A Novel Disk Scheduling Algorithm in Real-time Database Systems

S.Y.Amdani
B.N. College of Engg.,
Pusad. Maharashtra, India

G.R.Bamnote
PRM Institute of Technology & Research, Badnera-Amravati, Maharashtra, India

H.R.Deshmukh
B.N.College of Engg.,
Pusad, Maharashtra, India

S.A.Bhura
B.N.College of Engg.,
Pusad, Maharashtra, India

ABSTRACT

Conventional databases are mainly characterized by their strict data consistency requirements. Database systems for real-time applications must satisfy timing constraints associated with transactions. In this paper a novel disk scheduling algorithm for real-time database system is proposed. The main objective of this paper is to initiate an enquiry in Disk scheduling for real time database systems. The proposed work aims at the investigation of efficient disk scheduling techniques in real time databases. After investigation it was found that our proposed approach gives better performance than the existing algorithms.

Categories and Subject Descriptors
H.2.4 [Database Management]: Systems, Real-time,I.6.6 [Simulation output Analysis]

General Terms
Algorithms, Performance, Experimentation

Keywords
Real-time Database Systems, Disk Scheduling Algorithms.

1. INTRODUCTION

In the information age, information spreading worldwide through Internet, and other medium, is bulk and changing constantly and dynamic in nature. As our society becomes more integrated with computer technology, information processed for human activities necessitates computing that responds to requests in real-time rather than just with best effort. Real time data base systems combine the concepts from real time systems and conventional database systems. Real time

systems are mainly characterized by their strict timing constraints. Conventional databases are

mainly characterized by their strict data consistency requirements. Thus, real time database systems should satisfy both the timing constraints with data integrity and consistency constraints. Real-time Database Systems(RTDBS) have immerged as an alternative to manage the data with a structured and systematic approach. RTDBS have different performance goals, correctness criteria, and assumptions about the applications. The conventional database system's main objective is to provide fast response time, whereas a RTDBS may be evaluated based on how often

transactions miss their deadlines, the average "tardiness" of late transactions, the cost incurred in transactions missing their deadlines, data external consistency and data temporal consistency. The main criteria in assessing the success of any scheduling policy is the success ratio i.e the number of transactions completed successfully before their deadline. In this paper we have discussed the disk scheduling algorithms. We have simulated over some of the real time disk scheduling algorithms and compared their performance. The organization of this paper is as follows: section 2 gives an overview of disk scheduling problem. In section 3 conventional and real time disk scheduling algorithms are discussed. In section 4 experimental results and performance evaluation of real time disk scheduling algorithms is presented. Finally section 6 concludes with a summery.

2.DISK SCHEDULING PROBLEM

In a disk-based database system, disk I/O occupies a major portion of transaction execution time. As with CPU scheduling, disk scheduling algorithms that take into account timing constraints can significantly improve the real-time performance. CPU scheduling algorithms, like Earliest Deadline First and Highest Priority First, are attractive candidates but have to be modified before they can be applied to I/O scheduling. The main reason is that disk seek time, which accounts for a very significant fraction of disk access latency, depends on the disk head movement. The order in which I/O requests are serviced, therefore, has an immense impact on the response time and throughput of the I/O subsystem.

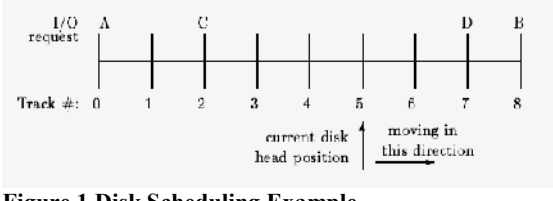

Figure 1 Disk Scheduling Example

The Elevator Algorithm, for example, moves the head from one end of the disk to the other and then back, servicing whatever requests are on its way, and changing direction whenever there are no more requests ahead in its direction. Referring to the example in Figure 1, the Elevator Algorithm will produce the following servicing schedule:

Elevator: D; B; C; A

The problem with the Elevator Algorithm, as applied to real-time systems, is that the priority of requests is not considered[19].

To service a disk request, several operations take place. First, the disk head must be moved to the appropriate cylinder (seek time). Then, the portion of the disk on which the disk page is stored must be rotated until it is immediately under the disk head (latency time). Then, the disk page must be made to spin by the disk head (transmission time). The above components needed to service a disk request are illustrated in Figure 2

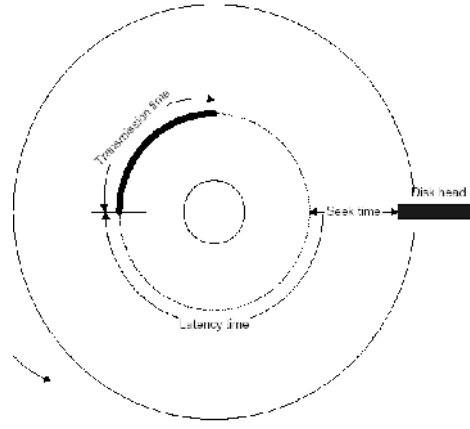

Figure 2 Components of a disk access

Queues build up for each disk because the inter-arrival time of the disk requests can be smaller than the time required by the disk to service a disk request. Disk scheduling involves a careful examination of the pending disk requests to determine the most efficient way to service the disk requests. The disk scheduling problem

involves reordering the disk requests in the disk queue so that the disk requests will be serviced with the minimum mechanical motion by employing seek optimization and latency optimization.
The following goal should be considered in scheduling a real-time system[11].

1. Meeting the timing constraints of the system
2. Preventing simultaneous access to shared resources and devices.
3. Attaining a high degree of utilization while satisfying the timing constraints of the system.
4. Reducing the communication cost in real-time systems.

Basically, the scheduling problem is to determine a schedule for the execution of the jobs so that they all completed before the overall deadline.

3. DISK SCHEDULING ALGORITHMS

3.1 Classical Disk Scheduling Algorithms

The following classical scheduling algorithms described below are well known[4].
FCFS: This is the simplest strategy in which each request is served in first-come-first-serve basis[4].

SCAN: This is also known as the elevator algorithm in which the arm moves in one direction and serves all the request in that direction until there are no further request in that direction[4].

C-SCAN: The circular SCAN algorithm works in the same way as SCAN except that it always scans in one direction. After serving the last request in the scan direction, the arm return to the start position[4].

SSTF: The SSTF, for shortest seek time first, algorithm simply selects the request closest to the current arm position for service[4].
A common feature of all these classical scheduling algorithms is that none of them takes the time constraint of request into account. This results in poor performance of classical algorithms in real-time systems.

3.2 EXISTING REAL-TIME DISK SCHEDULING ALGORITHMS

The real-time disk scheduling algorithms like Earliest Deadline First (EDF), Priority Scan (P-Scan), Feasible Deadline Scan (FD-Scan), Shortest Seek and Earliest Deadline by Ordering (SSEDO) and Shortest Seek and Earliest Deadline by Value (SSEDV) are discussed in this seminar.

EDF (Earliest Deadline First): Requests are ordered according to deadline and the request with the earliest deadline is serviced first. Assigning priorities to transactions an Earliest Deadline policy minimizes the number of late transactions in systems operating under low or moderate levels of resource and data contention. This is due to Earliest Deadline giving the highest priority to transactions that have the least remaining time in which to complete. However, the performance of Earliest Deadline steeply degrades in an overloaded system[14]. This is because, under heavy loading, transactions gain high priority only when they are close to their deadlines. Gaining high priority at this late stage may not leave sufficient time for transactions to complete before their deadlines. Under heavy loads, then, a fundamental weakness of the Earliest Deadline priority policy is that it assigns the highest priority to transactions that are close to missing their deadlines, thus delaying other transactions that might still be able to meet their deadlines[1].

P-SCAN: In Priority Scan (P-Scan) all request in the I/O queue are divided into multiple priority levels. The Scan algorithm is used within each level, which means that the disk serves any requests that is passes in the current served priority level until there are no more requests in that direction. On the completion of each disk service, the scheduler checks to see whether a disk request of a higher priority is waiting for service[15]. If found, the scheduler switches to that higher level. In this case, the request with shortest seek distance from the current arm position is used to determine the scan direction.

All the I/O requests are mapped into three priority levels according to their deadline information. Specially, transactions relative deadlines are uniformly distributed between LOW_DL and UP_DL, where LOW_DL and UP_DL are lower and upper bounds for transaction deadline settings. If a transactions relative deadline is greater than (LOW_DL + UP_DL)/2, then it is assigned the lowest priority. If the relative deadline is less than (LOW_DL + UP_DL)/4, then the transaction receives the highest priority. Otherwise the transaction is assigned a middle priority[4].

FD-SCAN (Feasible Deadline Scan) : In FD-Scan, the track location of the request with earliest feasible deadline is used to determine the scan direction. A deadline is feasible if we estimate that it can be met. More specifically, a request that is 'n' tracks away from the current head position has a feasible deadline 'd' if d >= t + Access(n) where 't' is the current time and Access(n) is a function that yields the expected time needed to service a request 'n' tracks away. Each time that a scheduling decision is made, the read requests are examined to determine which have feasible deadlines given the current head position. The request with the earliest feasible deadline is the target and determines the scanning direction. The head scans toward the target servicing read requests along the way. These requests either have deadlines later than the target request or have unfeasible deadlines, ones that cannot be met. If there is no read request with a feasible deadline, then FD-SCAN simply services the closest read request. Since all request deadlines have been (or will be) missed, the order of service is no longer important for meeting deadlines[15].

SSEDO (Shortest Seek and Earliest Deadline by Ordering): The idea behind the above algorithm is that we want to give requests with smaller deadlines higher priorities so that they can receive service earlier. This can be accomplished by assigning smaller values to their weights. On the other hand, when a request with large deadline is "very" close to the current arm position (which means less service time), it should get higher priority. This is especially true when a request is to access the cylinder where the arm is currently positioned. Since there is no seek time in this case and we are assuming the seek time dominates the service time, the service time can be ignored. Therefore these requests should be given the highest priority.

SSEDV (Shortest Seek and Earliest Deadline by Value): In SSEDO algorithm, the scheduler uses only the ordering information of request deadlines. The SSEDV uses the differences between deadlines of successive requests in the window i.e. choose the request with minimum value for service (remaining lifetime of request i.e. length of time between current time and request deadline). A common characteristic of SSEDV and SSEDO algorithm is that both consider time constraints and disk service times. Which part play the greater role in decision making can be adjusted by tuning the scheduling parameters, depending on the algorithm[4][20].

4. THE PROPOSED ALGORITHM

In our proposed algorithm all the request in the I/O queue are divided into multiple priority levels. Transactions are assigned the priority depending on the deadlines. All the I/O requests are mapped into three priority levels according to their deadline information. Specially, transactions relative deadlines are uniformly distributed between LOW_DL and UP_DL, where LOW_DL and UP_DL are lower and upper bounds for transaction deadline settings. If a transactions relative deadline is greater than (LOW_DL + UP_DL)/2, then it is assigned the lowest priority. If the relative deadline is less than (LOW_DL + UP_DL)/4, then the transaction receives the highest priority. Otherwise the transaction is assigned a middle priority.

The transaction selects the transaction with minimum deadline from the high level and also serves the transactions that are close to the current head position and then serve the transaction with next minimum deadline. Thus, the requests with smaller deadlines can receive service earlier and also when a requests with large deadline is very close to the current arm position are also served which will reduce the arm moment. Same procedure is repeated for all

the transactions in the three priority levels. The important steps in scheduling algorithm are:

- Construct three queues to store the transaction with minimum, middle or maximum priorities.
- Set start_time, end_time, seek_time, current_head_position, total_transaction_time, turn_around_time for the transactions with minimum deadline in the three queues.
- Find transactions with seek time within threshold.
- Check transaction is miss or hit in all the queues.

5. EXPERIMENTAL RESULTS

We have investigated and implemented all the above real-time disk scheduling algorithms namely EDF, P-SCAN, FD-SCAN, SSEDO, SSEDV. In these algorithms, preferential treatment is given to transactions, which are very critical, and with stringent timing constraints. Hence deadline is calculated on the basis of transaction execution time and slack time. We have also compared the performance of these algorithms under same workload condition. For the implementation of above-mentioned algorithms first we have formulated the disk-scheduling problem for real-time database systems and then implemented the mathematical model for all the algorithms. To get the evaluation parameters values, he have simulated the mathematical model for the number of times. The experimental results show that the performance of SSEDO and SSEDV is better than EDF, FD-SCAN and P-SCAN in heavy workload.

5.1 Performance of Various Disk Scheduling Algorithms

We have experimented the transaction loss probability of all the existing algorithms plus the proposed algorithm under different workloads. With random arrival fashion of the transaction, with arrival rate 0.15, number of transactions 100 and disk size in block 100, the comparison given here is based on the properties like total transactions, successful transaction, time spend an all transaction, time spend on successful transaction, utilization of system and success ratio.

Table 1. Performance of Algorithms for Random Arrial.

Properties	EDF	FD Scan	P Scan	SSE DO	SSE DV	Prop osed
Total Transaction	100	100	100	100	100	100
Successful Transaction	31	46	46	73	74	76
Time Spend on all Transactions	2073	795	799	756	781	790
Time Spend on Successful Transactions	574	389	407	357	626	610
Utlization of System	0.28	0.49	0.51	0.66	0.8	0.77
Success Ratio	0..31	0.46	0.46	0.73	0.74	0.76

Table 2. Performance of Algorithms for 10 Runs

Run#	EDF	FD_ SCAN	P_ SCAN	SSE DO	SSE DV	Propo sed
	100	100	100	100	100	100
1	34	47	47	73	76	79
2	28	45	45	71	74	77
3	21	40	40	68	69	70
4	35	48	48	75	77	79
5	38	50	50	77	77	78
6	34	47	47	74	73	74
7	29	47	47	73	74	76
8	25	43	43	71	72	72
9	32	46	46	76	76	77
10	30	45	45	74	74	76

As shown in table 1, performance of SSEDV is better than SSEDO, since SSEDV uses more timing information than the SSEDO for decision making. P-SCAN and FD-SCAN perform essentially at the same level. The EDF algorithm is good when the system is lightly loaded, but it degenerates as soon as load increases, as shown in Table 2. Table 2 shows the performance of algorithms for 10 runs. Figure 2 shows the performance of algorithms with 10 runs with 100 transactions.

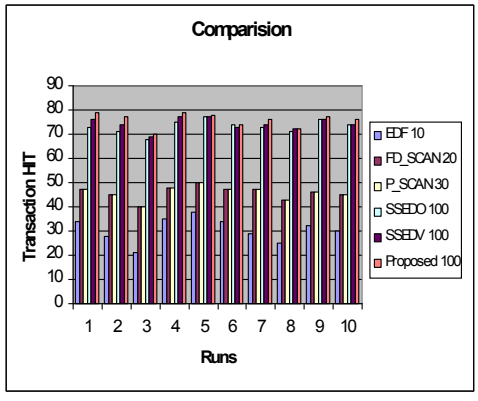

Figure 3. Performance of Algorithms at same load.

As shown in the Figure 3 the performance of our proposed algorithm is better than other real-time scheduling algorithms.

6. CONCLUSION

we investigated various real-time disk scheduling algorithms like EDF, P-SCAN, FD-SCAN, SSEDO and SSEDV. In EDF transactions are ordered according to deadline and the request with earliest deadline is serviced first. Priority-Scan divides all the request in the I/O queue the scan algorithm then serves any request that is passes in the current served priority level until there are no more request in that direction. In FD-SCAN, the track location of the request with earliest feasible deadline is used to determine the scan direction. In the SSEDO algorithm, the scheduler uses the ordering information of request deadlines, whereas SSEDV use the difference between deadlines of successive requests in the window.

The results of the comparison shows that, performance of SSEDV is better than SSEDO, since the SSEDV uses more timing information than the SSEDO for decision making. P-SCAN and FD-SCAN perform essentially at the same level, with one better at high load cases, but worse for low load cases. The EDF algorithm is good when the system is lightly loaded, but it degenerates as soon as load increases. When we implemented our proposed algorithm, which divides the transaction into three classes, then schedules the transactions considering the priority and the head position, it gave better results. As different algorithm show different results at various transaction load the further modification to this disk scheduling problem is real-time database system can be monitoring the I/O load dynamically, focusing on using analyses of disk accesses to determine the best disk scheduling algorithm for the current workload, and switching algorithms as necessary to improve performance.

7. ACKNOWLEGEMENTS

The authors would like to thank the Principal, BNCOE, Pusad (India) for encouragement and support.

8. REFERENCES

[1] R. Abbott and H. Garcia-Molina, “SchedulingReal-Time Transactions: A Performance Evaluation”, Proceedings of the 14th VLDB Conference, Los Angeles, California, March 1988.

[2] N. Audsley, A. Burns, “Real Time System Scheduling”, Technical Report No. YCS 134, Department of Computer Science, The University of York, UK, 1990.

[3] H. Chetto, M. Silly, and T. Bouchentouf, “Dynamic Scheduling of Real-Time Tasks under Precedence Constraints”, The Journal of Real-Time Systems, Vol. 2, pp. 181-194, 1990.

[4] Shenze Chen, John A. Stankovic, James Kurose and Don Towsley “Performance Evaluation of Two New Disk Scheduling Algorithms”, The Journal of Real-Time Systems, 1990

[5] S. Chen, J.A. Stankovic, J. F. Kurose, and D. Towsley, “Performance Evaluation of Two New Disk Scheduling algorithm for Real-Time Systems”, The Journal of Real-Time Systems, Vol. 3, pp. 307-336, 1991.

[6] J. R. Haritsa, M. Livny, and M. J. Carey, “Earliest Deadline Scheduling for Real-Time Database Systems”, Proceedings of the IEEE Real-Time Systems Symposium, pp. 232-242, 1991.

[7] S. Iyer. The Effect of Deceptive Idleness on Disk Schedulers. Master’s Thesis, Computer Science Department, Rice University. April 2001.

[8] D. Martens. Disk Access Analysis for Optimal Performance. Department of Computer Science, The University of Western Ontario. September 2005.

[9] P. S. Yu, K. Wu, K. Lin, and S. H. Son, “On Real-Time Databases: Concurrency Control and Scheduling”, Proceedings of the IEEE, Vol. 82, No. 1, pp. 140-156, January 1994.

[10] Systems Support for Preemptive Disk Scheduling - IEEE TRANSACTIONS ON COMPUTERS, VOL. 54, NO. 10, OCTOBER 2005

[11] Z. Dimitrijevic, R. Rangaswami, and E. Chang. Design, analysis, and implementation of Virtual IO. September 2002.

[12] A Real-Time Disk Scheduling Algorithm For Multimedia Storage Servers - A thesis submitted in partial fulfillment for the degree of Master of Science By Sameh Mohamed Ibrahim Elnikety, 1999

[13] Technical Report No. 2005-499 Scheduling Algorithms for Real-Time Systems Arezou Mohammadi and Selim G. Akl

[14] Lisa Cingiser DiPippo and Victor Fay Wolfe, "Real-Time Databases", Book chapter, September 23, 1995.

[15] Scheduling I/O Requests with Deadlines: a Performance Evaluation CH2933-0/90/0000/0113 1990 IEEE Robert K. Abbott Hector Garcia-Molina

[16] An Efficient Non-Preemptive Real-Time Scheduling - Wenming Li, Krishna Kavi and Robert Akl

[17] Sang H. Son , A Priority-Based Scheduling Algorithm for Real-Time Databases - Department of Computer Science University of Virginia Charlottesville, Virginia 22903, USA Seog Park Department of Computer Science Sogang University Seoul, Korea

[18] Audsley N. and Burns A.," Real-Time System Scheduling", Technical Report No. YCS 134, Department of Computer Science, The University of York, UK, 1990.

[19] Haritsa, J., Carey, M., Livny, M., "Earliest Deadline Scheduling for Real-Time Database Systems", Proceeding of the IEEE Real-Time Systems Syposium, pp. 232-242. 1991.

[20] Value-Based Scheduling in Real-Time Database. Systems. Jayant R. Haritsa, Michael J. Carey, and Miron Livny. Received May 15, 1991