# An Attempt to Implement the Algorithms for the Real-Time Database Disk Scheduling Applications

S. B. Thorat, Sudhir Jagtap, and Saleem Amdani

***Abstract*—In this age of information, the information is spreading widely worldwide through the Internet, and other medium. It is huge in size and the changes are constant as well as dynamic in nature. The society has became more integrated with the advents in computer technology since the information processed for numerous applications necessitates computing that responds to requests in real-time because of it overwhelming advantages. These days Real Time Database is right on & large number of users accesses the information which in turn degrades the system performance. This degradation may cause delay and trouble for particular end user in accessing the information. Hence accessing the information with easy, sophistication and within certain time limit being the real time application, by meeting the required timing constrains, assessing user's requirements and then providing them information in time are important aspects. The conventional databases are mainly characterized by their strict data consistency requirements. As against this, the database systems for real-time applications must satisfy timing constraints associated with every transaction. The authors of this Paper aim to initiate an enquiry in Disk scheduling for real time database systems & it's an attempt to implement the Algorithms for the Real-Time Database application. The proposed work implements the Algorithms for Disk scheduling for real time database systems.**

***Index Terms*—Earliest deadline first, priority scan, feasible deadline scan, shortest seek and earliest deadline by ordering, shortest seek and earliest deadline by value.**

## I. General Parameters consider for Algorithm Development

Deadline: Time by which execution of the task should be completed, after the task is being released.

Arrival Time: Arrival time of the transaction.

Total number of Transactions

Inter Arrival Time

Average Execution Time

Transaction Size

Slack Time: It is an estimate of how long we can delay the execution of transaction till we meet its deadline.

Access Time:

Access Time = Seek Time + Rotation Latency + Transfer Time

where, Seek Time is the time for the disk to move then to the cylinder containing the desired sector.

Rotation Latency is the time waiting for the disk to rotate the desired sector to the disk head.

Transfer Time is the transfer time needed to transfer data over the IO bus.

Priority: Priorities could be assigned by two ways-Earliest Deadline: The transaction with the earliest deadline has the highest priority. A major weakness of this policy is that it can assign the highest priority to a task that already has missed or is about to miss its deadline. its deadline. A negative slack time results either when a transaction have already missed its deadline or when we can estimate that it cannot meet its deadline. The slack time of a transaction which is not executing ise decreased & hence the priority of that transaction increases [1].

Manuscript received October 23, 2013; revised June 20, 2014.

S. B.Thorat is with the Institute of Technology and Management, Nanded-431 602, MS, India (e-mail: suryakant_thorat@yahoo.com).

Sudhir Jagtap is with the Swami Vivekanand College, Udgir, MS, India (e-mail: sudhir.jagtap7@gmail.com).

Saleem Amdani is with the B.N. College of Engineering, Pusad, Dist-Yavatmal, M.S., India (e-mail: salimamdani@yahoo.com).

## II. 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 nut shell here [1], [2].

## III. EDF Algorithm

The Earliest Deadline First algorithm is analogous to FCFS. In this algorithm, the requests are ordered according to their deadlines 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 happens due to Earliest Deadline policy of giving the highest priority to transactions that have the least remaining time to complete. However, the performance of Earliest Deadline steeply degrades in an overloaded system [2]. This degradation is obvious since under the heavy load conditions, transactions can gain high priority only when they are close to their deadlines. So gaining high priority at this late stage may not leave the sufficient time for transactions to get completed before their deadlines. Hence, for the heavy load, then, the severe 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 [3].

## IV. P-SCAN Algorithm

In Priority Scan (P-Scan) all request from the I/O queue are divided into multiple priority levels. The Scan algorithm

DOI: 10.7763/IJIEE.2015.V5.502

is used within each level, which means the disk serves any requests passing 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 [3]. If found so, 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 [3], [4].

## V. FD-SCAN Algorithm

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 can estimate that it can be met. To be specific, 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 [4].

The SSEDO and SSEDV algorithms are based on the following assumptions:

Let, *ri* : be the I/O request with the i-th smallest deadline at a scheduling instance;

*di*: be the distance between the current arm position and requests *ri*'s positioning;

*Li*: be the absolute deadline of *ri* [4], [5].

The two algorithms maintain a queue sorted according to the absolute deadline, Li, of each request. A window of size m is defined as the first m request in the queue, i.e., the window consists of m request with smallest deadline.

## VI. SSEDO Algorithm

At a scheduling instance, the scheduler selects one of the requests from the window for service. The scheduling rule is to assign each request a weight, say '*wi*' for request '*ri*', where $w1 = 1 <= w2 <= \ldots <= wm$ and '*m*' is the window size, and to choose one with the minimum value of 'widi'. We shall refer to this quantity widi as the priority value associated with request *ri* [5]. If there is more than one request with the same priority value, the one with earliest deadline is selected. It should be clear that for any specific request, its priority value varies at each scheduling instances, since '*di*', *ri*'s position with respect to disk arm position, is changing as disk arm moves [5], [6].

The basic idea in this algorithm is 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 [6]. 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. There are various ways to assign these weights wi. The weights can simply set to

$$wi = \beta i\text{-}1 \quad (\beta \geq 1) \quad i = 1, 2, 3 \ldots\ldots m.$$

where $\beta$ is an adjustable scheduling parameter. Note that '*wi*' assigns priority only on the basis of the ordering of deadlines, not on their absolute or relative values [7].

## VII. SSEDV Algorithm

In the SSEDO algorithm described above, the scheduler uses only the ordering information of request deadlines and does not use the differences between deadlines of successive requests in the window. For example, suppose there are two requests in the window, and *r*1's deadline is very close but *r*2's deadline is far away. If *r*2's position is "very " close to the current arm position, then the SSEDO algorithm might schedule *r*2 first, which may result in the loss of *r*1. However, if r1 is scheduled first, then both *r*1 and *r*2 might be served. On the other extreme, if *r*2 deadline is almost same as r1's and the distance *d*2 is less than d1 but greater than $d1/\beta$, then SSEDO will schedule r1 for service and *r*2 will be lost. In this case, since there could be loss any way, it seems reasonable to serve the closer one (*r*2) for its service time is smaller [8]. Based on these considerations, we expect that a more intelligent scheduler might use not only the deadline ordering information but also the deadline value information for decision making. This leads to the following algorithms: associate a priority value of $\alpha\ di + (1\text{-}\alpha)$li to request *ri* and choose the request with minimum value for service, where li is the remaining lifetime of request *ri*, defined as length of time between current time and *ri*'s deadline Li and $\alpha$ ($0 \leq \alpha \leq 1$ ) is a scheduling parameter [9].

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 $\alpha$ or $\beta$, depending on the algorithm [9], [10].

## VIII. Real-Time Algorithms Design

### A. EDF Algorithm Design

Sort transaction on deadline in increasing order.

[repeat steps 2 and 3 till no more transactions in queue]
[set startTime, endTime, seekTime, currentHeadPosition, totalTransactionTime, turnAroundTime for all the transactions in the queue]
for first transaction set startTime = actualArrivalTime
for all transactions set the following parameters
set startTime = endTime [except first transaction]
set currentHeadPosition = blockedAccessed
set seekTime = blockedAccessed – currentHeadPosition
set totalTransactionTime = seekTime + transmissionTime
set endTime = startTime + totalTransactionTime
set turnAroundTime = endTime – actualArrivalTime
[ check transaction is miss or hit ]
If (endtime > deadline) Set successful = false
Else Set successful = true
End if
Exit

### B. FD_SCAN Algorithm

Sort t transaction on deadline in increasing order.
[repeat steps 2 and 3 till no more transactions in queue]
[set startTime, endTime, seekTime, currentHeadPosition, totalTransactionTime, turnAroundTime for all the transactions in the queue]
for first transaction set startTime = actualArrivalTime
for all transactions set the following parameters
set startTime = endTime [except first transaction]
set currentHeadPosition = blockedAccessed
set seekTime = blockedAccessed – currentHeadPosition
set totalTransactionTime = seekTime + transmissionTime
set endTime = startTime + totalTransactionTime
set turnAroundTime = endTime – actualArrivalTime
[ check transaction is miss or hit ]
If (endTime > deadline)
Set successful = false
Set endTime = startTime
Else
Set successful = true
End if
Exit

### C. P_SCAN Algorithm

Construct three queue namely MIN[100], MID[100], MAX[100] to store the transaction with minimum, middle or maximum priorities.
[set LOW_DL and UP_DL]
Set LOW_DL = minDeadline
Set UP_DL = maxDeadline
Repeat steps 4 and 5 till no more transactions
[store the transaction in the corresponding queue i.e. MIN, MID, MAX]
If deadLine > (LOW_DL + UP_DL) / 2
MIN[i] = deadLine
i =i+1
Else
If deadline < ((LOW_DL + UP_DL) / 4
MAX[j] = deadline
J = j+1
Else
MID[k] = deadLine
End if
End if
[set startTime, endTime, seekTime, currentHeadPosition, totalTransactionTime, turnAroundTime for all the transactions in the MAX queue]
for first transaction set startTime = actualArrivalTime
for all transactions set the following parameters
set startTime = endTime [except first transaction]
set currentHeadPosition = blockedAccessed
set seekTime = blockedAccessed – currentHeadPosition
set totalTransactionTime = seekTime + transmissionTime
set endTime = startTime + totalTransactionTime
set turnAroundTime = endTime – actualArrivalTime
[set startTime, endTime, seekTime, currentHeadPosition, totalTransactionTime, turnAroundTime for all the transactions in MID queue]
[for all transactions set the following parameters]
set startTime = endTime
set currentHeadPosition = blockedAccessed
set seekTime = blockedAccessed – currentHeadPosition
set totalTransactionTime = seekTime + transmissionTime
set endTime = startTime + totalTransactionTime
set turnAroundTime = endTime – actualArrivalTime
[set startTime, endTime, seekTime, currentHeadPosition, totalTransactionTime, turnAroundTime for all the transactions in MIN queue]
[for all transactions set the following parameters]
set startTime = endTime
set currentHeadPosition = blockedAccessed
set seekTime = blockedAccessed – currentHeadPosition
set totalTransactionTime = seekTime + transmissionTime
set endTime = startTime + totalTransactionTime
set turnAroundTime = endTime – actualArrivalTime
[ check transaction is miss or hit in MAX queue]
If (endTime > deadline)
Set successful = false
Set endTime = startTime
Else
Set successful = true
End if
[ check transaction is miss or hit in MID queue]
If (endTime > deadline)
Set successful = false
Else
Set successful = true
End if
[ check transaction is miss or hit in MIN queue]
If (endTime > deadline)
Set successful = false
Else
Set successful = true
End if
Exit

### D. SSEDO Algorithm

Sort t transaction on deadline in increasing order.
[set startTime, endTime, seekTime, currentHeadPosition, totalTransactionTime, turnAroundTime for the transactions with minimum deadline in the queue]
set startTime = actualArrivalTime
set currentHeadPosition = blockedAccessed
set seekTime = blockedAccessed – currentHeadPosition
set totalTransactionTime = seekTime + transmissionTime
set endTime = startTime + total TransactionTime set

turnAroundTime = endTime – actualArrivalTime
[find transactions with seek time within (tv) threshold]
For all the transactions in the queue with seek time within threshold (tv)
If ((blockaccessed – currentheadposition) < = tv)
set startTime = endTime
set currentHeadPosition = blockedAccessed
set seekTime = blockedAccessed – currentHeadPosition
set totalTransactionTime = seekTime + transmissionTime
set endTime = startTime + totalTransactionTime
set turnAroundTime = endTime – actualArrivalTime
go to step 3
[ check transaction is miss or hit ]
If (endtime > deadline)
Set successful = false
Else
Set successful = true
End if
Exit

### E. SSEDV Algorithm

Sort t transaction on deadline in increasing order.
[set startTime, endTime, seekTime, currentHeadPosition, totalTransactionTime, turnAroundTime for the transactions with minimum deadline in the queue]
set startTime = actualArrivalTime
set currentHeadPosition = blockedAccessed
set seekTime = blockedAccessed – currentHeadPosition
set totalTransactionTime = seekTime + transmissionTime
set endTime = startTime + totalTransactionTime
set turnAroundTime = endTime – actualArrivalTime
[find transactions with seek time within (tv) threshold]
For all the transactions in the queue with seek time within threshold (tv)
If ((blockaccessed – currentheadposition) < = tv)
Tot Exec Time = totExec Time+Total Transaction Time
if(totExec Time>0 AND totExecTime<minDeadline) for all the transactions in the queue
If ((blockaccessed – currentheadposition) < = tv)
set startTime = endTime
set currentHeadPosition = blockedAccessed
set seekTime = blockedAccessed – currentHeadPosition
set totalTransactionTime = seekTime + transmissionTime
set endTime = startTime + totalTransactionTime
set turnAroundTime = endTime – actualArrivalTime
go to step 3
[ check transaction is miss or hit ]
If (endtime > deadline)
Set successful = false
Else Set successful = true
End if
Exit

## IX. Conclusion

After developing these Algorithms for Real-time disk scheduling applications it is observed that 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.

## References


[1] R. Abbott and H. G. Molina, "Scheduling real-time transactions: a performance evaluation," presented at the 14th VLDB Conference, Los Angeles, California, March 1988.
[2] L. C. D. Pippo and V. F. Wolfe, "Real-time databases," *Book Chapter*, September 23, 1995.
[3] S. Z. Chen, J. A. Stankovic, J. Kurose, and D. Towsley "Performance evaluation of two new disk scheduling algorithms," *The Journal of Real-Time Systems*, 1990.
[4] K. A. H. G. Molina, "Scheduling i/o requests with deadlines: a performance evaluation CH2933-0/90/0000/0113," *IEEE Robert 1990.*
[5] J. R. Haritsa, M. J. Carey, and M. Livny, "Value-based scheduling in real-time database systems," May 15, 1991.
[6] S. Shih, Y. K. Kim, and S. H. Son, "Performance evaluation of a firm real-time database system," in *Proc. Second International Workshop on Volume*, pp. 116 – 124, issue 25-27, Oct 1995.
[7] B. Kao and H. G. Molina, "An overview of real-time database systems, in advances in real-time systems," pp. 463-86, 1995.
[8] Y. F. Zhu, "Evaluation of scheduling algorithms for real-time disk I/O," Department of Computer Science and Engineering, University of Nebraska – Lincoln, May 3, 2002.
[9] S. B. Thorat and S. Amdani, "Algorithms for disk scheduling in real-time database systems," in *Proc. IJCS-Dec-2009*, Serials Publication-vol. 4, pp. 131-135.
[10] J. Haritsa, M. Carey, and M. Livny, "Dynamic realtime optimistic concurrency control," presented at 11th IEEE Real-Time Systems Symposium, Dec. 1990.



**S. B. Thorat** was born on 3rd May, 1963. He received his Ph.D. in computer Sc. & engineering passed in the year 2009 at Bihar, India. He has completed M.E. in computer Sc. & engineering in the year 1996 at Patiala, Punjab, India, and M. Sc. in electronics in the year 1986 at Amalner, Maharashtra, India and MBA in HR & marketing, in the year 2011 at Periyar, India. he has qualified AMIE, LM-ISTE in the year 1997 . He has got 27 years of teaching experience (including 12 years administrative experience as a director, Shri Sharda Bhavan Education Society's Institute of Technology & Management, VIP-Road, Nanded and 13 years as lecturer, senior lecturer & associate professor at B N College of Engineering Pusad, Dist-Yevatmal & 2 years as Lecture at SSGM College of Engineering Shegaon, Dist-Buldhana).

Now, he is working as the director of Shri Sharda Bhavan Education Society's Institute of Technology & Management, VIP, Road, Nanded - 431 602. (M.S.), India, He has published one book and several research papers.