Distributed Real-Time Database Systems

Saksham Mishra

M-Tech, Department of Computer science, B.I.T Gida Gorakhpur,U.P,India

ABSTRACT

A deadline, which specifies that it must be finished before a certain time in the future, is one way that a transaction's timing restriction is represented. If the deadline is missed in a firm real-time transaction, nothing terrible happens. Even after the deadline has passed, a soft real-time transaction carries some value, but this value usually decreases with time. The major purpose of DRTDBS is to reduce the proportion of transactions cascade rollbacks that miss their deadlines, as opposed to typical databases, where the main goals are to decrease response times and increase throughput. Since the database management techniques for accessing and editing data in DRTDBS need not only assure database consistency but also to satisfy the timing requirements, scheduling real-time transactions is significantly more difficult than regular real-time scheduling. This chapter's objective is to introduce several facets of DRTDBS, the problems, and difficulties they raise, as well as the research is done for this thesis.

Keywords :- *Transactions,Rollbacks,Real-time transactions,Database Consistency*

1.INTRODUCTION

1.1 Overview

Numerous applications that heavily rely on database systems for the proper storing and information retrieval from various remote sites, such as medical monitoring, military tracking, stock arbitrage systems, aircraft control, network management, and factory automation, among others, have specific timing requirements [3,5]. These applications highlight the requirement for distributed realtime database systems(DRTDBSs). DRTDBS is a grouping of various databases that are dispersed throughout a computer network and are conceptually related to one another. They assist transactions with clear timing restrictions [6]. A deadline, which specifies that it must be finished before a certain time in the future, is one way that a transaction's timing restriction is represented. Based on the impacts of missing deadlines, the transactions can be divided as hard, firm, or soft types. A difficult real-time transaction must adhere precisely to its deadline. A disaster could happen if a deadline is missed. If the deadline is missed in a firm real-time transaction, nothing terrible happens. However, once the deadline has passed, the results are useless [5]. Even after the deadline has passed, a soft real-time transaction carries some value, but this value usually decreases with time. The major purpose of DRTDBS is to reduce the proportion of transactions cascade rollbacks that miss their deadlines, as opposed to typical databases, where the main goals are to decrease response times and increase throughput. Since the database management techniques for accessing and editing data in DRTDBS need not only assure database consistency but also to satisfy the timing requirements, scheduling real-time transactions is significantly more difficult than regular real-time scheduling. This chapter's objective is to introduce several facets of DRTDBS, the problems, and difficulties they raise, as well as the research is done for this thesis. The contributions and structure of this thesis are discussed after an examination of the fundamental problems and research difficulties crucial to the effectiveness of DRTDBS.

1.2Performance Issues And Research Challenges

With the competing demands of preserving data consistency and achieving transaction deadlines, the deployment of

DRTDBS is challenging. The complexity arises from the unpredictable response timings of the transactions. Due to concurrency control, input/output operations, and communication delays, each distributed transaction accessing a data item takes a different length of time [6,7]. The management and scheduling of the system resources in DRTDBS can take into account the time limitations in addition to ensuring the consistency of the underlying database.

To the greatest extent possible, access to the CPU, I/O devices, main memory, and shared data should be managed to meet the transaction deadlines. Transaction scheduling is one of the most significant factors in the architecture of DRTDBS [6]. In DRTDBS, the scheduling of transactions is carried out in accordance with the priority given to the transactions and includes both CPU scheduling and data scheduling. Because priorities dictate the order in which transactions request to access resources, which in turn determines how likely they are to fulfil deadlines and cascade rollback, the importance of the priority assignment policy becomes crucial in determining the system performance.

When conflicts arise in conventional databases, the preferences typically stem from either fairness or resource usage. However, the DRTDBS schedules transactions in accordance with the urgency of the transactions, which establishes their priority. Very few studies have attempted to solve the priority assignment problem. Fairness and optimal resource use become secondary objectives when a transaction's priority is typically assigned based on its deadline, such as in the earliest deadline first (EDF) priority assignment strategy [7]. Two issues may result from this. First, if transactions that are closer to completion are abandoned in favour of transactions with a higher priority, more CPU resources are squandered.

Second, longer transactions could be more difficult to complete, leading to a hunger issue. Execution of a global transaction in a distributed system needs the execution of cohorts on multiple sites. The majority of DRTDBS priority assignment heuristics take into account the sequential execution of a transaction's subtasks (cohorts). Other heuristics—aside from ultimate deadline (UD)—are inappropriate when a transaction's subtasks (cohorts) are carried out concurrently. When data contention is not minor, the UD also loses its effectiveness. The global atomicity for distributed real-time transactions is supported in large part by the atomic commit protocols. These procedures are used to make sure that all cohorts concur on the transaction's final result.

They often demand for the participation of different sites and the exchange of numerous messages in various phases, as well as the maintenance of data logs for failure recovery. This greatly lengthens the transaction execution time and may make it more difficult for the system to complete transactions by the due date [3,6]. It is difficult to meet the deadline for all transactions in DRTDBS due to the distributed nature of the transactions and the existence of other sources of unpredictability which including data access conflicts, uneven transaction distribution across sites, variable local CPU scheduling time, communication delay, failure of coordinator and cohort's sites, etc. Because the blocking duration of the waiting cohorts owing to execute-commit conflict may grow longer, the unpredictable nature of the commitment phase makes it more dangerous.

Hence, due to the unique properties of the committing transactions and uncertainty in the commitment process, the creation of an efficient protocol for committing transactions is another major challenge that influences the performance of DRTDBS [6]. Crucial database system resources are CPU, main memory, drive and data items. Buffer space in the main memory is allotted to a transaction before it can begin to be executed [3,5]. A transaction might not be able to proceed if the main memory is running low. The quantity of concurrently executable transactions is thus constrained by the system's memory capacity. In big-scale the real time database systems, the performance of the transactions will be severely impacted by the low memory.

The efficient use of main memory space in data driven applications is also another difficult problem. Temporary records are made as transactions are carried out in order to keep track of their progress. Up until the transaction commits, these records are retained in the main memory. The amount of main RAM used by this is significant. Memory will be tied up for a long time and unavailable for other transactions because uncertainty in the commitment phase may cause the transaction to remain in the system for a long time. Thus, commit techniques that reduce the number of temporary objects created are required.

Other intriguing issues are brought up by the development and application of DRTDBS. Predictability and consistency, which are essential to real-time transaction processing yet may call for conflicting behaviours, are among these issues. We may need to block some transactions in order to maintain consistency. However, blocking certain transactions could result in unpredictable transaction execution and a breach of time restrictions. Other unpredictable events include communication lags, site outages, and interactions between transactions and the underlying operating system and I/O subsystems. Data access mechanisms and invariance, new metrics for database integrity and throughput, maintaining global system information, reliability, fault tolerance, failure recovery, etc. are some more design considerations for DRTDBS [6,7].

Once more, there is no method that is well-designed for scheduling the CPU as the main resource in the DRTDBS. Despite the fact that these concerns have been the subject of extensive investigation, there are still many difficult and unresolved problems. We have limited our attention to only a few of these topics because of how diverse they are. Designing new priority assignment policies is part of our job, and we compare their effectiveness to that of current policies and protocols.

We presumed that the transactions are soft to firm in real-time and that the data items they access are known before the transactions begin to be executed.

When there is a conflict between two transactions with the same deadline, Two priority allocation approaches are used for the transactions which are static peer sub-transactions and dynamic priority no of peer executed.

1.3Contributions Of The Thesis

The concept proposed in this thesis provides different solutions for some of the issues mentioned above. The contributions of this thesis may be described as follows:

A. A new priority allocation policy to prioritize sub-transactions executing in parallel along with the method to compute the deadlines of the local and the global transactions has been proposed. In our scheme, each sub-transaction is assigned an initial priority which is directly proportional to the number of peer transactions at other execution sites. A static intermediate priority of the subtransaction is calculated when a data contention occurs for the same deadline and the initial priority of the newly arrived cohort (TA) is higher than the priority of the dynamically allocated priority of cohort (TD). The intermediate priorities are based on the no of executed cohorts at other sites of particular local transactions. This approach minimizes the miss and cascade rollback of near completion low priority cohorts. The proposed scheme has been compared with the Earliest Deadline First (EDF) priority assignment policy.
B. Discussion of the existing Priority allocation approach in DRTDBS and the protocols using it.

1.4 Organization of Thesis

This Thesis comprises five chapters which are as follows:

- Chapter 1 consists of an introductory part and the objective of the work.
- Chapter 2 consists of a background and literature review.
- Chapter 3 consists of proposed work with our aim to design a task scheduling policy
- Chapter 4 is about the discussion of existing priority allocation policy and metrics.
- Chapter 5 Conclusion and future work.

2.BACKGROUND AND LITERATURE REVIEW

2.1 Background

In today's culture, databases and its management systems are now an important part of daily life. Due to the change in information technology, quick access to information and effective information management are now essential for the success of any activities in industries like business and others that are similar. Database systems that allow real-time transactions are known as distributed real time database systems (DRTDBSs). They have a wide range of uses, including telemedicine, banking, air traffic control, and stock market trading.

In DRTDBS, there are two distinct types of transactions: local and global transactions. Local transactions, which are only carried out at the site from where they originate whereas the global transactions are distributed real-time transactions that are carried out at several sites [6,7] . In a typical distributed real-time transaction paradigm, a coordinator process runs at the location where the transaction is lodged, and cohorts, a collection of other processes, run at various locations where the required data items are located.

2.2Real Time Systems

Real-time systems are ones whose tasks are associated with constraints on their completion times such as deadlines. The performance and the correctness of the systems depend heavily on how well these constraints are met. For example, an on-board flight control system needs to respond to a change of wind speed by adjusting the ailerons to maintain a steady flight in a timely manner [3,7]. A nuclear reactor monitoring system needs to add coolant to an overheating reactor in a short time frame. For these kinds of systems, scheduling tasks to meet their deadlines is of utmost importance. Recently, there is a trend of using databases to support real-time applications. Traditional real-time systems manage only small amount of data (e.g., hundreds of sensor readings). Usually, they store information in certain ad-hoc data structures, such as pointers and arrays. As these systems evolve, however, they need to handle more data. The ad-hoc approach of data management becomes inefficient and discourages data sharing[1,2]. In the meantime, many data intensive applications requiring real-time supports have emerged. RTS can be categorized into several forms, according to the temporary restrictions:

- **Critical real-time systems**

A Critical real time systems (RTS) are those where breaking the temporary limits can have unavoidable negative effects on both the system and its users. Hard real-time systems are another name for these important real-time systems. These include things like systems for kidnapping and rescue, military tactical training, and alarms for earthquakes.

- **Non-critical real-time systems**

Systems classified as non-critical RTS are those in which failure to meet deadlines results in a reduction in the system's operation or performance. Soft real time is another name for these non-critical RTS. A common example of this is multimedia systems, which include audio, pictures, and video[1].

One more classification of real-time systems depends on the number of processors used for the execution of the task are:

– Conventional real-time systems: the systems where the tasks are running on a single processor.

– Distributed real-time systems: systems where different processors are used for different tasks. Additionally, when processors use the same operating system, these systems may be highly connected. These real-time distribution systems include a number of characteristics, including concurrent systems, reactive systems, systems that run in challenging settings (such as noise, high temperatures, etc.), and systems that are innately dependable and fault-tolerant.

Types of real-time systems based on deadline constraints:

- **Soft real-time system:** This kind of system has a reasonably low possibility of occasionally missing its deadline. If the deadline is missed, there are no serious consequences. The usefulness of results produced by a soft real-time system progressively decreases with time.

- **Hard real-time system:** With this approach, the due date can never be missed. Missing the deadline could have disastrous effects. The usefulness of outputs produced by a hard real-time system rapidly decreases and may even turn negative as tardiness increases. How far behind schedule a real-time system completes a task is referred to as "tardiness." a flight control system, as an example.

2.3 Distributed Real Time Database System

. RTS are frequently related to important applications where the safety of people or costly equipment may be at risk. Therefore, even if an action or computation is functionally accurate, it may be

Real Time Systems (RTS) are those systems whose accuracy relies not only on the temporal but also on the logical characteristics of the outcomes produced worthless or even destructive in such systems if it is executed too late (or too early) or uses temporally incorrect input[6].

RTS applications are getting more and more complicated as they develop, and frequently they need quick access to and predictable processing of enormous amounts of real-time data. The term "DRTDBS" refers to data-base systems that are specifically created for the effective processing of these kinds of real-time data. As a result, DRTDBS are a group of various, logically connected databases that are dispersed across a computer network and in which transactions are subject to explicit timing limitations, typically in the form of deadlines. Data items shared across transactions in such systems are dispersed across distant sites. To preserve the logical consistency of the database, access to these common data objects must be restricted. Due to the distributed nature of the transactions and the requirements for database consistency, meeting the timing restrictions of various real-time operations in distributed systems may be challenging. Partially ordered sets of read and write operations are produced when user programs interact with databases. A transaction is a name given to this series of database activities. A transaction changes the database system's existing consistent state into a new consistent state. There are two different kinds of transactions in DRTDBS: global and local.

Local transactions are only carried out at the originating site (parent site), whereas global transactions are dispersed real-time transactions carried out at several sites. The following is a typical representation of a distributed transaction. One process, known as the coordinator, runs at the location where the transaction is submitted, while a group of additional processes, known as cohorts, run at locations where the transaction has access. The transaction is a discrete piece of labor that can either be finished entirely or not at all. In order to ensure the uniform commitment of distributed transaction execution, a distributed commit protocol is required. The transaction is presumed to be successful upon the commit process, and all updates should therefore be permanently incorporated into the database[6,7].

The database management system must cancel or eliminate all of the impacts of an missed operation because it means the transaction has failed. A transaction is a unit of execution that is "all or nothing," to put it simply. Real-time transactions can often be divided into three categories: firm, soft, and hard. No hard real time transaction should ever miss its deadline, and the system needs to guarantee this. On the other hand, any soft real time transaction deadline violations may simply lead to system performance degradation.

The number or percentage of deadline violations, as well as their mean or worst-case reaction time, are the key performance indicators for soft real-time transactions. Firm real-time transactions are a specific type of soft real-time transactions that are terminated once their deadline has passed. The quantity or percentage of deadline violations serves as the performance metric. Real-time transaction completion may provide value to the whole system. A value function of time is the relationship between the value provided by a real-time transaction and the time it takes for it to complete. Once the deadline has passed, a firm real-time transaction is worthless. A hard real-time transaction loses value when it is late for its deadline. It implies that a disaster could happen. A soft real-time transaction's value could gradually decline once it misses its deadline.

2.4 Existing Approach to priority assignment

A real-time database system is typically a component of a substantial and intricate real-time system. In that they are both units of work and scheduling, tasks in RTS and transactions in DRTDBS are comparable. Tasks and transactions, however, are distinct computing notions, and these distinctions have an impact on how they should be scheduled and handled. On the basis of real-time restrictions like ED, HV, and HRU, priority assignment policies for distributed real-time database systems are created. These systems typically carry out the transaction that, according to a calculation, has the highest execution priority. The notations used are listed below, along with a brief description of the following transaction scheduling policy.

t_i	transaction in system
$P(t_i)$	priority of i-th transaction
$V(t_i)$	value of i-th transaction
$D(t_i)$	deadline of i-th transaction
$Arr(t_i)$	arrival time of i-th transaction
$S(t_i)$	slack time of i-th transaction
$Crt(t_i)$	communication delay left of transaction i-th transaction
$Ert(t_i)$	remaining execution time of i-th transaction

$F(t_i)$ failure ratio of executing i-th transaction

$Lrt(t_i)$ locking time remaining to access item for i-th transaction

W Weight of adjusted transaction's value and urgency

2.4.1 Earliest Deadline First (EDF)

This states that the absolute deadlines of current jobs, where the deadline of a job depends on the time of its next occurrence, are inversely proportional to the precedence allocated to tasks. Transactions with impending deadlines are given special priority by EDF.

The formula for the above priority assignment is given by

$$P(t_i) \leftarrow \frac{1}{D(t_i)}$$

2.4.2 First Come First Serve (FCFS)

In this priority allocation approach basically, timestamp ordering is followed where transaction that arrives first is given high priority than transaction arriving later after some time. It is biased to the arrival time of the transaction.

The formula for the above priority assignment is given by

$$P(t_i) \leftarrow \frac{1}{Arr(t_i)}$$

2.4.3 Least Slack Time First (LSF)

Transactions with less slack time will receive better priority under the LSF policy. The length of time the cohort can wait to finish before its deadline is referred to as the slack time. In this, the relationship between the transaction's Priority and slack time is inverse.

The formula for the above priority assignment is given by

$$P(t_i) \leftarrow \frac{1}{S(t_i)}$$

2.4.4 Highest Value (HV)

The drawback of EDF is that it doesn't take transaction values into account; as a result, transaction priority is determined by the value acquired. HV stands for giving high priorities to transactions with high values.

The formula for the above priority assignment is given by

$$P(t_i) \leftarrow V(t_i)$$

2.4.5 Highest Reward and Urgency (HRU)

HV is more concerned with completing high-value transactions than EDF is. The urgency of a transaction is not taken into account, though. HRU takes into account both the timeframe and value as design variables in order to overcome the drawback. A transaction having a high value and the shortest remaining execution time is given a high priority.

The formula for the above priority assignment is given by

$$P(t_i) \leftarrow \frac{V(t_i)}{Ert(t_i)} - S(t_i) * W$$

2.4.6. Flexible High Reward (FHR)

In order to decrease the needless waiting time for a remote transaction, FHR is built for a distributed RTDBS by extending the HRU priority technique together with a communication delay factor.

The formula for the above priority assignment is given by

$$P(t_i) \leftarrow \frac{V(t_i)}{Ert(t_i) * Lrt(t_i)} * F(t_i) - \frac{S(t_i)}{iif(Crt(t_i) \neq 0 , Crt\left(t_i\right) ,1)}$$

2.5 Literature Review

This section offered numerous excellent initiatives for cloud scheduling strategies that have provided a clear resolution to this issue.

UdaiShanker · ManojMisra · Anil K. Sarje (2008) gives an overview and background study of Real time database systems. iscusses the The research that has been conducted so far on issues like priority assignment policy, commit protocols, and optimising memory use in non-replicated/replicated environments with regard to distributed real time transaction processing is then surveyed after discussing performance issues that are significant to DRTDBS.

Sarvesh Pandey, UdaiShanker (2020) gives Reads-write Avoid starvation, and a Priority Inversion period Declined (RAPID) commit protocol has been proposed to effectively address the aforementioned issue by reducing the number of reader transactions that are either requested or currently holding the conflicting data item in read mode by missing some of them (if possible). According to the simulation results, the RAPID protocol outperforms the 2PC and PIC protocols.

Manoj Kumar (2016) this paper presents a review and comparison of distributed database system commit mechanisms, blocking, atomicity, two-phase commit (2PC), and three-phase commit (3PC), distributed database systems, distributed transactions, commit protocols,

Swati Gupta, Kuntal Saroha, Bhawna (2011) This essay's goal is to provide an introduction to distributed databases, which are now seeing a surge in popularity. Due to the steadily growing demand for trustworthy, scalable, and accessible information, distributed databases and client/server applications are becoming more and more important in today's business environment

3. Problem Framework

Abbot R. and Garcia-Monila H. were the first to examine the effectiveness of various scheduling policies for soft deadline-based transactions. Through simulation, they investigated the effectiveness of three priority assignment approaches—FCFS, EDF, and LSF—with various concurrency control techniques, including serial execution (SE), high priority (HP), and conditional restart (CR). Abbot R. and Garcia-Monila H. have once more reported on their groundbreaking work in RTDBS performance evaluation of several scheduling choices for a real-time database system with disc and shared locks. Along with concurrency control methods including wait, wait-promote, high priority, and conditional restart, the scheduling algorithms, EDF, FCFS, and LSF were used for this study.

In this work we propose a novel priority assignment policy for the RTS transactions. The heuristic proposed here will improve the overall system efficiency and performance by servicing the transactions that have most peers at other sites executed asynchronously and waiting for a few peers to execute to commit successfully the global transactions that in turn decreases the cohorts misss/ rollbacks thus decreasing the wastage of system processing time.

3.1 Proposed Approach

Suppose we have a global transaction arriving in a system that is broken down into n-sub transactions that can execute parallelly at different sites, If all other n-1 peer transactions are executed successfully and any one transaction at any site misses its deadline and get about then all the remaining n-1 executed and waiting for commit transactions need to rollback all the operations performed thus making an immense wastage of resources and time. To reduce transaction rollbacks from the above scenario we propose a novel approach to priority allocation where a transactions scheduler will prioritize transactions with the same EDF using the below approaches.

1. **Static Priority allocation:** When a global transaction is reduced to smaller transactions then each sub-transaction stores total peers transactions as no_peer_trans in its member variable, So this no_peer_trans variable is consumed at the data node for prioritizing the particular transaction.

2. **Dynamic Priority allocation**: In this approach after every transaction commit message from the coordinator the priority(peer transactions commuted) is piggybacked with it thereby decreasing messages in the system.

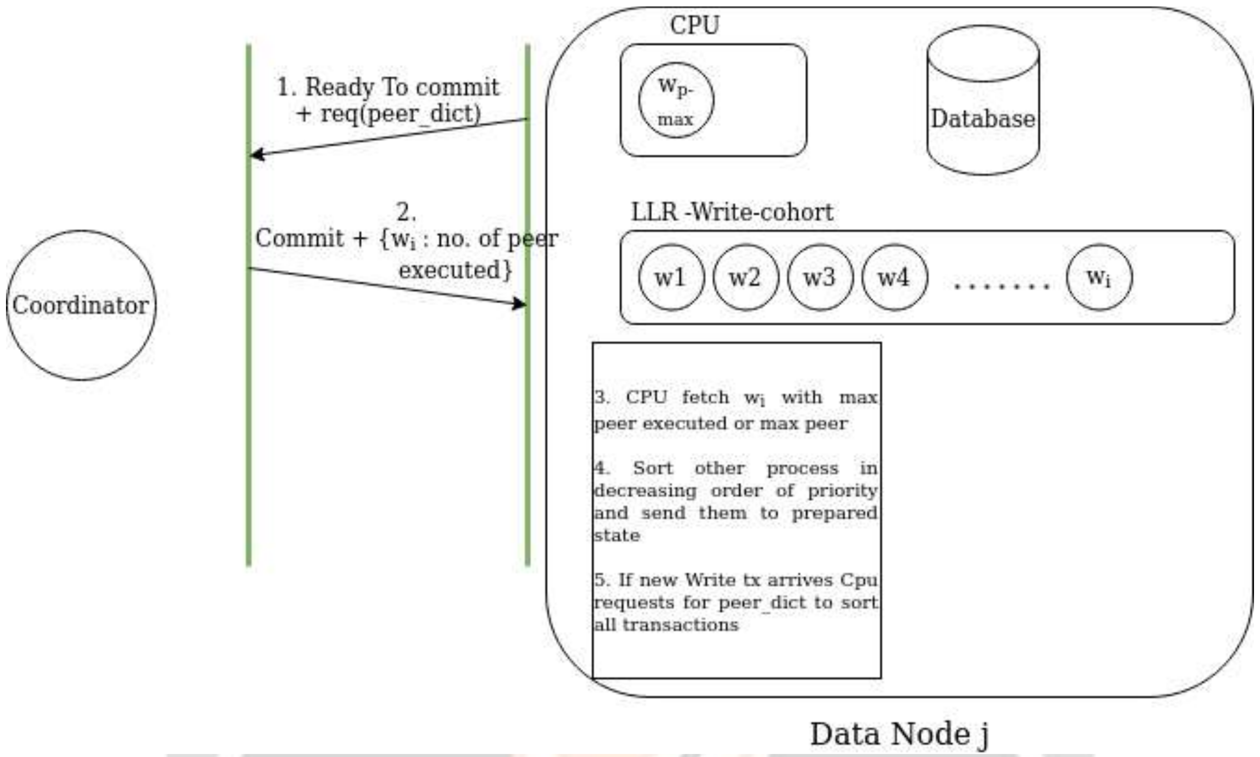

CPU
w_{p-max}
Database
1. Ready To commit + req(peer_dict)
2. Commit + {w_i : no. of peer executed}
LLR -Write-cohort
w1
w2
w3
w4
....
w_i
Coordinator
3. CPU fetch w_i with max peer executed or max peer
4. Sort other process in decreasing order of priority and send them to prepared state
5. If new Write tx arrives Cpu requests for peer_dict to sort all transactions
Data Node j

Fig 3.1: Priority fetch for Dynamic Priority allocation

In the above workflow, we can easily understand how to fetch executed peer's priority data from the coordinator without the need for flooding the network with excess data packets. Initially, static priority is assigned when there are no executed transactions in the system at any site. The proposed priority assignment heuristic is based on the number of peer transactions executed at other sites. The initial priority(static allocations) of the cohort (T_i) of the transaction (T) is computed as below.

$$P_{static} = N_{Peers\ at\ other\ sites}$$

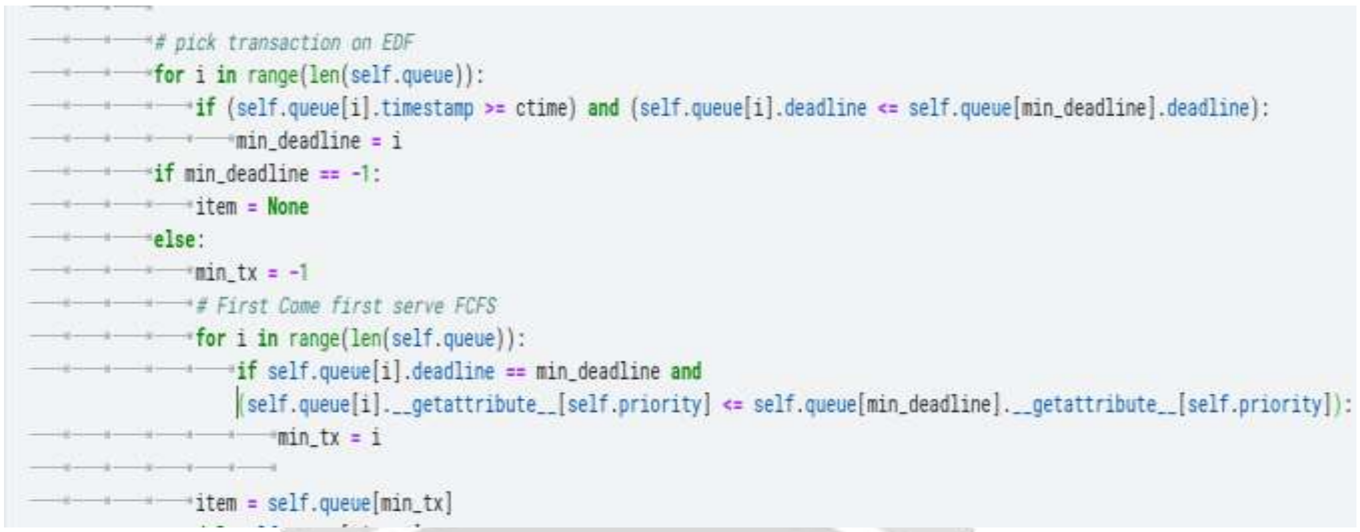
After committing some transactions Dynamic Priority is allocated to the cohort (T_i) of the transaction (T) is computed below.

$$P_{dynamic} = N_{Executed\ Peers\ at\ other\ sites} / P_{max\ -static}$$

The transaction manager is in charge of overseeing the transaction across many sites. A transaction enters the commit stage when an atomic commit protocol is carried out, once all of its read-write activities have been completed. Each site has a communication interface that handles sending and receiving data and messages to and from other sites. Depending on the policy, the scheduler gives each transaction a priority. Based on the priority factors, the scheduler organizes data access requests.

3.2 Proposed Algorithm

Simulation of Distributed Realtime distributed database is a very complex task in itself. The below algorithm was tested in the simulated environment for comparison with the existing approaches.

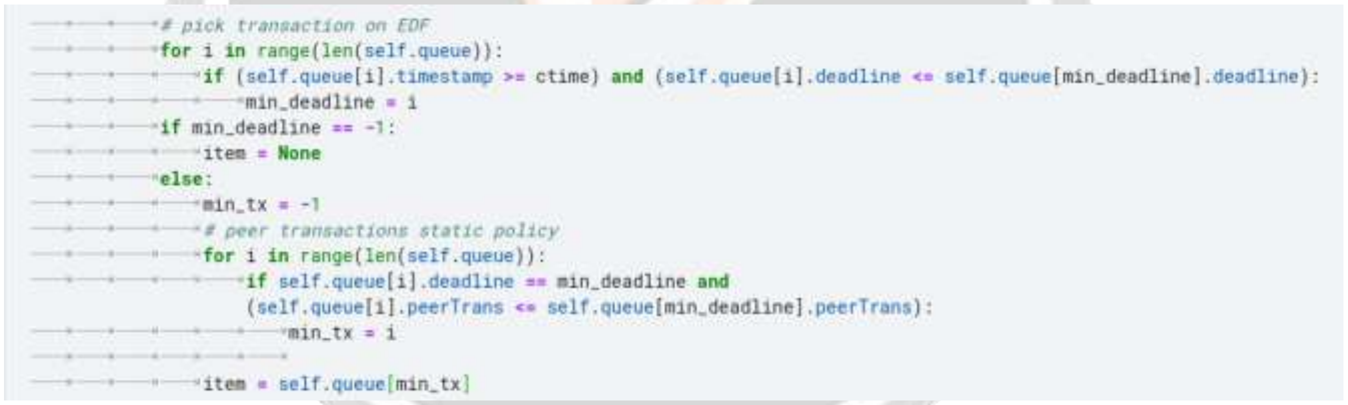


```
# pick transaction on EDF
for i in range(len(self.queue)):
    if (self.queue[i].timestamp >= ctime) and (self.queue[i].deadline <= self.queue[min_deadline].deadline):
        min_deadline = i
if min_deadline == -1:
    item = None
else:
    min_tx = -1
    # First Come first serve FCFS
    for i in range(len(self.queue)):
        if self.queue[i].deadline == min_deadline and
        (self.queue[i].__getattribute__[self.priority] <= self.queue[min_deadline].__getattribute__[self.priority]):
            min_tx = i

    item = self.queue[min_tx]
```

Fig 3.2 Selection of FCFS for transactions with the same Deadlines

In the above code snippet first, the transactions at the local sites are filtered on basis of transaction arrival time plus the CPU burst time. So that early rejection of transactions becomes easy at the code level. Then FCFS is used for the data contention if there are transactions with the same deadline. With FCFS the transaction schedular is biased towards the early arriving transaction without considering any slack time if at all transactions are generated at the same time.



```
# pick transaction on EDF
for i in range(len(self.queue)):
    if (self.queue[i].timestamp >= ctime) and (self.queue[i].deadline <= self.queue[min_deadline].deadline):
        min_deadline = i
if min_deadline == -1:
    item = None
else:
    min_tx = -1
    # peer transactions static policy
    for i in range(len(self.queue)):
        if self.queue[i].deadline == min_deadline and
        (self.queue[i].peerTrans <= self.queue[min_deadline].peerTrans):
            min_tx = i

    item = self.queue[min_tx]
```

Fig 3.2 Peer transactions policy

In the above snippet, the contention among the transactions with the same deadline to access the cpu is resolved by allotting the cpu to the transaction with the most no of peer transaction which ultimately helps in reducing the transaction miss/ rollback due to deadline miss of the transaction which has most peer executed (for the dynamic allocation). The above approach is tested with the existing approach for any performance gain.

4.PERFORMANCE EVALUATION

4.1 Overview of performance metrics

4.1.1. Miss Percentage

This is the most interesting metric as it is specifically designed for the Real-time database system as it considers the transaction value that is lost due to deadline miss. It is the ratio between total transaction values loss to the total of all transactions arriving in the system.

The formula is given by:

$$Miss\text{[illegible]}age = \left(\frac{\text{[illegible]}}{\text{[illegible]}ctionsinthesystem}\right) * 100$$

4.1.2. Hit percentage

The "Hit percentage" is regarded as a key performance indicator for evaluating the effectiveness of the DRTDBS. It is the statistic that is most frequently used to gauge how well a transaction scheduling system is working [9].

The formula is given by:

$$Hit\text{[illegible]}age = \left(\frac{\text{[illegible]}}{\text{[illegible]}ctionsinthesystem}\right) * 100$$

Above are some of the most used performance metrics in realtime distributed database systems. These metrics basically help in identifying the effectiveness of the transaction schedular in the proposed algorithm.

4.2 Result and Analysis

4.2.1 Comparison with First Come First Serve (FCFS)

On Comparing the proposed approach with the standard priority allocation policies when there is the contention of data access with the First come First Serve (FCFS) and Least Slack Time First (LSF). The approach is found to be having lower transaction rollbacks/miss rates when the arrival rate of the transactions increases with the fixed deadline and CPU burst time.

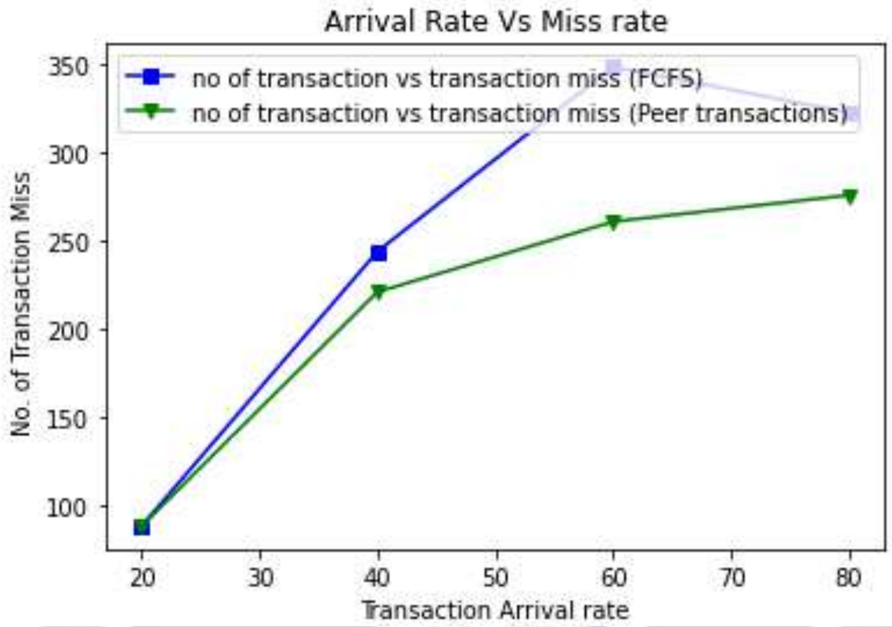
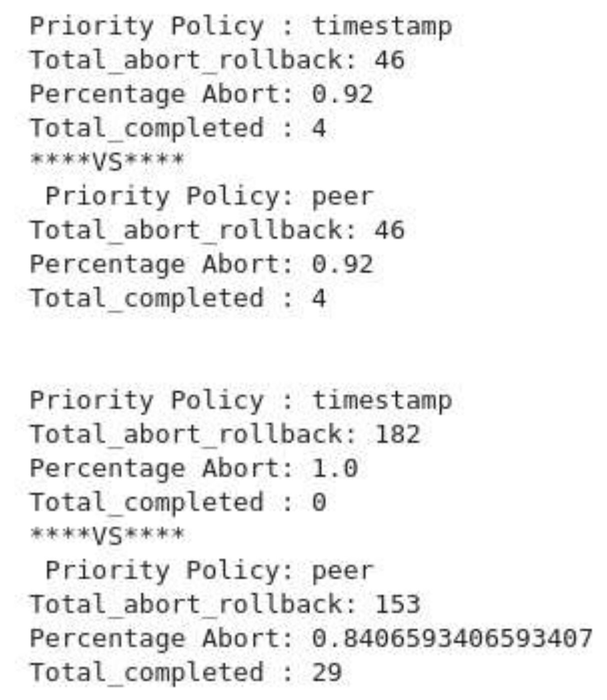


Fig 4.1 plot for transaction arrival rate Vs miss rate with FCFS

As we can clearly observe in the above plot that there is a significant decrease in the transaction miss rate in comparison with the FCFS priority scheduling. Below are the raw data generated while performing the above calculation.


```
Priority Policy : timestamp
Total_abort_rollback: 46
Percentage Abort: 0.92
Total_completed : 4
****VS****
 Priority Policy: peer
Total_abort_rollback: 46
Percentage Abort: 0.92
Total_completed : 4


Priority Policy : timestamp
Total_abort_rollback: 182
Percentage Abort: 1.0
Total_completed : 0
****VS****
 Priority Policy: peer
Total_abort_rollback: 153
Percentage Abort: 0.8406593406593407
Total_completed : 29
```

Fig 4.2 percentage miss comparison between FCFS and peer transactions

4.2.2 Comparison with Shortest Job First (SJF)

In the Shortest Job First priority scheduling approach transaction with the least CPU time is given the priority therefore it can lead to starvation of the longer transaction having a longer CPU Burst. On comparing the Peer transaction approach with SJF there is a significant decrease in transaction rollbacks/ miss is achieved thus making it the most suitable approach over SJF. Below is the plot proving the above argument.

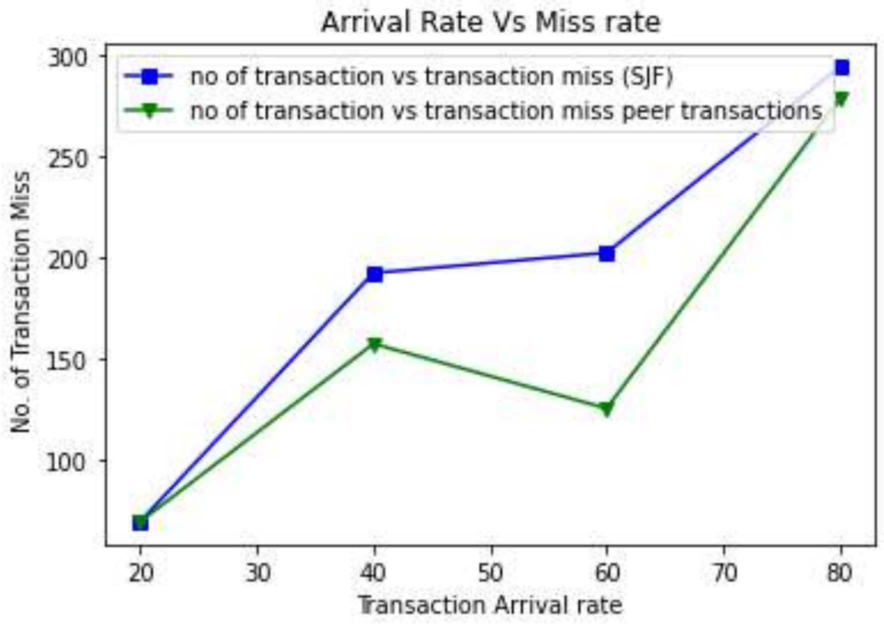
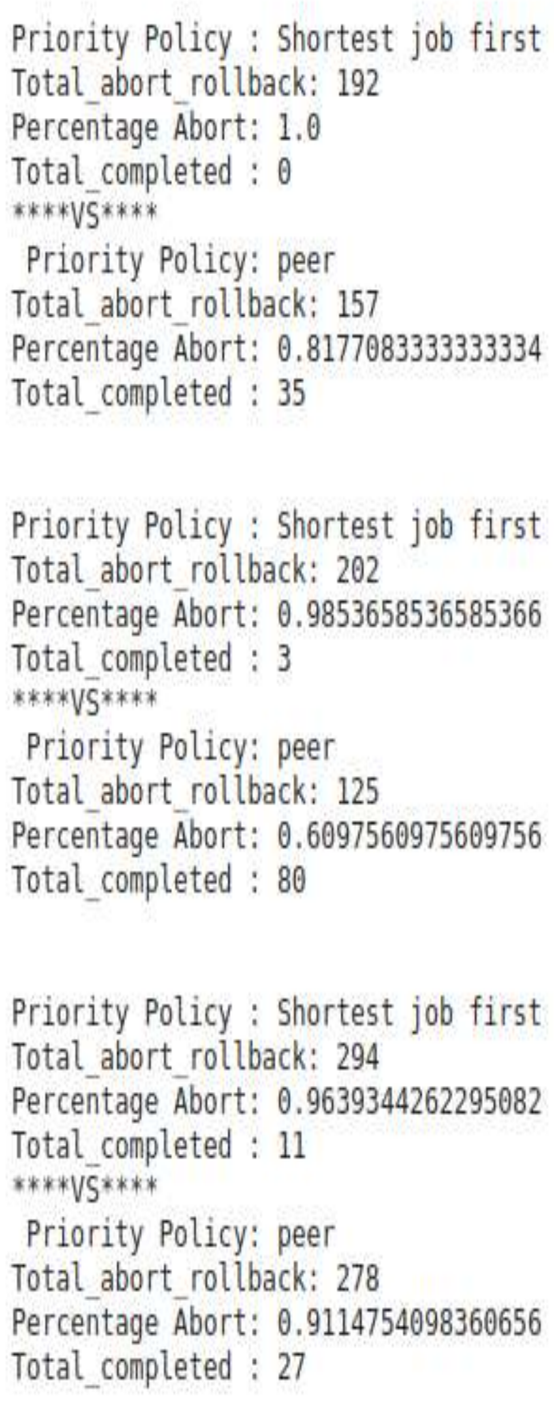


Fig 4.3 plot for transaction arrival rate Vs miss rate with SJF

With the moderate arrival of transactions into the system, the peer transaction priority allocation policy shows a tremendous improvement over the Shortest job first (SJF). The no of completed transactions in SJF lacks behind by a big margin. The below log image makes it evident.



```
Priority Policy : Shortest job first
Total_abort_rollback: 192
Percentage Abort: 1.0
Total_completed : 0
****VS****
 Priority Policy: peer
Total_abort_rollback: 157
Percentage Abort: 0.8177083333333334
Total_completed : 35


Priority Policy : Shortest job first
Total_abort_rollback: 202
Percentage Abort: 0.9853658536585366
Total_completed : 3
****VS****
 Priority Policy: peer
Total_abort_rollback: 125
Percentage Abort: 0.6097560975609756
Total_completed : 80


Priority Policy : Shortest job first
Total_abort_rollback: 294
Percentage Abort: 0.9639344262295082
Total_completed : 11
****VS****
 Priority Policy: peer
Total_abort_rollback: 278
Percentage Abort: 0.9114754098360656
Total_completed : 27
```

Fig 4.4 percentage miss comparison between SJF and peer transactions

4.2.3 Comparison with Least Slack Time First(LSF)

On Comparing the proposed approach with the standard priority allocation policies when there is the contention of System Resource among same deadline transactions with the Least Slack Time First (LSF) and Least Slack Time First (LSF). The approach is found to be having lower transaction rollbacks/miss rates when the arrival rate of the transactions increases with the fixed deadline and CPU burst time.

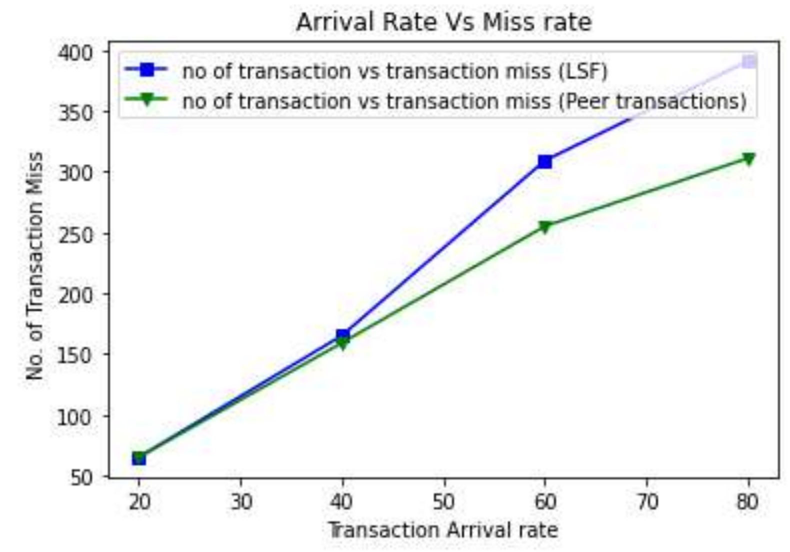


Fig 4.5 plot for transaction arrival rate Vs miss rate with LSF

So above are a few comparison studies that make the peer transaction approach an eminent approach for priority allocation over FCFS, SJF, and LSF.

5.CONCLUSION AND FUTURE WORK

Previous approaches in real-time database systems usually focused on priority assignment policies to optimize scheduling transactions and minimized the missed real-time transactions. However, most performance studies always neglect the effect of the rollback of a global real-time distributed transaction. To prevent deadline miss of the most peer committed transaction above priority allocation scheme is proposed. Although the most used priority allocation policy is the earliest deadline first (EDF), here we proposed a novel approach that is unbiased to scheduling short transactions favorably.

The approach presented in this thesis is only the starting point. Many other issues are still to be resolved and warrant further investigation. Following are some suggestions to extend this work.

1. Alternative strategies, including analytical techniques and field tests, can be utilized to assess how the proposed priority assignment policies, deadline computation method, and commit procedures will affect the functionality of DRTDBS.
2. Research is still needed to determine how communication between cohorts of the same transaction

(siblings) affects system performance as a whole.

3. In our research, we used the supposition that each site has a single-processor system. A multiprocessor environment is a logical outgrowth of our work.
4. More rigorous evaluation is needed in the real word distributed real-time database system

6.REFERENCES

1. Ramamritham, K., 1993. Real-time databases. Distributed Parallel Databases 01 (02), 199–226.
2. Ramamritham, K., Son, S.H., Dipippo, L.C., 2004. Real-time databases and data services. Real-Time Systems 28 (2–3), 179–215.
3. Shanker, U., Misra, M., Sarje, A.K., 2005. Priority assignment heuristic to cohorts executing in parallel. In: Proceedings of the 9th WSEAS International Conference on Computers, World Scientific and Engineering Academy and Society (WSEAS), pp. 01–06.
4. Hong-Ren Chen1 and Y.H. Chin, 2007. Efficient Priority Assignment Policies for Distributed Real-Time Database Systems, WSEAS International Conference on Computer Engineering and Applications, Gold Coast, Australia, January 17-19, 2007
5. Shanker, U., Misra, M., Sarje, A.K., 2006. SWIFT - A new real time commit protocol. Distributed Parallel Databases 20 (01), 29–56.
6. Shanker, U., Misra, M., Sarje, A., 2006. Some performance issues in distributed real- time database systems. Proc. VLDB Ph.D. Work, Conv. Exhib. Cent. (COEX), Seoul, Korea.
7. Shanker, U., Misra, M., Sarje, A.K., 2008. Distributed real time database systems: background and literature review. Int. J. Distributed Parallel Databases, Springer Verlag 23 (02), 127–149.
8. Shanker, U., Agarwal, N., Tiwari, S., Goel, P., Srivastava, P., 2010. ACTIVE-a real time commit protocol. Wireless Sensor Network 2 (3).
9. Sarvesh Pandey, UdaiShanker, RAPID: A real time commit protocol, Journal of King Saud University - Computer and Information Sciences, Volume 34, Issue 6, Part A, 2022, Pages 2916-2925, ISSN 1319-1578,

IJARIIE