International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 10 (2015) pp. 25175-25188

http://www.ripublication.com

A Parallel Disk Scheduling Algorithm for Multicore Architectures

PranavPATEL[1], Chirag PATEL[1], RutulSHAH[1] and VijayarajanV.[1,*]

[1]School of Computing Science and Engineering,
VIT University, Vellore, Tamilnadu, India
([*]Corresponding author; e-mail:pranav.uv@gmail.com)

Abstract

During the previous years, the speed of processors along with primary memories has been greatly increased. However during this period, there have not been significant improvements in the speed of secondary memory as magnetic disk when compared to the advancement in speed of processors and primary memories. Hence, the speed of disk can be a bottleneck on overall performance of a system. The scheduling algorithms used in secondary memory as disk are one of the major factor on which the performance of a disk depends. New processor technologies included more than one cores inside them. Commonly used algorithm does not take advantage of this. If a disk scheduling algorithm can be made so that it can take advantage of multiple cores, its performance can be improved. It motivates the implementation of a parallel algorithm that takes the advantage of having multiple cores in a processor. This algorithm makes scheduling decision by running on more than one core at a time to perform its internal functions. It has been observed that the proposed algorithm is more efficient than the same algorithm running on a single core which is the case with conventional algorithms. This reduces the time taken by algorithm and improves the overall disk performance.

Keywords: primary memory, secondary memory, parallel algorithm, disk scheduling, OpenMP

Introduction

It can be said that the performance of a computer system mainly depends on speed of microprocessor, speed of primary memory and speed of secondary memory used inside it. This is true for today's common computer architectures. Here speed of the

microprocessor is measured as how many numbers of instructions it can perform in given time period while speed of the memories is measured as how many numbers of information bits it can read or write in given time period. Today's operating systems tries to make an efficient use of all these memories for the convenience of its users. All memories play crucial role in the program execution. There for performance and responsiveness of system dependents on the speed of these memories. In recent years microprocessor technology has seen great advancements. This advancement has brought improvement in computational power as well as the operating speed of the microprocessors. Today's new generation microprocessors are faster in computation then the previous generation microprocessors by a factor of two or even more. Similar is the case with the primary memory technologies. Semiconductor technology has also seen good advancements in the previous years. Capacity and the speed of the secondary memories have increased greatly in the recent years. Overall system performance might not be improved just by use of faster microprocessor, Overall system performance can be improved by use of a faster microprocessor along with faster primary as well as relatively fast enough secondary memory. Because of the advancement in semiconductor technology it can be said that the improvement in speed of primary memory has matched the improvement in performance of microprocessors. Today's microprocessors employ multicore architecture which means they have multiple cores inside them which shares the work load given to the microprocessor. Along with this, simultaneous multithreading (Hyper threading) is used to improve utilization of each core. There for today's microprocessors can perform task much faster then before, they can process large amount of data in considerably less amount of time. This means that more amounts of data need to be brought from secondary memory to the main memory, and need to be given to the microprocessor in relatively less amount of time. There for to maintain overall performance of the system adequately faster secondary memory is necessary. However in last few years secondary memories as magnetic disk has not seen much improvements in speed that matches the improvements in microprocessors and primary memory. There for speed of secondary memory is a bottle neck to overall performance of a system. A magnetic disk consists of several parts underneath such as platters, read-write head, spindle, etc. Platters store information bits in circular form on the surface called tracks. Read write head moves back and forth over this track until it finds the requested track. Movement of read-write head requires significant amount of time when compared to other times required in the disk to bring data to primary memory. Disk scheduling algorithm is used to determine which request is to be granted next, among all other requests that are waiting in queue. Primary goal of the disk scheduling algorithm is to minimize the number of seek and hence seek time, to improve speed of disk. There for the disk scheduling algorithm is one of the major factor on which performance of a disk depends. Improvement in disk scheduling algorithm improves overall performance of the disk. In this context implementation of disk scheduling can take advantage of multiple cores in the microprocessor, which are present in almost all current generation microprocessors. The conventional algorithms do not take advantage of presence of multiple cores in microprocessor to make scheduling decision. This is also true for real time operation system where time plays

crucial role. In real time operating system small delays can also be undesirable. In such a situation an efficient disk scheduling algorithm can improve overall performance and reliability of the system.

Authors of this paper are doing work in this direction. Several popular disk scheduling algorithms have been studied for this purpose. A disk scheduling algorithm used in real time operating systems called "dead line scheduling algorithm" has been taken under consideration. Rest of the paper is given as following. In Related work section authors give the study that has been done on existing conventional algorithms. Proposed work section first describes working of a conventional disk scheduling algorithm called "Deadline scheduler" or "Deadline Scheduling Algorithm". After that proposed method has been described that is called "Parallel Deadline Scheduling Algorithm". The Experiments and Results section explores performance of implemented programs on different types of systems. This is denoted by the means of graphs. Discussion and Conclusion section ends the discussion by denoting improvements given by proposed method.

Materials and methods

Many conventional disk scheduling algorithms exists like SSTF [23], FCFS [23], N step SCAN [24], continum V(R) [25], SCAN-EDF [27]. Many other have been proposed by various researches. Some of them have been studied here.

Noop Scheduler: The name of the Noop scheduler itself indicates that it does simple least operations on the Input Output request before forwarding it to the primary physical device [1].It considers that either the order of request will be revised or primary physical device is accessing randomly [1].This scheduler is suitable for solid state device as these devices supports random access [1].One cannot say that Noop is essentially the ideal input output scheduler for the given scenario in previous paragraph [5].Because if we take whichever performance based regulation, each and every direction will depend on work load of schedulers [5].There is a chance that other schedulers will produce improved performance, if there is a conflict for existing input output bandwidth from different devices [5].

Deadline Scheduler: Deadline scheduler modifies/replaces the unordered FIFO queue of Noop scheduler to sorted queue for minimizing seeks times [1]. It places assigns deadlines for each request to guarantee a start service time for request. Deadline queues are basically sorted by their deadline, while the sorted queues are sorted by the sector number [6]. The deadline scheduler achieves this with the help of four queues. These four queues include one deadline queue and one sector sorted queue for both reads and writes [1]. While requesting the service, the scheduler first checks the deadlines of requests and it will identify whether any requests have increased beyond their time limit [1]. If any have, they are serviced immediately. If not, the scheduler services the following request in the sector sorted queue. [1].It yields a satisfying performance over the noop scheduler because it uses the sorted queues to lessen the seek times in disk. This scheduler also prevents the requests from going into starvation as it guarantees a start service time [1].

Anticipatory Scheduler: Anticipatory scheduler is the modification of deadline scheduler. Deceptive idleness is a trouble in the Deadline scheduler. Deceptive idleness is a state in which a process will finish accessing and reading from the disk in parallel of when it is in fact processing data in searching of the subsequent read action [2]. The scheduler selects the immediate request quickly. This situation causes deceptive idleness, a situation in which scheduler wrongly guesses that the process which is issued by final request has no additional requests, and it will be imposed to shift to a request from another process. This situation is disadvantageous to the throughput of reads which are occurred at the same time because it falls into a workload for seeking. [10].Deceptive idleness can be avoided by anticipatory scheduler by pausing for a tiny amount of time after a read process in the hope of another closely related request [3]. In [10] authors come up with the method to solve this dilemma based on the non-work-conserving scheduling regulation.

CFQ (Completely Fair Queuing): There are numerous analysis and research on quality and service management of disk input and output [17, 18, 19, 20] by fair queuing, which demand to accomplish the goal of performance virtualization. The Completely Fair Queuing Scheduler is slackly depend on facts of Random Fair Queuing [12,14]. Different queues are associated with different applications. This scheduler inserts requests in the queue from various kinds of applications. By round-robin principle it removes requests and dispatches from a busy queue. After that it will apply shifting in cyclic way on the list of busy queues. In former work of [15], authors have concluded that size of the requests and lagging of head positions correlated with requests are overlooked by round robin algorithm, and merely deals with the number of requests. This ends up in unfair queuing and it is unsuccessful to produce performance separation [4]. In the round-robin algorithm framework, performance of other application with different types of various seeks and different size of requests will be affected by applications which are heavy loaded with comparatively big size requests and applications with stochastic seeks [4]. To purge the effect of big request size on performance, in [16] authors have devised Time Sliced CFQ in disk scheduling. However, well-built fairness is not generated using TCFQ [4].In [26] authors have proposed YFQ. It gives assurance of accessing file with little wait and good fairness. It also promises tremendously increasing service.

NStep-SCAN Scheduler: The goal of N-step SCAN is to reduce the variation of respective waiting times when throughput is not degraded drastically [21]. This scheduler divides the request queue into sub queues of length N [7].Sub queues are processed using SCAN inside each sub queue [8]. It gives promise of servicing requests possible by dividing the queue into sections of N [7].As sub queues are previously occupied fully by elevator algorithm, this scheduler will be unable to push the request queue into sub queues of N size [7]. Thus starvation is avoided and gives promising behavior of servicing N requests [7].As an analogy we can say that there is a buffer of N requests. It services all the requests in any particular traversal. If any request comes for service it will have to wait in the separate queue. After completion of servicing all the requests which are in buffer, scheduler selects the next N processes which are waiting to be served. In short in this scheduler disk arm moves backward and forward same as it moves in SCAN, however all requests that are

coming in the way of movement of disk arm in one direction are stored. It will reorganize all stored requests for serving optimally while returning movement of head [22].

FSCAN Scheduler Algorithm: In [28] Raymond M.K.Chang and Donald have introduced fixed period scan. For hard real time system, FSCAN uses two queues. During scanning all the requests are put in one queue. While this queue is being processed, all other arriving requests will be kept in another queue which is called as backup queue. Once all the requests from the original queue have been serviced, it becomes the backup queue and now the old backup queue is serviced [8].This method prevents from arm stickiness. In other algorithms arm stickiness is occurred when series of requests for the identical sectors. It does not allow the disk arm to go ahead in that direction of the track [9].

Budget Fair Queuing (BFQ): the synchronous proportional shared time stamp based disk scheduler. Usually important in applications such as file copy or transfer, web, DBMS (Data Base Management System), or video streaming that required the synchronous disk requests [28]. In synchronous disk request there may be possibility that the work-conserving scheduler may fail to provide high throughput. To overcome this problem the author has presented a new approach for the synchronous proportional shared time stamp based disk scheduler called Budget Fair Queuing (BFQ), which is proposed for high number of requests passing through the system of Disk Scheduling with Fair frequency range allocation [28]. BFQ allows the disk idling, combine with proper back shifting of request time stamp based disk schedulers to present both guaranty and high throughput [28]. In BFQ it always guarantees that the over any time interval and independently of synchronous request a restricted delay with regarding to its tiny uncommunicated proportion of total number of bytes transferred by the disk device [28]. BFQ is implemented in Linux kernel with real and imitation mainstream applications.

The power consumption: of system can also be depending on the disk scheduling technique [29]. In real time disk scheduling algorithm to reduce the file fragmentation large data blocks are used [29]. In disk scheduling techniques the Input/output scheduler called as traffic mixer, which will dynamically schedule process among available, according to deadlines and service time estimation. According to evaluation result file system endeavor in power consumption reduction is between 9% to 21% disk powers consumption is leading to a reduction [29]. The proposed method in [29] shows that the in disk utilization important reductions can be reduce the disk power consumption up to 20% with 4MB of data buffer per video stream in the real time system.

Proposed method

Related work section mentions several disk scheduling algorithms used in various operating systems. All the algorithms try to minimize number of seeks to reduce total seek time. Different algorithms do this task by taking one or more other criterions in consideration along with seek time. Which algorithm is to be used depends on the requirement of operating system. In real time operating system along with minimizing

the number of seeks, "dead line" assigned to each request must also be satisfied. This dead line denotes the time before the request should be satisfied in order to continue proper execution. "Dead line scheduling algorithm" is used for this purpose in real time operating systems.

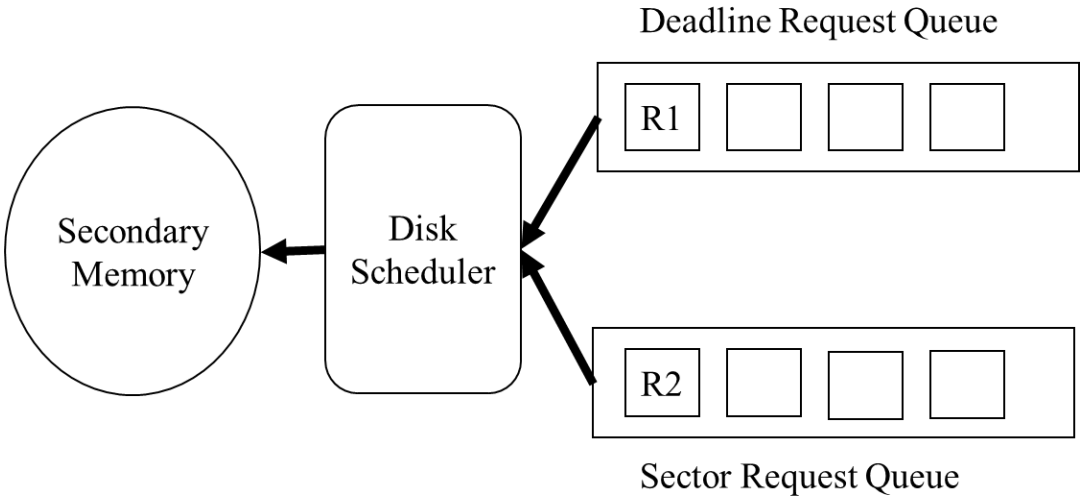

Figure 1 Deadline Disk Scheduling.

The method proposed here is called "Parallel Deadline Scheduling Algorithm". This algorithm ensures that each request will be satisfied before the deadline of the request, at the same time algorithm tries to minimize the total number of seeks. This method is similar to the deadline scheduling algorithm but performs some of its tasks in parallel on a multicore processor. In proposed method requests are stored in 2 different queues. One queue is used for managing deadlines and second one is used for reducing the number of seeks whenever possible. The first one is denoted as QUEUE1 and the second one is denoted as QUEUE2. In this method QUEUE1 is sorted in order of their deadlines. The QUEUE2 is schedule in order of sector numbers that is going to be executed. When schedule needs to select next request that is going to be satisfied, this method takes the first request in the QUEUE1 (QUEUE1 is sorted based on the values of deadline). That is request with the minimum deadline value. If deadline of this method is about to expire in near future then this request is granted by the schedule to be satisfied next on the disk. This request is then removed from both queues. If this request's deadline is not going to expire in near future, the scheduler takes first process from the sorted QUEUE2 to be satisfied next on the disk. That is the process having requested sector number closest to the previous requested sector number. Thus the proposed method ensures required service time of a request that is essential to the real time operating system and also maintains lower seeks of read-write head on a disk.

ParallelDeadlineScheduling()

1. read values in QUEUE1 and QUEUE2
2. in parallel
    1. parallel sort QUEUE1 based on Dead Lines
    2. parallel sort QUEUE2 based on Sector Numbers
3. while (QUEUE1 is not empty or QUEUE2 is not empty)
    1. NEXT = First Entry of QUEUE1
    2. if(NEXT is about to expire)
        1. select NEXT as request to be satisfied by scheduler.
        2. remove first entry of QUEUE1.
        3. parallel search NEXT is QUEUE2 and remove.
    3. Else
        1. NEXT = parallel search request in QUEUE2 such that its sector number is nearest to PREVIOUS request sector number.
        2. select NEXT as request to be satisfied by scheduler.
        3. remove NEXT from QUEUE1.
        4. parallel search NEXT is QUEUE2 and remove.

The internal tasks of proposed methods are performed in parallel unlike conventional algorithm. In a multi core architecture system since there is multiple cores the operating system is able to perform more than one task at the same time i.e. in parallel. There for load is distributed over multiple cores. Based on this an algorithm is proposed in this paper which is mentioned in Algorithm 1. In first step algorithm initialize various configuration parameters. These parameters generally depend on type of the system and application. Parameters here are given values based on type and requirement of operating system. One of the important parameter to be configured is timeQuantom. This parameter is given estimated value that is equivalent or greater than the time required to server "one disk request". In second step algorithm takes input from the system in the form of disk scheduling requests. This request may be submitted to the operating system by various programs running on operating system. Each request has 3 important parameters, REQUEST_ID,

sectorRequest, deadline. REQUEST_ID is number that uniquely identifies each request. It is maintained by operating system. requestedSector is number of sector on the disk that is needed to be accessed by the request. Deadline parameter holds deadline time stamp of the request. It is required that the request should be satisfied before this deadline time value. These parameters of all current requests are copied into 2 different queues QUEUE1 and QUEUE2. In third step it is required that both queues QUEUE1 and QUEUE2 should be shorted. QUEUE1 is shorted by deadline parameter and QUEUE2 is sorted based on value of requestedSector parameter. This step performs sorting on QUEUE1 and QUEUE2 in parallel. That improves the performance of the algorithm. Fourth step is executed until there is a request in any of the queue. In each step algorithm determines which request is to be granted next. This is done by taking first request in QUEUE1 and checking if deadline of this process will expire in next time quantum, if yes then that process is selected by scheduler and granted for disk access if not then first request from QUEUE2 is selected and granted for disk access. After the request is granted it is need to be removed from both queues. This is done in parallel for QUEUE1 and QUEUE2. SearchAndRemoveInQUEUE1 and SearchAndRemoveInQUEUE2 procedures are also parallel procedures. They perform the task of searching the request in given queue by dividing entire queue in smaller chunks of requests. These chunks are assigned to the cores available dynamically. Whenever a core completes its current chunk next chunk is assigned to it. This greatly improves the performance of the algorithm when compared to the serial algorithm.

The proposed algorithm can be implemented by several parallel programing techniques. This algorithm has been implemented by authors using OpenMP libraries in Microsoft Visual Studio 2010.

Results and discussion

We conduct a series of experiments on different types of machine for results evaluations and analysis. For performance evaluation different CPU have been taken under consideration. For evaluating performance on 2 core machine Intel Core2duo 3.2 GHz processor running windows 7 with 2 GByte of RAM has been taken under consideration, for 4 core machine Intel Corei5 2.4 GHz processor running windows 7 with 4 GByte of RAM has been taken and for 8 core machine Intel Corei7 2.8 GHz processor running windows 7 with 4 GByte of RAM has been considered.

Result Analysis on 2 Core Machine

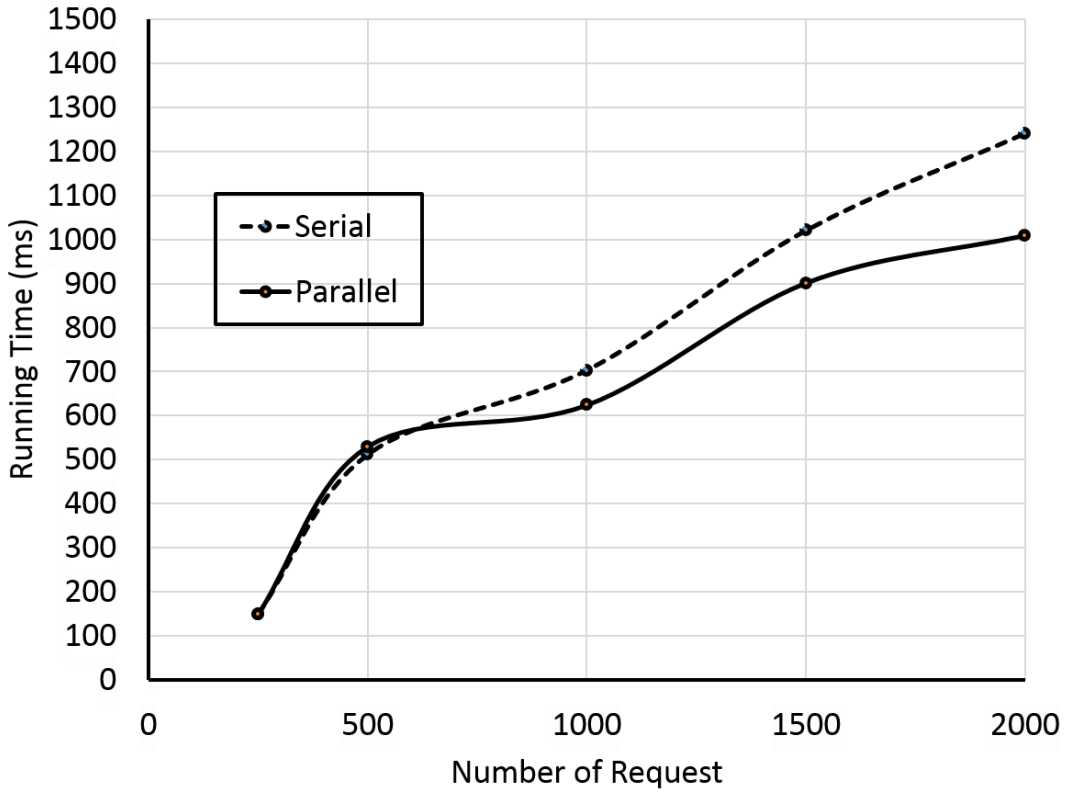

Figure 2 An Example of serial vs parallel on 2 core machine.

Figure 2 shows the comparison between serial deadline scheduling algorithm and parallel deadline scheduling algorithm in 2 Core machine. X-axis shows the number of I/O requests and Y-axis represents the running time of deadline scheduling algorithm in milliseconds. For small inputs (request) deadline scheduling algorithms takes same time, when after one point parallel algorithm take less time than serial algorithm. Gradually we know that parallel algorithm takes less time than serial algorithm as number of inputs increases.

Result Analysis on 4 core Machine

In figure 3 we showed the comparison between serial deadline scheduling algorithm and parallel deadline scheduling algorithm but this time we have tested it on 4 core machine. X-axis represents the number of I/O requests and Y-axis represents the running time of algorithms in milliseconds. Unlike 2 Core graph (in figure 1) parallel algorithm works faster even for smaller number of I/O requests in 4Cores and gradually parallel deadline algorithm works more faster than serial with the increasing

I/O requests. We can see that the end points of both serial and parallel algorithms go far from each other.

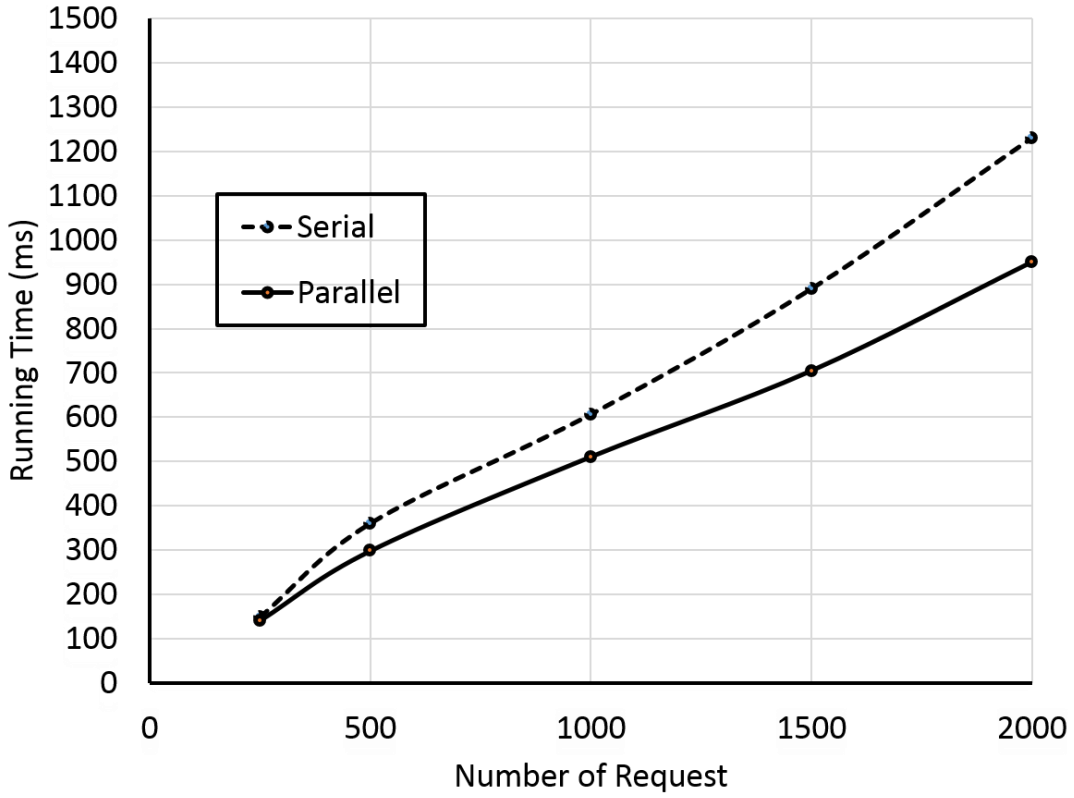

Figure 3 An Example of serial vs parallel on 4 Core Machine.

Result Analysis on 8 Core Machine

Figure 4 shows algorithm testing on 8 Cores. We can see that it gives the improvement in the execution time over 4 Cores.

In Table I. gives experimental values with execution time (ms) for different number of core, for different number of input (requests queue size). As in the Table I. It can be seen that when input size (request queue size) is increased than the execution time of parallel program is also increases, but when compared with serial program execution time is less in case of parallel program running on 2 core or 4 cores or 8 cores.

Table 1 Experimental Data Table.

Request Queue Size	**Running Time (ms)**		
	2 Core	***4 Core***	***8 Core***
100	150	142	136
500	530	299	262
1000	624	511	408
1500	901	705	596
2000	1009	950	734

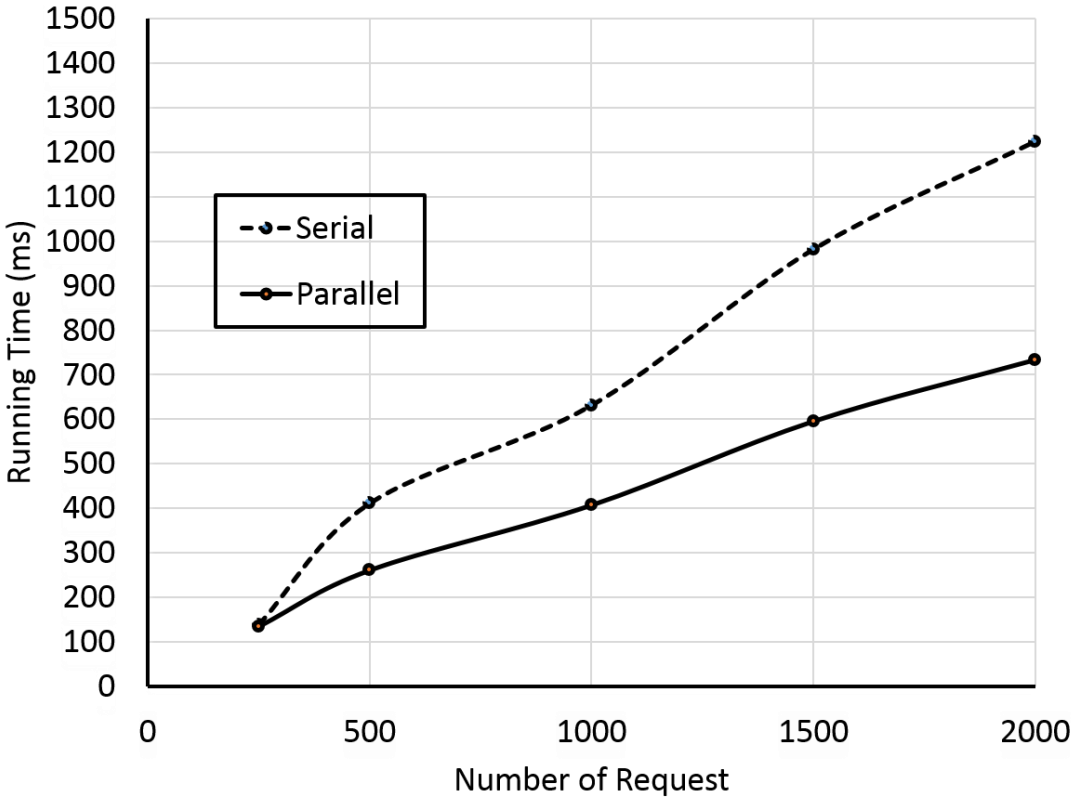

Figure 4 Example of serial vs parallel on 8 Core Machine.

Comparisons between 2,4,8 Core Machine

We have depicted the parallel algorithm execution time analysis for all 2 Cores, 4 Cores and 8 Cores in figure 5.We analyzed that on 8 Cores our parallel algorithm works more faster than 2 Cores and 4 Cores. The number of input (request) increases for parallel deadline scheduling algorithm it will execute in less time.

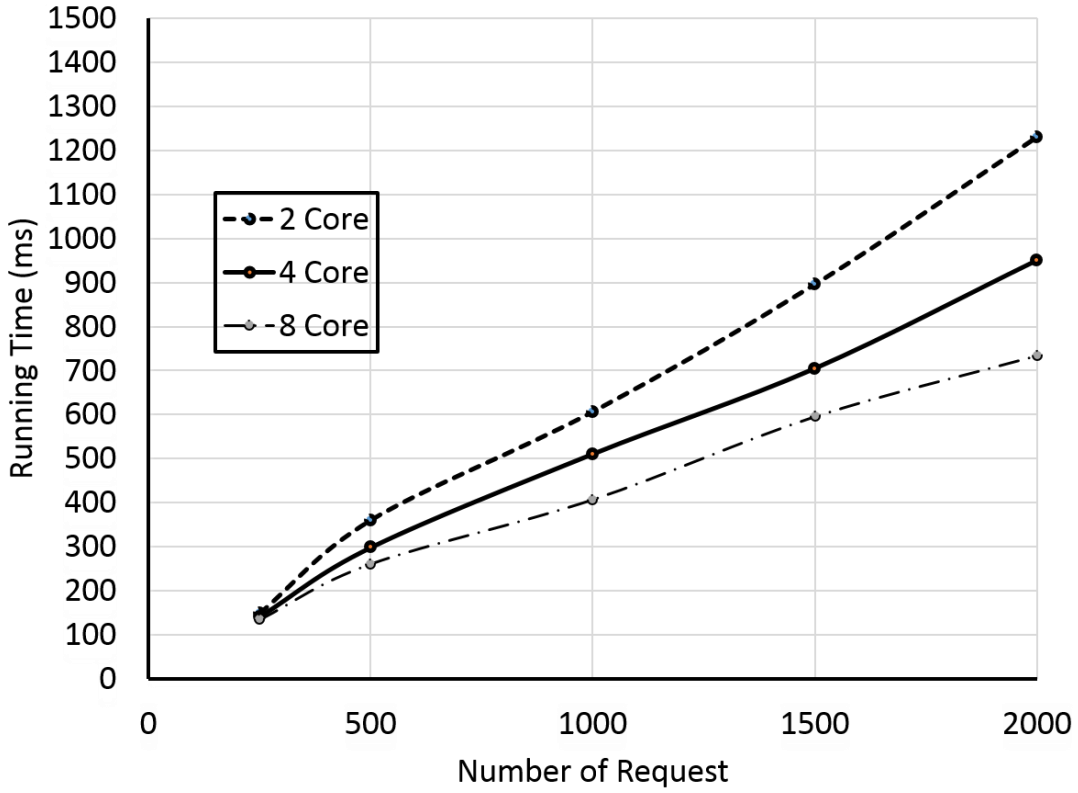

Figure 5 Parallel algorithm compare on 2, 4, and 8 Core.

Conclusion

In this paper various conventional disk scheduling algorithms have been studied out of which deadline scheduling algorithm has been taken into consideration. Deadline scheduling algorithm has been studied in depth in this paper and a similar parallel algorithm has been proposed. This parallel deadline scheduling algorithm takes advantage of having multiple cores in a system. This algorithm performs internal tasks in parallel. The proposed algorithm has been taken into consideration for various type and size of input of scheduling request. Performance analysis denotes that the proposed method has significant improvement over normal algorithm. This has been shown by means of graphs. Since scheduling algorithm is one of the major factor on which performance of disk depends it can be said that performance of disk will be increased by use of proposed method. Hence proposed method provides improvement in overall performance of a system.

References

[1] Marcus Dunn and A. L. Narasimha Reddy.”A New I/O Scheduler for Solid State Devices.”

[2] Iyer, Sitaram. “The Effect of Deceptive Idleness on Disk Schedulers”. Retrieved 2010-04-20.

[3] Morton,Andrew (2003-01-23). "2.5.59-mm5". “linux-kernel,linux-mmmailing list.” Archived from the original on 15 June 2007. Retrieved 2007-05-23.

[4] Seetharami R. Seelam, P. Teller. “Virtual I/O Scheduler: A Scheduler of Schedulers for Performance Virtualization”

[5] Wikipedia contributors. “Noop scheduler”. *Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Noop_scheduler&oldid=565633280. 24 July 2013 16:26 UTC

[6] Wikipedia contributors. “Deadline scheduler”. *Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Deadline_scheduler&oldid=563543820. 9 July 2013 16:40 UTC

[7] Wikipedia contributors. “N-Step-SCAN”. *Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=N-Step-SCAN&oldid=553727943. 6 May 2013 04:16 UTC

[8] http://logos.cs.uic.edu/385/notes/Disk%20Scheduling%20Algorithms.htm

[9] Wikipedia contributors. “FSCAN”. *Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=FSCAN&oldid=549225593. 7 April 2013 22:14 UTC

[10] Sitaram Iyer, Peter Druschel.“Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O, 18th ACM Symposium on Operating Systems Principles (SOSP 2001)

[11] Seetharami R. Seelam, Patricia J. Teller.Virtual I/O Scheduler: A Scheduler of Schedulers forPerformance Virtualization.2007 ACM 978-1-59593-630-1/07/0006

[12] J. Axboe, “Linux Block IO - Present and Future,” Proceedings of the Ottawa Linux Symposium 2004, Ottawa, Ontario, Canada, July 21-24,2004, pp. 51–61.

[13] J. Axboe, “Time Sliced CFQ I/O Scheduler,” http://kerneltrap.org/node/4406.

[14] P. McKenney, “Stochastic Fairness Queueing,” Internetworking:Research and Experience, January 1991, Vol. 2, pp. 113–131.

[15] S. Seelam and P. Teller, “Fairness and Performance Isolation: an Analysis of Disk Scheduling Algorithms,” Proceedings of International Workshop on High Performance I/O Techniques and Deployment of Very Large Scale I/O Systems (HiperIO’06),Barcelona, Spain, September 25-28, 2006.

[16] J. Axboe, “Time Sliced CFQ I/O Scheduler,” http://kerneltrap.org/node/4406.

[17] W. Jin, J. Chase, and J. Kaur, “Interposed Proportional Sharing for a Storage Service Utility,” Proceedings of the ACM Sigmetrics -Performance (SIGMETRICS’04), New York, June 2004

[18] C. Lumb, A. Merchant, and G. Alvarez, “Facade: Virtual Storage Devices with Performance Guarantees,” Proceedings of the Conference on File and Storage Technology (FAST’03), San Francisco, CA,March 31-April 2, 2003, pp. 131–144.

[19] R. Wijayarante and A. L. N. Reddy, “Providing QOS Gurantees for Disk I/O,” Multimedia Systems, January 2000, 8(1):57–68.

[20] J. Zhang, A. Riska, A. Sivasubramaniam, Q. Wang, E. Riedel, "Storage Performance Virtualization via Throughput and Latency Control," Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS), September 2005.

[21] Toby J. Teorey and Tad B. Pinkerton. " A Comparative Analysis of Disk Scheduling Policies." Communications of the ACM March 1972.Volume15,Number3

[22] Toby J. Teory."Properties of disk scheduling policies in multiprogrammed computer systems".

[23] P. J. Denning,Effects of scheduling on file memory operations, Proc. AFIPS, Montvale, N.J., (1967), pp. 9–21.

[24] H. Frank,Analysis and optimization of disk storage devices for time sharing systems, J. ACM 16, No. 4 (1969), pp. 602–620.

[25] R. Geist and S. Daniel,A continuum of disk scheduling algorithms, ACM Trans. Computer Syst. 5, No. 1 (1987), pp. 77–92.

[26] John Bruno, Jos6 Brustoloni, Eran Gabber, Banu Ozden and Abraham Silberschatz. "Disk Scheduling with Quality of Service Guarantees Bell Laboratories, Lucent Technologies. 0-7695-0253-9/9 1999 IEEE

[27] A. L. Narasimha Reddy,Jim Wyllie."Disk scheduling in a multimedia I/O system"

[28] Paolo Valente and Fabio Checconi "High Throughput Disk Scheduling with Fair Bandwidth Distribution" IEEE TRANSACTIONS ON COMPUTERS, VOL. 59, NO. 9, SEPTEMBER 2010.

[29] Damien Le Moal, Donald Molaro and Jorge Campello "Power Efficient Real-Time Disk Scheduling" NOSSDAV'09, June 3–5, 2009, Williamsburg, Virginia, USA. Copyright 2009 ACM 978-1-60558-433-1/09/06