed algorithm reduces the number of head movement and seek time thus improving the performance of disk bandwidth for disk drives. For few requests, our algorithm is equal to SSTF disk scheduling algorithm

REFERENCES:

1) C. Ruemmler and J. Wilkes, “An introduction to disk drive modeling,” IEEE Computer, vol. 27, pp. 17–28, 1994.

2) M. Seltzer, P. Chen, and J. Ousterhout, “Disk Scheduling Revisited,” in Proceedings of the Winter Usenix, 1990.

3) M. Hofri, “Disk Scheduling: FCFS vs. SSTF revisited,” Communication of the ACM, vol. 23, no. 11, November 1980.

4) D. Jacobson and J. Wilkes, “Disk Scheduling Algorithms based on Rotational Position,” Concurrent Systems Project, HP Laboratories, Tech. Rep. HPLCSP917rev1, 1991.

5) E. Coffman, L. Klimko, and B. Ryan, “Analysis of Scanning Policies for Reducing Disk Seek Times,” SIAM Journal on Computing, vol. 1, no. 3, September 1972.

6) R. Geist and S. Daniel, “A Continuum of Disk Scheduling Algorithms,” ACM Transactions on Computer Systems (TOCS), vol. 5, no. 1, February 1987.

7) B. Worthington, G. Ganger, and Y. Patt, “Scheduling Algorithms for Modern Disk Drives,” in Proceedings of the ACM Sigmetrics, 1994.

8) 一半君, Most up to date guide on CPU governors and I/O schedulers by gsstudios.

9)Linux programmer’s manual for fdatasync. http://www.kernel.org/doc/manpages/online/pages/man2/fsync.2.html.

10) Disk IO benchmarking in the cloud : http://blog.cloudharmony.com/2010/06/disk-iobenchmarking-in-cloud.html.

11)Carl Henrik Lunde Improving Disk I/O Performance on Linux, Master Thesis, 2009
: http://home.ifi.uio.no/paalh/students/CarlHenrikLunde.pdf

12)Seelam, Seetharami, et al. ”Enhancements to Linux I/O Scheduling.” Proc. of the Linux Symposium. Vol. 2. 2005.