Disk Scheduling Algorithms and presentation of mathematical model with database

Uma Shanker Jayswal* Gaurav Kumar Jain Ravi Ranjan*****

ABSTRACT

Database management systems have entered the internet age. If too many users approach for information, then this degrades the system performance. The degradation may cause delay and trouble for particular end user in accessing the information. Accessing information in easy way and within certain time, by keeping its freshness, assessing user's requirements and then providing them information in time is important aspect.

Conventional database are mainly characterized by their strict data consistency requirements. Database systems for real time applications must satisfy timing constraints associated with transactions. The main objective of this paper is to initiate an enquiry in real time databases and presents, Mathematical model for Disk scheduling for real time database systems.

OVERVIEW

The vast majority of research in the field of real-time databases has focused in concurrency control and transaction scheduling. Scheduling transactions in real-time database involves determining which transactions execute when. Similar to tasks in other real-time systems, real-time transactions have priority and must be scheduled accordingly in order to meet specified timing constraints. However, unlike most other real-time process, real-time transactions access shared data. Therefore, real-time transaction scheduling must take into account the logical consistency. That is, concurrency control must be considered when scheduling real-time transaction. The primary scheduling goal in real-time systems is to satisfy the timing constraints of transaction [16, 17]. A Real-time database system requires integrated approach to consider data consistency requirements and timing constraints together in scheduling transactions.

The goal of transaction and query processing in real-time database is to maximize the number of successful transactions in the system [16].

General Parameters for Disk Scheduling:

The following parameters are chosen for Disk Scheduling.

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

Arrival time: Arrival time if the transaction

Total number of Transactions

Inter Arrival time

Average Execution Time

Transaction Size

Slack time: is an estimate of how long we can delay the execution of transaction and still meet its deadline

Access Time:

Access time =Seek Time + Rotation Latency + Transfer Time.

Where,

Seek Time is the time for the disk to move the heads to the cylinder containing the desired sector.

Rotation Latency is the transfer time needed to transfer data over the IO bus.

Priority: Priorities can be assign 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 r is about to miss its deadline.

Least Slack:

Slack time is an estimate if how long we can delay the execution of transaction and still meet its deadline. If S>=0 then it will finish at or before its deadline. A negative slack time results either when transactions have already issued its deadline or when we estimate that is cannot meet its deadline. The slack time of a transaction which is not executing decreases. Hence the priority if that transaction increases.

Classical Disk Scheduling Algorithm:

The four classical scheduling algorithms described below are well known.

FCFS

This is the simplest strategy in which each request is served in first-come-first-serve basis [3].

SCAN

This is also known as the elevator algorithm 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 [3].

SSTF

The SSTF, for shortest seek time first, algorithm simply selects the request closest to the current arm position for service [3].

A common feature o fall these classical scheduling algorithms is that none of them takes the time constraints of request into account. This results in poor performance of classical algorithms in real-time systems.

REAL-TIME DISK SCHEDULING ALGORITHMS

The real time disk scheduling algorithms like Earliest Dead line First (EDF), Priority Scan (P-Scan), Feasible Deadline Scan (FD- Scan), Shortest Seek and Earliest Deadline by ordering (SSEDO) and Shortest Seek ans earliest Deadline by Values (SSEDV)are discussed in brief prior to develop the mathematical madel fr them as under.

EDF ALGORITHM

The Earliest Deadline First algorithm is an analog of FCFS. Requests are orders 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 resources and data contention. This is due to Earliest Deadline steeply degrades in an overloaded system [12].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 their deadlines, thus delaying other transactions that might still be able to meet their deadlines [1].

P-SCAN ALGORITHM

In Priority scan (P-Scan) all request in the I/O queue are divided into multiple 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 [13]. 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 ti their deadline information. Specially, transactions relative deadline 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 and UP_DL) / 4, then the transaction receive

the highest priority. Otherwise the transaction is assigned a middle priority [4].

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 estimate that it can be met. More specially, a request that is n tracks away from the current head position has a feasible deadline d if d >= t + Access(n) where 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 easiest feasible deadline is the target and determines the scanning direction. The head scan toward the target servicing read requests along the way. These requests either have deadline later than the target request or have unfeasible deadline, 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 deadline have been (or will be) missed, the order of service is no longer important for meeting deadlines [13].

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[3][15].

The two algorithms maintain a queue sorted according to the absolute deadline, Li, of each request in the queue, i.e., the window consists of m requests with smallest deadline.

SSEDO ALGORITHM

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

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 assumping the seek time dominates the seervice 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 =βi-1 (β≥) i= 1,2,3…m.

Where β 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 [3].

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 r1's deadline is very close but r2's deadline is far away. If r2's position is "very" close to the current arm position, then the SSEDO algorithm might schedule r2 first, which my result in the loss of r1. However, if r1 is scheduled first, then both r1 and r2 might be served. On the other extreme, of r2 deadline is almost same as r1's and the distance d2 is less then d1 but greater than d1 / β, then SSEDO will schedule r1 for service and r2 will be lost. In this case, since there could be loss any

way, it seems reasonable to serve the closer one (r2) for its service time is smaller. 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 α di = (1-α)li to request ri and choose the request with minimum value for service, where li is the remaining life time of request ri, defined as length of time between current time and ri's deadline Li and α (0 ≤ α ≤ 1) is a scheduling parameter.

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 α or β, depending on the algorithm [3] [15].

SYSTEM ANALYSIS:

System requirement and analysis is the main step I System Development Life Cycle (SDLC). Systems analysis and design is a systematic approach to identifying problems, opportunities, and objectives; designing computerized information flows in organization; and designing computerized information systems to solve a problem.

In the System Analysis phase here, we are defining the system boundaries, opportunities and objectives and system requirement.

Real-time database systems comine the concepts from real-time systems and conventional database systems. Real-time systems are mainly characterized by their strict timing requirements in terms of deadline. Conventional databases are mainly characterized by their strict data consistency constrains. The primary scheduling goal in real-time systems is to satisfy the timing constraint of transaction.

A transaction is a collection of actions, which comprise a consistent transformation of the system-state. Each transaction, when executed alone, transforms a consistent stable into a new consistent state; that is , transactions preserve consistency of the database information. Interleaving transactions access to the database can maximize throughput and resource utilization. Therefore, various actions of different transaction need to be executed with maximal concurrency by interleaving actions from several transactions while continuing to give each transaction a consistent view of the database. A particular sequencing fo the actions from different transactions is called a schedule. A schedule that gives each transaction a consistent view of the database state is a called a consistent schedule.

Real time scheduling algorithms should therefore be based on the "inequalities" of transactions. Which is popular method is to assign a numeric priority to each transaction, with higher priority is given an upper hand in gaining access to system resources. A transaction has many attributes that may affect its priority mainly deadline, for, in time completion of transaction.

As a major asset of a computer system, efficient use of CPU cycles is very important. Conventional scheduling algorithms [14], as employed by most of the existing operating system, aim at balancing the number of CPU-bound and I /O-bound jobs to maximize system utilization and thought put. They are also designed to treat processes fairly; each one gets its fair share of the system resource. Other performance criteria include small job turnaround time, small waiting time, and fast response time.

However elaborated, these algorithms are not adequate for real-time transaction scheduling. This is because in a RTDBS, transaction should be scheduled according to their criticalness and the tightness of their deadlines. Even if this means sacrificing fairness and system through put.

Real-time scheduling algorithms should therefore be based on the "inequalities" of transaction. They should give preferential treatment to transactions, which are very critical, and with stringent timing constraints. A popular method is to assign a numeric priority to each

transaction, which reflects its relative urgency. A transaction with higher priority is given an upper hand in gaining access to system resources. A transaction has many attributes that may affect its priority, those attributes that are most relevant to a RTDBS for making decision about different scheduling techniques.

Most of the real time transaction scheduling algorithm assumes that the transaction scheduler is supposed to have no idea about transaction's computing time and resource requirement in advance, which is the case of soft or firm deadline applications. Priority of a transaction is thus assigned based on its timing constraint (I.e., deadline) and / or value, without considering information about its routine behavior. Also, conflict resolution schemes used in real time concurrency control protocols do not utilize such information. Consequently, they cannot guarantee that each transaction will complete by its deadline, but try to minimize the deadline miss ratio of transactions or to maximize the total value of transactions have different values.

Real word examples of applications supporting soft or firm deadline transaction are provided in [2]. Banking system and airline reservation system usually process soft deadline transactions. When a customer submits a transaction within its deadline, the customer prefers getting the response late than not getting it at all. The stock market trading is an example of applications supporting firm deadline transactions. If, for instance, a transaction is submitted to learn the current price of a particular stock, the system should either return the operation at all, because conditions in the stock market changes fastly.

The scheduler thus schedule each transaction based on its transaction deadlines using *Earliest Deadline first, Priority Scan, Feasible Deadline Scan, priority Scan, Shortest Seek and Earliest Deadline by ordering and Short Seek and Earliest Deadline by value least slack time first etc. algorithms.*

Much of the work done on real time job scheduling, focuses mainly on CPU scheduling. Transaction scheduling, however, involves not only the CPU. In fact, due to the extensive data processing requirements of a database system, resources such as data, disk I/O, and memory are also subject to serve competition among concurrently running transactions. Careful scheduling the use of these resources is very important to the performance of RTDBSs.

So far less consideration is given to the architecture of the RTDBS, using main memory data bases, we can keep some database tables in memory in order to provide freshness of information, which is valid for shorter period. Using such technique we are able to reduce the disk I/o and achieve predictability, consistency and timeliness of transaction.

From the above analysis of system, we have investigated various real time disk scheduling algorithms like EDF, P-Scan, FD-Scan, SSEDO and SSEDV. The EDF is analog to first come first serve except transactions are ordered according to deadline and the request with earliest deadline is serviced first. 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.

In P-scan all request in the I/O queue are divided into multiple priority levels. The disk serves any requests that is passes in the current served priority level until there are no more requests in that direction. All the I/O requests are uniformly distributed between LOW_DL and UP_DL. If a transactions deadline is greater than (LOW_DL+UP_DL) / 2, then it is assigned the lowest priority. If the deadline is less then (LOW_DL +UP_DL) / 4, Then the transaction receives the highest priority. Otherwise the transaction is assigned a middle priority.

In FD-Scan, the track location of the request with earliest feasible deadline is used ti determine the scan direction. A request that is n tracks away from the

current head position has a feasible deadline d .= t + Access(n0 where t is the current time and Access(n0 is a function that yields the expected time needed to service a request n tracks away.

In the SSEDO algorithm, the scheduler uses the ordering information of request deadlines. SSEDO assign each request a weight, say wi for request ri, widi is the priority value associated with each request ri. The idea is to give higher priorities t requests with smaller deadlines so that they can receive service earlier. Request with large deadlines "very" close to the current arm position (which means less service time), it should get higher priority.

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 values for service (remaining life time of request i.e. length of time between current time and request deadline).

For the above analysis of system, for transaction scheduling in Real time database system, we have developed the **mathematical model for real time disk scheduling for** all the above fine algorithms with no preemptive policy for soft deadline transaction. 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. Also we are trying to compare the performance of these algorithms under same work load condition.

SYSTEMDESIGN:

Our subject of work is the investigation of the various real time disk scheduling algorithms like EDF, FD-SCAN P-SCAN, SSEDO AND SSEDV. General investigation architecture is depicted in the figure1.

TRANSACTIONS ⟹ALGORITHMS EVALUTION ⟹ PARAMETERS

For the development of the mathematical model for above said algorithms first we have formulated the disk scheduling problem for real time database systems and then implemented the mathematical model for all the algorithms.

Mathematical model for Real-time Disk scheduling problem

In general, the transactions in real –time database systems arrive in two fashions i. e. random and constant. In the mathematical model, we have assumed that the arrivals of transactions are random or constant. In addition to this, the following are the some assumptions which are required.
Initial head position is always at the block number 4.
Transaction maximum size is fixed, for instance, 100.

1) Slack factor = 1.2.
2) First arrival's, inter arrival time is the start time of the system.

Mathematical model is based on the queuing theory. In queuing theory, the arrival fashion can be of random, constant or exponential type and service fashion can be of random, constant or exponential type. General queuing model, always try to satisfy all the arrivals. Like this, here in real-time disk scheduling problem we would like to satisfy r meet the deadline f most of the transactions i.e. maximization of number of successful transactions or minimization of number of failure of transaction. For the evolution of mathematical model of all these algorithms performance, we have summarized the comparison on the basis of two evolution parameters, namely utilization of system and success ratio.

Basic parameters involved in the mathematical model are: transmission time, transaction arrival time, total transaction time, transaction arrival rate, total number of transaction, transaction arrival fashion (or distribution), seek time, average execution time,

transaction inter arrival time, actual arrival time, transaction turnaround time.

As soon as the transaction enters the queue, a unique ID is assigned to the transaction. The transaction is identified by its ID till the time it is available in the scheduler.

The type of operation the transaction is going to performed like read, write, read write and read write and compute. The read Time, write Time and compute Time are randomly assigned by the system.

The transaction arrival rate, *transaction arrival rate* will be input from the user.

For simulation we have considered maximum disk size of 100 blocks.

The block accessed, *block Accessed* will be assigned randomly between (1 to max disk size i.e. 100)
The inter arrival time inter Arrival Time (IAT) of the transactions are calculated depending on the arrival fashion of the transaction. For random arrival inter Arrival time – (1/ transaction arrival rate) log (1/block Accessed).

On the basis of equation (1), we get the interval arrival time depends on the transaction arrival fashion. In the case of constant transaction arrival fashion, ITA is fixed.

The required different parameters for simulation are calculated on the basis of the following equations.

Arrival Time, arrival time will be arrival Time = arrival time + inter arrival time. (2)
The actual arrival time, actual arrival time is the time when the transaction enters the scheduler and it is calculated using the following equation actual arrival time = arrival time (3)
Arrival time get overrides when the next transaction comes. The transmission factor, transmission factor is set depending on the type of operation for read operation transmission factor = 0.6 (4)
For write operation transmission Factor = 1.2 (5)
For read and write operation transmission Factor=1.2 (6)
For read write and compute operation transmission factor (7)
The average execution time, average execution time is calculated as follows average execution time = 1.5 8 transaction size (8)

The deadline, deadline for each transaction can be calculated using the following equation dead line = arrival time = (slack factor* average execution time) (9)

Calculated Properties of Transaction:

After getting the above parameters values, following parameters of transaction are calculated.
Seek time, seek time is the time required to move the arm head at the appropriate cylinder of the disk. In our simulation, we have calculated the seek time using following equation. Seek Time= block Accessed –current Head Position (10)
Transmission time, transmission time is the time required to transmit the data through I/O bus, Transmission Time= transaction Size* transmission Factor (11)

Total transaction time, *total Transaction Time* is seek time plus transmission time.
Total Transaction Time = seek Time =transmission Time (12)
Start time, start time the time when the transaction starts its execution. For the first transaction start Time = 0, for the next transaction Start Time = end time, the end time of previous transaction End time, end Time the by which the transaction completes its execution. *End Time = start Time = total Transaction Time* (13)

Turnaround time, turn Around Time is the total time from the arrival of the transaction till its execution. Turn Around Time = end Time – actual Arrival Time (14)

From simulation point of view, all the transaction are generated with either random transaction or constant transactions. The values of some parameters of mathematical model are constant values; these constant values are empirically drawn.

Evolution parameters for the simulation are US and SR where, US is utilization of system and SR is success ratio.

Table Profile: Table profile shows the structure of the table, where disk and transaction parameter values are stored, table structure is shown below:

Column name Description

Transaction No. Transaction ID block Accessed block number to be accessed IAT inter Arrival Time of Transactions Arrival Time Arrival Time of the transaction

Start Time when transaction starts execution End Time when transaction completes execution Seek Time

Time required to move the head to the appropriate block of disk TA Time transmission Time Transaction status True Transaction Time Size of Transaction Status true if transaction is met, else false another table is used to store the operation, deadline and priority of transactions

Column Name Description

Transaction No. transaction ID read true if read operation else false write true if write operation else false Deadline of the transaction Priority of transaction.

CONCLUSION

After developing the Mathematical model for **Real-Time disk scheduling problem** and after comparing all the Algorithms it is observed that in EDF transactions are order according to deadline and the request with earliest deadline is serviced first. Priority scan decides 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 windows.

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.Garcia-Molina, “Scheduling real time transcation: A performance evaluation,”proceedings of the 14 VLDB conference,los angeles, California, (1988).

[2].N. Audsley, A.Burns, “real time system scheduling ”, technical report no.YCS134,department computer science, the university of York,UK,(1990).

[3]. Shenze Chen, Joh A.Stankovic, James Curose and DonTowsley, “performance evaluation of two new disk scheduling algorithmn”, The Journal of real time system, (1990).

[4]. S.Chen, J.A.Stankovic, J.F.Kurose, and D.Towsley, “ performance evolution of two new disk scheduling algorithm for real time systems ”, the Journal of real time system, 3(1991)307-336

[5]. Z.Dimitrijevic, R.Rangaswamy, and E.Chang, “design, analysis, and implementation of virtual IO”.(2002)

[6]. Value Based Schedduling In real time database. Systems.Jayant R.Haritsa, Michael J.Carey, and Miron Livny.Recieved(1991).

[7]. Kao, B.and Garcia-Molina,H.,”overview of real time database systems,”in advances in real time systems (ed.S.H.Son), (1995)463-86.

[8]. "Evaluation of sceduling algorithm for real time disk I/O"-YIFENG Zhu, department of computer science and engineering, university of Nebraska-Lincoln,(2002).