A HARD REAL-TIME SCHEDULER ALGORITHM FOR SOLID STATE DEVICE

INDURAJ.P.R

Department of electronics and communication,Bharath university,BIST, selaiyur- India.
Email: induraj.gandhian@yahoo.com

Abstract: This paper presents an approach to use the solid state devices in hard real time application where delay in retrieval or write of data to and fro to them can result in a catastrophe. This new algorithm proposes a new approach of scheduling by considering the deadline's associated with data's, multiple synchronous read or write requests along with the algorithm for overcoming the problem of performing new block writes resulting in I/O bottleneck.

Keywords: *Deadlines, multiple synchronous requests, new block writing.*

I. INTRODUCTION

Two Decade ago when magnetic storage devices were introduced they started influencing themselves in the area of storing large set of data's like entertainment files, etc. due to this invention of many algorithm's to make them effectively efficient was at the peek. So far many algorithms have been proposed as to how to make write and read in a magnetic storage device and even many have been proposed in order to reduce the seek time and rotational latency due to spin of magnetic head in order to make read or write, some algorithms considerably notable are FCFS, STFS, SCAN, C-SCAN, LOOK algorithm, etc [1, 2, 3, 4].

But with the advent of solid state devices, their reduced size and higher data capacity and higher data transfer rate became a notable factor that led to the use of solid state device considerably go high in the vast fields. With their high data capacity, no seek time, reduced size and reduction in prices they are to create an era where magnetic disk will rarely seen as of tape recording devices & gramophones now.

With the portable and efficient nature of solid state devices like flash drives they even tend to replace magnetic storage devices used in black box of avionics today and in other crucial applications where role of magnetic storage devices is significant[5,6,7].

As the solid state drives do not possess a rotating head or a rotating disk the algorithms proposed to reduce seek rate, rotational latency for magnetic storage devices as mentioned in the operating system concept book cannot be implemented. Considerably other operational characteristics like FTL, different policies in FTL have adverse effects on the performance of solid state devices.

Even though deadline based disk scheduling algorithm EDF [8] was proposed, considering the deadline alone ignoring the relative position of data will not produce satisfactory result in hard real time system, many schedulers based on algorithms had been
proposed with one considering to being better than other by overcoming the limitation of its predecessor algorithm. Some of the scheduling mechanism that have been proposed like priority scan (PSCAN), earliest deadline scan, feasible deadline scan (FD-SCAN), scan-EDF, shortest earliest deadline by order/value (SSEDO, SSEDV), etc [9, 10, 11, 12] all these apply for a magnetic device and are not directly applicable to solid state devices.

With modern system with high end processor, the use of magnetic storage device results in performance deterioration. With Moore's law stating that number of transistors used doubles every decade resulting in increase of processor performance but there exist a severe limitation in terms of data transfer rate involved between processor and magnetic storage device as the data transfer rate is increasing only at the rate of half compared to that of processors.

With deadline's involved in highly sensitive hard real time systems, the idleness caused by anticipatory algorithm [13] for overcoming deceptive idleness would cause building of infinite queue, loss of request, loss of data and even in catastrophe as the scheduler tends to wait anticipating a request to be received for a service to be performed on the same block. And in order to achieve accurate performance the resources must be used efficiently and effectively. This becomes not possible due to erase on write policy and many other policies being implemented on solid state devices. There is also a need to schedule the requests to acquire the device thus here we propose an algorithm considering the case of both multiple read and write requests that would appear in real time application and to overcome the disadvantage of writing to new block for each request.

In section II we introduce the description of algorithm that have been previously proposed depicting their incapability in highly sensitivity environments and in section III we describe our algorithm to be implemented on scheduler and section IV being the conclusion

II. SCHEDULERS

The mechanical disks in general cause i/o bottle necks in large performance oriented data intensive applications, this is due to those poor read/write performance but flash based solid state devices eliminate the i/o bottle neck problem. With the stress on attaining performance and reliability many scheduling methods were proposed. Conventional scheduling methods to achieve fairness[14, 15] and quanta based scheduling[16,17] fail to recognize unique flash characteristics like substantial read blocked by write, although i/o anticipation was proposed as performance enhanced seek reduction technique for mechanical drives, but its role for maintaining farness has been largely ignored. The quanta based scheduling suppress the i/o parallelism between concurrent tasks which degrades the i/o efficiency of flash devices.

Many file system approaches such as log structured file systems have been proposed to use the special characteristics of the solid state devices and to alleviate the problem of large seek times. SSD's are generally organized as multiple banks [18] with each block size ranging from a minimum of 512 byte; hence here data's are stored and accessed in blocks with a minimum granularity of 512 byte. To a data to be written, an erase on write policy needs to be implemented by which necessary blocks are completely erased and a new data's are written. Other than this, process may request data to be written to a new block forsaking the last used block in which further operation can be carried out. Because of which the performance efficiency of flash device reduces.

Adaptive disk scheduling [19] tends to overcome a phenomenon called deceptive idleness. It is made clear that disk schedulers are work consuming, meaning that they select a request for service before the previous requests are completed, so when synchronous requests are received, each process maintains at least one outstanding request's at any time. When schedulers make decision too early it selects a request from other process with no prior knowledge that the last processor itself will request for service as soon as the current service is finished. Thus causing the incoming request's to wait for service. To overcome the deceptive idleness, concept of adaptive scheduling was used in which scheduler after finishing a request; the scheduler tends to wait anticipating a request to be received for a service to be performed on the same block. Though it has been assumed here that time spent on anticipating is negligible, in hard real time systems expecting multiple synchronous requests, even a small time spent on anticipating by making requests from other process to wait, could lead to building of infinite queue and even in a catastrophe.

The method proposed in [18] although achieves performance by anticipating and looking forward and backward of the block serviced. This doesn't provide a methodology to tackle building of queues and applicability in hard real time system becomes a questionnaire. Even many algorithms like noop scheduler, deadline scheduler, complete fair queuing has been proposed for magnetic disks they become applicable with certain constraints like seek time, rotational latency being omitted, but suffice for applicability in hard real time.

III. ALGORITHM

In the advent of proposing algorithm for solid state devices used in highly sensitive real time environments to overcome the problem of new block write we develop this new algorithm for multiple read/write requests satisfying hard real time constraints.

Our algorithm considers two entirely different cases, one in which the write request to solid state devices is asynchronous and another with multiple synchronous request. Here we propose an algorithm providing solutions to both cases considering the way of satisfying real time constraints like deadline associated with each requests.

Case I:

When the solid state device receives asynchronous write request (Rn=1) from different processes as in fig (a) and fig (b), the solid state device implemented by our algorithm checks whether the received request are to write a data to the same block (B) that was used lately or to a different block (DB). If the request is to write a data to new block we anticipate for a certain time till the time of arrival of the next request which is negligible, the algorithm proposed by[18] anticipates for a request to same block during each time a new request arrives irrespective of the request waiting to be served in different block thus resulting in further idleness which is not suitable for hard real time system but our algorithm here anticipates only when there are no multiple synchronous request thereby not making other request to wait on queue i.e. building up an infinite request queue, resulting in successful schedule of all hard real time requests.

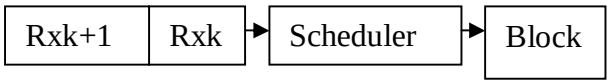
In general, if two requests to write data within the same are made, the algorithm by [18] chooses the largest of the request received, writes the data to the block and proceeds to the next waiting smallest request. This scheduling of multiple request based on the largest of request received will prove wrong in real time if the largest request possessing larger deadline is scheduled to be serviced other than the smallest request with small deadline resulting in the miss of the request with least deadline and also this method works fine with multiple requests staked

in the queue it doesn't when multiple synchronous requests are issued.

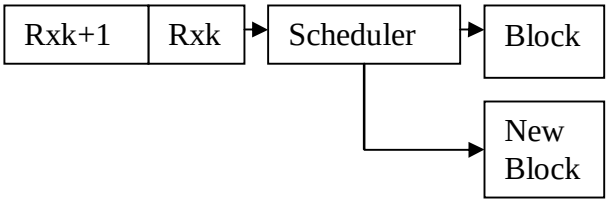
Thus we propose the idea of deadline based scheduling along with the methodology to overcome the new block write problem.

Here if a request is made to write within the same block, our algorithm services the request and then service the other request, and if the request is made to write data outside the block, the algorithm anticipates staying in the same block for a time by which no queues are build and no requests are missed, if the deadline of the anticipated request (DARn+1 where n=xk) has the deadline greater than the received request (DRxk), we service the received request requesting the write of data in different block and return to the previous block to service the anticipated request.

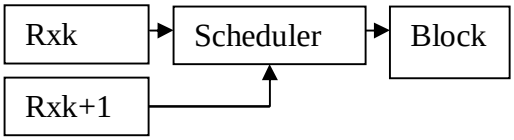
while if the case is such that the if the deadline of the anticipated request is lesser than the deadline of the received request, then our scheduler serves the anticipated request and then serves the received request, after servicing the received request if no new requests are received the scheduler waits to receive a request in this new block and acts accordingly.



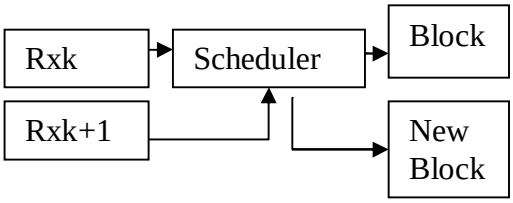
A. Asynchronous Request to same block



B. Asynchronous Request to new block



C. Synchronous Request to same block



D. Synchronous Request to new block

Case II:

Here we depict the case of how our algorithm would handle when multiple synchronous requests are received as in fig c and fig d.

When multiple requests are made we assume that for each request there exists a deadline associated with it. So while scheduling, the deadlines associated with each request are taken into account with the knowledge of preventing new block write operation, i.e. say that two simultaneous requests are made. The scheduler first determines which of the two new requests are made to the same block.

If the two requests are made to the same block then scheduler finds the request with least deadline and responds to it first and then serves the request with larger deadline.

If the two requests made are to write their own data to different blocks other than the last operated block, then our scheduler finds the request with the least deadline serves it and then starts servicing the request with larger deadline. If the first request served is in different block it returns to the previous block continues to serve the request made to the same block and when the deadline of the request made to the same block is least it services it and moves to service the request made to write in different block.

If requests are made to write into both within the block and outside the block at the same time, the scheduler first compares the deadline associated with each request's. If the deadline associated with the request to write within the same block has the least deadline the scheduler responds allowing a write to be carried in the same block, but if the deadline associated with the request to write outside the block has the least deadline then our scheduler services the request to write the data outside the block and returns to the previous block thus servicing the waiting request to write on the same block.

ALGORITHM:

1. Determine the latest serviced block (B)

2. If R=1 i.e. R={Rxk @ t1, Rxk+1 @ t2,…,R= Rxk+n @ tn }, where xk= {0,1,..n}

If (Rxk = =B)

- Serve Rxk

If (Rxk+1 != B)

If (DARn+1>DRxk+1) where n = xk+1

- Serve Rxk+1 then
- Serve ARn+1

If (DARn+1< DRxk+1)

- Serve ARn+1 then
- Serve Rxk+1

3. If Rn=0 i.e. R={Rxk,Rxk+1..Rn}@t1

If (Rxk && Rx+1 ==B)

3.1.1 If (DRxk < DRxk+1)

- Serve Rxk then
- Serve Rxk+1

3.1.2 Else If (DRxk+1 > DRxk)

- Serve Rxk+1 then

- Serve Rxk

If (Rxk && Rxk+1 != B)

3.2.1 If (DRxk<DRxk+1)

- Serve Rxk then
- Serve Rxk+1

3.2.2 Else If (DRxk+1 < DRx)

- Serve Rxk+1 then
- Serve Rxk

If (Rx= =B && Rx+1 != B)

3.3.1 If (DRx < DRxk+1)

- Serve Rxk then
- Serve Rxk+1

3.3.2 Else If (DRxk+1 < DRxk)

- Serve Rxk+1
- Return to previous block
- Serve Rx

IV. CONCLUSION

The same algorithm also applies to solving the case of read on write block where multiple read requests will cause a write request to be bypassed and write on read block in which multiple write requests will cause a read request to be bypassed in order to avoid I/O bottle neck and also in the cases of multiple read requests. It is evident from our algorithm that multiple synchronous write request situations in hard real time system can be handled effectively by proceeding with our algorithm considering the deadline, multiple request situations and anticipation.

REFERENCES

[1] EG.Coffmanand, M.Hofri.Onthe, "*Expected Performance of Scanning Disks*" SIAM Journal of Computing, 10(1):60–70, February 1982.

[2] EG.Coffman, LA.Klimko, and B.Ryan. "*Analysis of Scanning Policies for Reducing Disk Seek Times*" SIAM Journal of Computing, 1(3): 269–279, September1972.

[3] PJ.Denning "*Effects of Scheduling on File Memory Operations*" In Proceedings of AFIPSSJCC, page 9–21, 1967.

[4] R.Geistand, S.Daniel, "*A Continuum of Disk Scheduling Algorithms*" ACM Transactions on Computer Systems, 5(1):77–92, February 1987.

[5] P. Reiher, A. Wang, G. Kuenning and G. Popek, "*The Conquest file system: Better performance through disk/persistent-ram hybrid design*", ACM Trans on Storage, 2006.

[6] M.Wu and W.Zwaenpoel, "*eNVY: A non-volatile, main memory storage system*", ACM ASPLOS, 1994.

[7] E. Miller, S. Brandt and D. Long, "*High performance reliable MRAM enabled file system*", Proc. of USENIX HOTOS, May 2001

[8] CL.Liuand, JW.Layland, "*Scheduling Algorithms for Multi-programming in a Hard Real Time Environment*". Journal of the ACM, 30:47–61, 1973

[9] R.K.Abbottand, H.Garcia-Molina, "*Scheduling I/O Requests with Deadlines: A Performance Evaluation*" In Proceedings of RTSS, page 113–124, December1990

[10] MJ.Carey, R.Jauhari, and M.Linvy.Priorityin, "*DBMS Resource Scheduling*" In Proceedings of the 15th VLDB Conference, 1989.

[11] S.Chen, J.A.Stankovic, J.F.Kurose, and D.Towsley "*Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems*", Journal of Real-Time Systems, 3:307–336, 1991

[12] A.L.Narasimha Reddy and J.Wyllie, "*Disk Scheduling in Multimedia I/O System*" In Proceedings of ACM Multimedia'93, Anaheim, CA, page 225–234, August1993

[13] S. Iyer and P. Druschel, "*Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O*", Proc of ACM SOSP, 2001

[14] J.Bruno, J.Brustoloni, E.Gabber, B.Ozden, and A.Silberschatz. "*Disk scheduling with quality of service quarantees*".In IEEE Int' lConf.on Multimedia Computing and Systems, pages 400–405, Florence, Italy, June1999

[15] W.Jin,J.S.Chase,andJ.Kaur "*Interposed proportional sharing for a storage service utility*" In ACMSIGMET-RICS, pages37–48, NewYork, NY, June2004

[16] J.Axboe, "*Linux block I/O—present and future*" In Ottawa Linux Symp, page 51–61, Ottawa, Canada, July2004

[17] M.Wachs, M.Abd-El-Malek, E.Thereska and G.R.Ganger Argon "*Performance insulation for shared storage servers*" .In FAST'07 : 5th USENIX Conf.on File and Storage Technologies, pages 61–76, SanJose, CA, Feb 2007.

[18] Marcus Dunn and A. L. Narasimha Reddy, "*A New IO scheduler for solid state devices*" Publisher by Texas A&M University

[19] Sitaram Iyer, Peter Druschel "*Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O*" in proceedings of 18th ACM symposium on operating systems principles (SOSP 2001).

◈◈◈