International Journal of Advance Engineering and Research Development (IJAERD)
Volume 1,Issue 6,June 2014, e-ISSN: 2348 - 4470 , print-ISSN:2348-6406

Overload Management in Real Time Database Systems

Miss. Rutuja A. Gulhane[1] , Dr. M. S. Ali[2]
[1]Department of Computer Science and Engineering, Prof. Ram Meghe College of Engineering & Management, Email id: gulhanerutuja@gmail.com

[2]Department of Computer Science and Engineering, Prof. Ram Meghe College of Engineering & Management, Email id: softalis@hotmail.com

Abstract - As the complexity of real-time systems and application is going up, the amount of information to be handled by real-time systems increases, motivating the need for database and data service functionality. A conventional DBMS aims to maximize transaction throughput and minimize response time. However, a real-time DBMS aims to provide predictability in transaction processing and offer different quality of services. To address this need, it requires the architecture of a real-time database management system which includes a real-time control layer and provides sophisticated admission control, scheduling and overload management. In this paper, we introduce a dynamic admission control and parametrable scheduling algorithm called MOA. The MOA algorithm makes the transactions with higher importance as early arrival to be executed first in overload situations. In addition, the system architecture for real-time database system (RTDBS) has been discussed to achieve a significant performance even in overload situation.

Keywords - Real-Time Database System, Overload Management, Transaction Scheduling.

I. INTRODUCTION

As the volume of information being handeled by data-intensive applications faced with timing requirements increasing day-by-day, there is a need to apply database technology to real-time systems. The design objective of conventional databases do not support timing and temporal requirements, and therefore they are not appropriate for real-time applications. A high-performance database which is simply fast and do not have the capability of specifying and enforcing time constraints are also not suitable for real-time applications.

A real-time system must include ability to meet the time constraints which has basic specification and design correctness arguments and that correctness depends not only on the logical result of a computation, but also on the timeliness of its actions. A database system which supports a real-time application can be called ‘a real-time database system (RTDBS)’. RTDBS is a database system where at least some transactions have explicit timing constraints (such as deadline). It stores data whose operations execute with real-time response to the transactions of data-intensive applications, such as e-commerce applications, stock market, banking, internet bids, and control systems.

Although real-time transaction processing is complex because in addition to satisfying database consistency requirement, as in traditional database systems, RTDBS timing constraints are an integral part of the correctness criterion. The serializability is a criterion for correctness of concurrent transaction execution. The principles and techniques of transaction management in Database Management Systems need to be applied to real-time applications for efficient storage and manipulation of information. Furthermore, the data in such database has application-acceptable levels of logical and temporal consistency of data, which is able to handle the transactions with the ACID properties: atomicity, consistency, isolation and durability. RTDBS is the more efficient way of handling large amounts of data which gather database from the environment, process it in the context of information acquired in the past to provide timely and temporally correct response.

The scheduling of operations in a RTDBS based on two components: time-critical scheduling and concurrency control. To guarantee the isolation and atomicity properties, it requires the design of real-time concurrency and commit protocols. Traditionally concurrency control real-time protocols can be broadly classified as either pessimistic (or locking) or optimistic (or validation). Pessimistic protocols detect conflicts as soon as they occur and resolve them using blocking that may result in future inconsistencies are detected. Two Phase Locking (2PL) [1] is the most common pessimistic concurrency control protocol. Optimistic protocols such as SCC [2], [3], WAIT-50 [4] detect conflicts at transaction commit time and resolve them using rollbacks (restarts). Examples of real-time commit protocols are OPT [5] and PROMPT [6]. For a system configuration, the primary real-time performance determinants are the transaction scheduling policies for the system resources. Scheduling policies such as EDF [7], EDF-CR [8], [9] are used to assign transaction priority and it helps to schedule transactions within the scheduling queue.

Although all of these feature, the goals of conventional DBMS and real-time DBMS are different. Moreover, real-time DBMS performance objectives differ from those of conventional database system in that maximizing the number of transactions that complete before their deadlines becomes the decisive performance objective, rather than merely maximizing concurrency (or throughput).

II. RELATED WORK

In 1991, J.R. Haritsa et al. [10] have proposed a new priority assignment policy called Adaptive Earliest Deadline (AED). It stabilizes the overload performance of earliest deadline in RTDBS environment. It features a feedback control mechanism that detects overload conditions modifies transaction priority assignments accordingly. They have evaluated the Hierarchical Earliest Deadline (HED) is the extension of AED policy and was designed to handle applications where transactions may be assigned different values. They have shown the results that, both for workloads with limited spread in transaction values and for workloads with pronounced skew in transaction values, the HED policy provided the best overall performance.

In 1992, Sang H. Son et al. [11] have been proposed a new approach to real-time transaction scheduling in which there is two hybrid real-time concurrency control protocols which combine pessimistic and optimistic approaches to concurrency control in order to control blocking and aborting in a more effective manner. The two-phase locking is termed as being pessimistic and an optimistic approach is a natural alternative which schedules all operations hoping that nothing will go wrong, such as non-serializable execution. They have shown the results that over the entire operational range, optimistic schemes outperform the locking-based pessimistic protocol.

In 1993, Ozgur Ulusoy et al. [12] have designed a Real-Time Transaction Scheduling in Database Systems and concentrated on the concurrency control protocol in RTDBS. The authors have evaluated the performance of the protocols through simulations by using a detailed model of a single-site RTDBS. They have proposed a new concurrency control protocols such as Data Priority-based locking protocol (DP) and Optimistic protocol (OP) under conditions of high transaction load and high data contention with improved performance.

In 1994, Y -K. Kim et al. [13] have proposed a database server for distributed real-time systems. They have chosen the ARTS operating system kernel as the basis for the real-time database server. ARTS-RTDB supports both hard and soft real-time transactions. They have provided a flexible programming interface and standard client template to allow quick prototyping. They have incorporated the notion of imprecise computation into RTDB.

In 1994, Brad Adelberg et al. [14] have emulated soft real-time scheduling using traditional operating system schedulers. This paper focuses on soft real-time applications. The authors have addressed methods of emulating real-time scheduling algorithms on top of standard time-share schedulers and developed three strategies for priority assignment to emulate EDF and Least Slack First scheduling within a traditional multi-tasking environment. They have shown the result that the emulation algorithms are comparable in performance to the real-time algorithms and in some instances outperform them. On the other hand, in 1995, Brad Adelberg et al. [15] have applied updated streams in a soft real-time database system. In this paper, four different algorithms are used to schedule both applying updates to imported views and running user transactions perform under different assumptions about data freshness. The authors have discussed about the various properties of updates and views (including staleness) that affect the tradeoff between transaction response time and data freshness. The results have shown that the system performance was affected by the choice of algorithm and and system properties. Whereas in 1996, Brad Alderberg et al. [16] have proposed database support for efficiently maintaining derived data. The authors have propsed the forced delay recomputation algorithm. They have shown that forced delay greatly reduced the recomputation cost while only modestly diminished data timeliness across a wide range of parameter values. It can exploit update locality to improve both data freshness and transaction response time.

In 1996, B. Adelberg et al. [17] have proposed the Stanford Real-time Information Processor (STRIP) which is a soft real-time database system and was built for the UNIX operating system. It is a database designed for heterogeneous environments and provides support for value function scheduling and for temporal constraints on data. Its goals include high performance and ability to share data in open systems. It does not support any notion of performance guarantees or hard real-time constraints and hence cannot be used for the applications we are envisioning in our work. Whereas in 1996, S. F. Andler et al. [18] have proposed Active Real-Time Database System (DeeDS) which is distributed and supports both hard and soft transactions. It is an event-triggered real-time database system using dynamic scheduling of sets of transactions. The reactive behavior is modeled using (event-condition-action) ECA rules. In the current prototype, they do not support temporal constraints of data and multimedia information.

In 1996, J. Taina et al. [19] have a RODAIN, a real-time object-oriented database system for telecommunications. It supports firm real-time database system. In 1996, A. Datta et al. [20] have proposed a multiclass transaction scheduling and overload management in firm real-time database systems. The authors have introduced a dynamic admission control policy and priority based scheduling policy for disk resident RTDBS called Adaptive Access Parameter (AAP) which is a scheduling mechanism for multiclass transactions in RTDBS. The admission control policy of AAP serves dual purposes – overload management as well as bias control towards particular transaction classes. And it leads to dramatic performance improvement both in terms of reducing transaction misses as well as fairness. Whereas in 1996, A. Bestavros et al. [21] have proposed an admission control paradigm for value-cognizant Real-Time Database in which a transaction is submitted to the system as a pair of processes: a primary task and a compensating task. The authors have considered only hard-deadline transactions. The goal of the admission control and scheduling protocol employed in the system is to maximize system profit dynamically.

In 1999, J. A. Stankovic et al. [22] have proposed BeeHive: Global Multimedia Database Support for dependable, real-time applications. It is a virtual database where the data in the database can be located in multiple locations. It includes features along real-time interface, fault tolerance interface, quality of service for audio and video, and security dimensions. The authors have achieved a high degree in the usability of a virtual database system where a user can obtain secure and timely access to time valid data even in the presence of faults. Whereas In 1999, J. Hansson et al. [23] have proposed value-driven multi-class Overload Management. They have presented a value-driven overload management algorithm with a bias control mechanism, called OR-ULD/BC, which attempts to bias the execution among transaction classes such that minimum completion ratio constraints are satisfied. They have shown the performance analysi of OR-ULD/BC enforces transaction class timeliness requirements within a specified operational envelope.

In 2002, K. -D. Kang et al. [24] have proposed Service Differentiation in Real-Time Main Memory Database to execute transactions in temporally consistent data. Based on the importance of the transactions, they have been classified into several services. Different degree of deadline miss ratio and feedback control has been applied among the service classes to support the miss ratio and freshness guarantees. They have shown the results that the approach can provide the specified QoS when the baseline approaches fail to support the miss ratio and/or freshness guarantees in the presence of unpredictable workloads and access patterns. The target performance is achieved by dynamically adapting the system behaviour based on the current performance error measured by the monitor.

In 2002, Buttazzo et al. [25] have presented the work of the periodicity of a set of tasks in response to load variations is changed. Aperiodic tasks are not considered in this model. A feedforward scheduling algorithm for optimizing the performance of a set of control tasks is presented A. Cervin et al. in [26]. In order to keep the utilization at a certain reference, the rates of the control tasks are adjusted. Further, they have used a feedforward structure to make the feedback scheduler more reactive to change in the workload. The approaches above do not address imprecise computation. A feedback control scheduling framework has been presented by Parekh et al. [27]. Each task has several QoS levels giving results of varying quality. Utilization and miss percentage are monitored and control by changing the QoS of a set of tasks. Experiment shows that their algorithms provide performance guarantees even when execution time varies considerably from the estimate. However, they do not address QoS management of real-time data services.

In 2003, M. Amirijoo et al. [28] have addressed the question of QoS in terms of imprecision. Further, in 2004, M. Amirijoo et al. [29], they have proposed an approach called RDS, for managing the performance of multi-class real-time data services. In this approach, the transactions are classified according to their importance, and a QoS specification can be chosen for each importance level. RDS is a major improvement of the previous approaches, in which the QoS specification increased with the importance level.

In 2006, M. Amirijoo et al. [30] have proposed an approach for managing the quality of service of real-time databases that provide imprecise and differentiated services, and that operate in unpredictable environments. Transactions are classified into service classes according to their level of importance which are further classified into subclasses based on their quality of service requirements. In this way transactions are explicitly differentiated according to their importance and quality of service requests. The authors have shown that performance evaluation during overloads the most important transactions are guaranteed to meet their deadlines and that reliable quality of service is provided even in the face of varying load.

In 2006, M. Amirijoo et al. [31] have proposed a framework for QoS specification and management. It has been developed in real-time databases supporting imprecise computation. The architecture based on feedback control scheduling, and a set of algorithms implementing different policies and behaviors. The approach gives a robust and controlled behavior of real-time databases, even for transient overloads and with inaccurate runtime estimates of the transactions. Further, performance experiments show that the proposed algorithms outperform a set of baseline algorithms, where transactions are scheduled with EDF and feedback control.

In 2006, Leila Baccouche [32] has proposed Multi-Class Overload Architecture (MOA) for Real-time Database Systems: Framework and Algorithms. The architecture includes a real-time control layer which provides sophisticated admission control, scheduling and overload management. They have considered an importance classes that have been categorized as high, medium and low level of transaction workload. They have evaluated the H/M/L dispatching algorithm with the developed simulator and they have shown the results in terms of percentage. The proposed algorithm has made transactions with higher importance have small miss deadline percentage.

In 2012, R. K. Mishra et al. [33] have proposed a novel protocol RCCOS (Replica Concurrency-Control for Overloaded Systems) for overload management and admission control in real-time distributed database systems. It can be easily integrated in current systems to handle overload processors without altering the database consistency which is the main objective of DRTDBSs. Our protocol RCCOS (Replica Concurrency-Control for Overloaded Systems) augments the protocol MIRROR, a concurrency control protocol designed for firm-deadline applications operating on replicated real-time databases in order to manage efficiently transactions when the distributed system is overloaded.

III. ANALYTICAL MODEL FOR REAL-TIME TRANSACTIONS

Real-Time transactions can be distinguished based on the effect of missing their deadlines. They can be grouped into three categories: transactions that have hard deadlines, soft deadlines and firm deadlines.

A. Hard Deadline Transactions

A hard deadline transaction has hard timing constraints that must absolutely be met. The transactions are those which may result in a catastrophe if their deadlines are missed. There is no scope to miss its deadline in hard deadline transactions. Missing a hard-deadline can result in catastrophic consequences. Such systems are known as safety-critical. And we can view such tardy transactions as carrying a large negative value to the system. Typically safety-critical applications (e.g. nuclear power plant control) that respond to life or environment-threatening emergency situations can be classified in this category. Thus, the design of a hard real-time system requires that a number of performance and reliability trade-off issues to be carefully evaluated.

B. Soft Deadline Transactions

A soft real-time application is characterized by a soft deadline whose adherence is desirable, although not critical, for the functioning of the system. Soft deadline transactions have some diminished value to the system even if they complete after their deadlines have expired. Such systems tolerate imprecise results and less quality of service. There is scope to miss its deadline in soft deadline transactions which may lead to performance degradation but do not entail catastrophic results. That is, missing a soft-deadline does not cause a system failure or compromises the system's integrity.

C. Firm Deadline Transactions

A firm real-time task, like a soft real-time task, is characterized by a firm deadline whose adherence is desirable, although not critical, for the functioning of the system. Firm deadline transactions, have no value once their deadlines expire, completing them is of no utility and may even be harmful to the system. Thus, tardy transactions are permanently aborted (killed) and discarded as soon as its deadline is missed. Firm deadlines should be met but may be missed occasionally (e.g. during transient overloads).

Real-time database transactions are usually in either firm deadline or soft deadline class. In addition to these timing constraints, as there is a need to maintain consistency between the actual state of the environment and the state as reflected by the contents of the database, it needs to extend the timing correctness requirements in a RTDBS. This leads to the temporal consistency and its two components: absolute consistency between the state of the environment and its reflection in the database, and relative consistency among the reflected data used to derive other data.

The transaction temporal consistency requirements are required for real-time transactions. A transaction τ_i is characterized by the following attributes:

TABLE 1. Attributes of Transaction τ_i

Attributes	**Meaning**	**Description**
r_i	Ready time	The time in which the transaction arrives to the system.
d_i	Relative deadline	It indicates requirements to complete transaction before the instant *di*.
we_i	Worst case execution time	The execution time of a transaction is data dependant.
re_i	Remaining execution time	It represents the time to complete the transaction.
st_i	Slack time of τ_i	It represents maximum time of transaction can be delayed and still satisfy its deadline. $st_i = d_i - r_i - we_i$.
imp_i	Importance of τ_i	It indicates how much it is critical to the system that the transaction meets its deadline.

Transaction type can have two values local or distributed. A local transaction is created by the local node. A distributed transaction is a sub transaction initiated by a distant node and sent to the current node to be executed.

Transaction can be either periodic or aperiodic. Periodic transactions tend to have *hard* deadlines, characterized by their *period*(*s*) and their required execution time per period. They need to update data frequently. Let *pi* be the invocation period. A periodic transaction is executed each *pi* instants. Usually *pi*=*di*. Aperiodic transactions tend to have soft deadlines, characterized by unknown arrival instants.

IV. SYSTEM ARCHITECTURE

The system architecture is the Multi-Class Overload Architecture (MOA) which allows executing transactions under overload situation in real-time database systems. The present work proposes a framework for dynamically resolving transient overloads in real-time database systems, yielding predictable behavior and graceful performance degradation during transient overloads. The framework consists of a strategy and a scheduling architecture. Features of the system architecture are:

- It controls transactions upon their arrival.
- It reduces miss deadline, function of transaction classes.

- Overload situations detection and resolution.

MOA is a real-time control layer that should be at end integrated to the database system. The system for overload management consists of three components: a transaction controller, a transaction scheduler and a transaction manager. Figure 1 presents the system architecture of MOA.

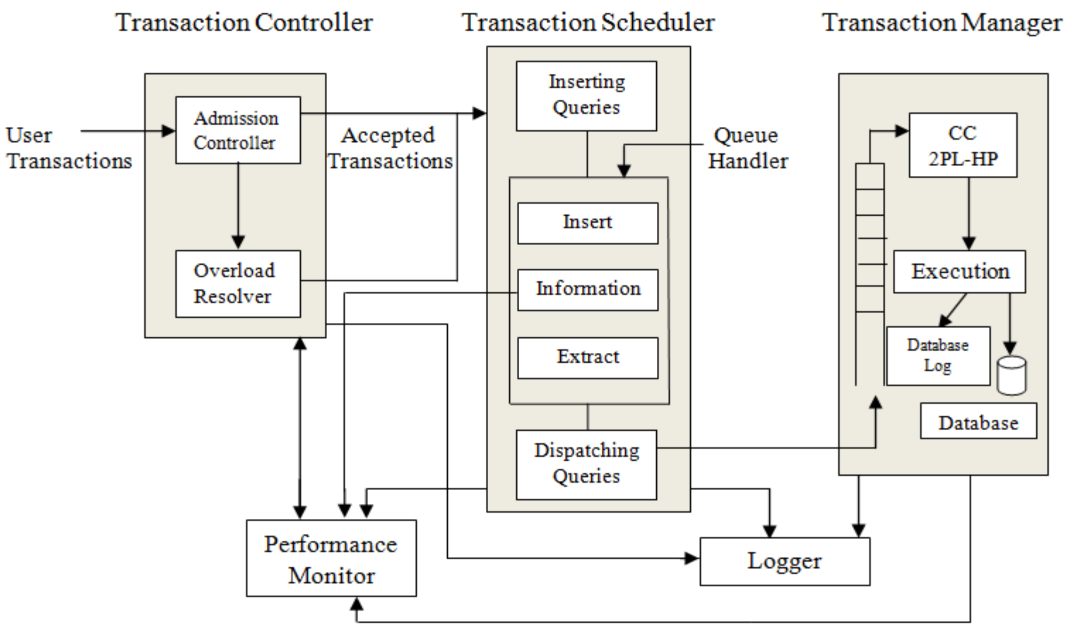


Figure 1. System architecture (MOA)

4.1. The Transaction Controller

The transaction controller requires the admission controller to test the admission of transactions and the overload resolver to solve overload situations.

A. The Admission Controller

When a transaction is submitted to the system, an admission controller is employed to decide whether to admit or reject that transaction. The admission controller controls the admission of new transactions and tests the acceptability upon their arrival. The controller makes a decision which depends on the state of the current system and attributes of transaction.

The schedule is not computed at that time, the controller simply checks when an acceptance condition is satisfied. If the test succeeds, this means that the transaction scheduler will be able to find a feasible schedule including the new transaction guarantying that each transaction meets its deadline.

B. The Overload Resolver

Depending on the characteristics of the transaction, the overload resolver component may be invoked in order to find a solution where the rejected transaction by the admission controller is accepted. The resolver computes the amount of processing time that needs to be released in order to resolve transient overloads, and initializes a negotiation of the requirements of the admitted transactions and the new transaction.

That means if the overload situation occurs, overload resolver resolves the situation by tracking the time needed to execute the current query and then performing more or less queries based on the load of the database system.

4.2. The Transaction Scheduler

The transaction scheduler requires queue handler for inserting and dispatching queries. Transactions accepted by the admission controller are sent to the inserting module. This later inserts them in the associated queues according to the priorities. However, in the system, the priorities are assigned to the transactions according to the First Come First Serve (FCFS) scheduling policy with serial execution. This policy assigns the highest priority to the transaction with the earliest release time.

To address this need, a novel parametrable priority based scheduling algorithm called MOA is applied by the dispatcher which is specific to the transaction multi-class model. The MOA algorithm makes the transactions with higher importance as early arrival to be executed first in overload situations. Dispatching queries are responsible of transaction extraction from the queue handler. When transaction execution is finished, the transaction scheduler is executed in order to select a transaction for execution and information regarding any gained processing time is reported by the dispatcher back to the scheduler. The dispatcher applies a novel parametrable priority based scheduling algorithm called MOA which is specific to transaction multi-class model.

A. Queue Handler

Queue handler provides the basic for the queues handling and the maintenance of information on the queues. It consists of three modules: one for transaction insertion at the position specified by the transaction scheduler, one for the extraction according to the algorithm applied by the dispatcher and a last module for the maintenance of information on the queues.

Upon each transaction arrival, the inserting algorithm based on FCFS scheduling policy is executed to insert the transaction in its appropriate queue. The MOA algorithm is used by the dispatcher to extract a transaction from a specified position. Besides, the information module computes all the information needed by the dispatching algorithm for example the total number of transactions in queue, the frequency, cost and time require to calculate the total score of each of the transaction.

B. MOA Algorithm

The MOA architecture is demonstrated in two ways: manually enter data to insert, update and delete command and by applying MOA algorithm for query file containing select statement.

When we deal with overloads, we have to manage them in order to guarantee real-time properties of transactions. To manage the overload, set the threshold to some value and manually enter the data by the use of insert, update and delete command. Start the query thread to execute the queries and update the database. Apply fuzzy logic to get the new number of record for processing on the basis of time. Check the entry for new transactions upon their arrival for processing by considering the previous and current time. If the previous time is less than current time, decrement the threshold by 1 and if previous time is greater than current time, increment the threshold by 1. That means based on the load of the database, MOA has reduced or increased the number of processing queries in the system. The controller makes a decision based on the current system state and transaction attributes to check that all queries get executed by MOA.

MOA is also demonstrated by applying MOA algorithm to the chosen query file containing select statement. The system applies First Come First Serve (FCFS) scheduling policy for scheduling the transactions and finds the frequency, cost and time of each of the query.

- To find frequency of the query:

Step 1: Read the contents of the query file line by line and get the count
Step 2: Compare two resultsets rsCurrent (cnt) and rsInner (icnt)
Step 3: Check if the same query occurs in both resultsets, then increment the frequency by 1.

- To find cost of the query

Step 1: Check if the number of records is equal in ResultSet to find the storage cost
Step 2: Get the count of number of columns in ResultSet
Step 3: Check all the columns in the record to get the string result in bytes
Step 4: Increment the records to check all the records present in the table and get the length of the records in Resultset.

- To find execution time of the query

Step 1: Get the count of list of queries between start date and end date
Step 2: Subtract the start time from the end time to get the current execution time for each query.

After calculating the frequency, cost and time for each of the query in the chosen file, the system needs to find the query threshold value for maintaining the view on the database. Query threshold can be calculated on the basis of the total score of each of the query. And for calculating the total score (sq), there is need to find the values of normalized frequency (nf), normalized time (nt) and normalized storage (ns).

- The formula for calculating nf, nt and ns:

  nf = f / max_f;
  nt = t / max_t;
  ns = s / max_s;

- The formula for calculating total score:

  sq = (alpha * nf) + (beta * (1 - ns)) + (delta * (1 - nt));

- The formula for calculating threshold:

  Threshold = Threshold + sq;
  Threshold = Threshold / Total no. of queries in the file

After calculating the query threshold, check which queries have total score value more than threshold, update the view for those queries. And finally get the result in Percentile Feedback. By getting the result in percentile, we can conclude if we need to or not need to make the view for the query, thereby making it easier to fetch the data from the database.

- The formula for calculating Percentile Feedback:

$$\text{Percentile Feedback} = \frac{\text{No. of maintained view} * 100}{\text{Total no. of executed queries}}$$

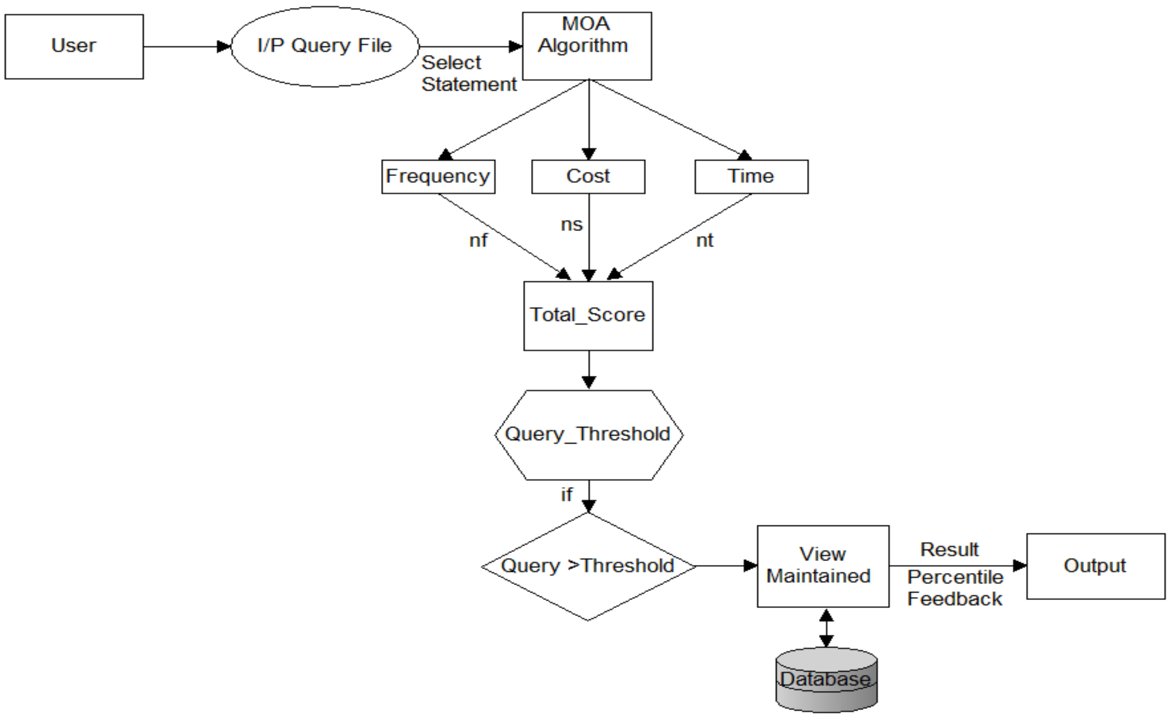


Figure 2. Implementation of MOA Algorithm

Figure 2 shows the implementation of MOA algorithm. The MOA algorithm has been created to measure the performance of the system in percentile. It will help to achieve a significant performance even in overload situation as the percentile gives the relative results.

4.3. The Transaction Manager

The transaction manager represents a conventional database engine. It is responsible for transaction execution, consistency check, concurrence control and native transaction logging operations. Transactions executed by the MOA algorithm are inserted in a queue called TM queue which is used by transaction manager to execute transactions. For each transaction extracted from TM queue, the resource set is checked and once all the required resources obtained, the transaction begins its execution.

The concurrency control algorithm applied by the transaction manager is two phase lock high priority (2PL-HP) which is free from inversion priority. A concurrency controller (CC) checks and manages the concurrency of the user transactions. The objective of a concurrency control algorithm is to make sure operations issued by transactions are executed in an order such that the results produced by the involved transactions are consistent.

4.4. The Performance Monitor and Logger

The performance monitor measures the state of the system in terms of percentile feedback. Each time a message is sent to the logger, a message is sent to the performance monitor to maintain statistics on the metrics used by the system. Examples of useful statistics are frequency, cost and time (functions of MOA parameters).

The logger represents the journalizing module of the real-time database system. It receives the most relevant information to preserve on behalf of the various modules and it registers them in a log. For example, each time total score of a transaction exceeds the threshold value, this information is written in the log. Log information is useful to analyze overload situations.

V. PROPOSED WORK

This works aims at the overload management in real-time database systems. Real-time systems have a finite set of resources, and hence, have a finite processing capacity. Due to the requirement of guaranteeing temporal behavior with finite processing capacity, the real-time system must be designed to handle peak load situations generated by the environment. The peak load is determined both by the transaction workload and by the event load imposed on the system, since event-triggered systems are prone to event-showers.

In short, the scheduling problem can best be described as how to dynamically schedule transaction workload and how to gracefully degrade system performance during overloads. The primary criterion is that any schedule must enforce the timeliness of critical transactions requesting resources. Secondly, any schedule should also attempt to ensure that all robustness requirements of non-critical transactions are satisfied. Hence, the primary focus is on the transient overload situations since these are, although hopefully infrequent, the worst threat to the primary criterion.

The detailed internal working of the system for the purpose of overload management as in multi-class overload architecture (MOA) has been explained in the Figure 3, using the flow diagram of the proposed system and the implementation steps.

Implementation steps for MOA:

➢ MOA is demonstrated in two ways:

- Manually entering data to insert, update and delete command
- Applying MOA for select query file

➢ Manually entering data for insert, update and delete command

Step 1: Set the Query Threshold to some value (for e.g. 5)

Step 2: Start the query thread to execute the queries and update the database

Step 3: Apply Fuzzy Logic to get the new number of records for processing

Step 4: Check that all queries get executed by MOA on the basis of execution time

➢ Applying MOA for select query file

Step 1: Choose the query file which contains select statement from the folder for execution

Step 2: Read the queries and find the frequency, cost and time of each of the query

Step 3: Now find the values of Normalized Frequency, Cost and Time for calculating total score of each of the query

Step 4: Find the threshold sum and division to get the query threshold

Step 5: Check which queries have value more than threshold, update the view for those queries

Step 6: Maintain the view to get result of Percentile Feedback

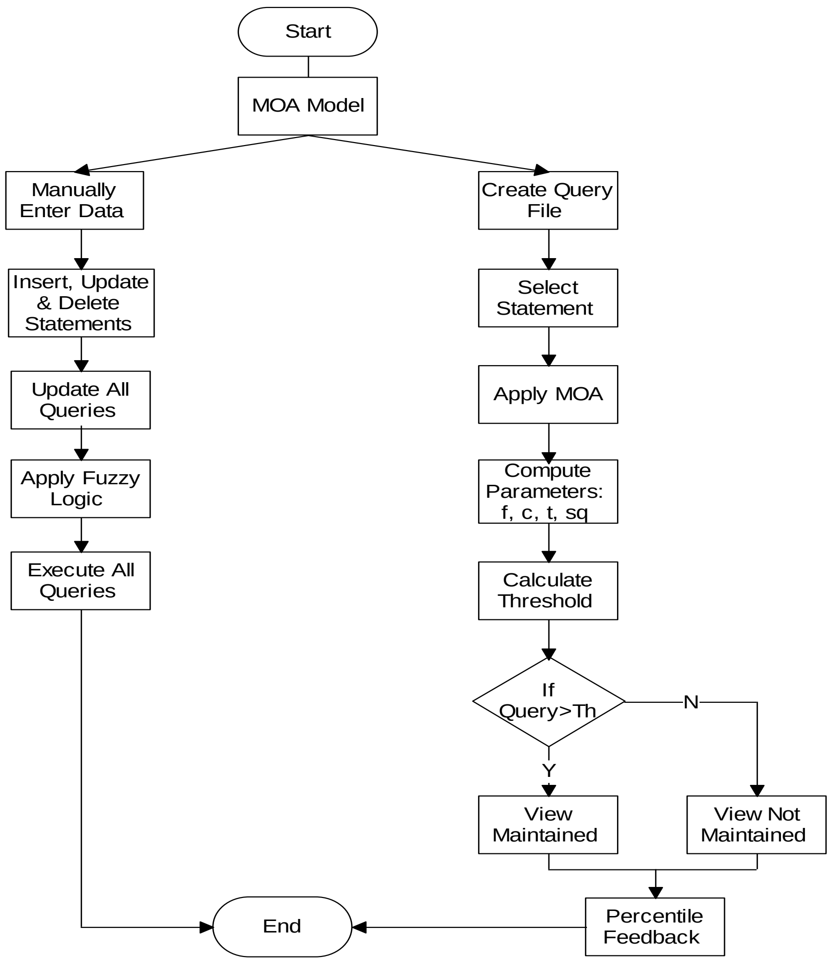


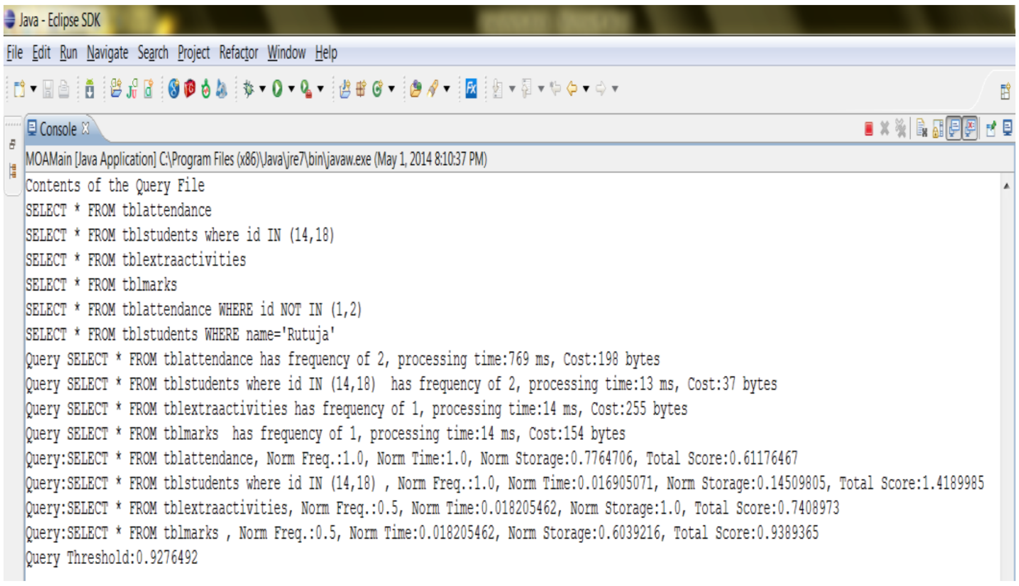
Figure 3. System Flow Diagram

VI. EXPERIMENTAL RESULTS

Initially, the query threshold is set to some value (e.g. 3). The user is allowed to insert, update and delete data from the database up to the threshold value. For example, user inserts new record into the tblstudents, user updates the record into the tblmarks and user deletes one record from tblattendance. The threshold value is set to 3 and that is why stopping the further processing of the transactions. All the three user transactions are added to the queue. As the threshold value of query is reached, now it starts updating the database. Then all those transactions which are added to the queue get executed. By tracking the time needed to execute the current queries and then performing more or less queries based on the load of the database. As the previous time needed o execute the queries is less than the current time needed to execute the queries, MOA has reduced the number of processing queries to 2. In this way all queries are get executed by MOA.

Now apply fuzzy logic to get the new number of records for further processing. And repeat the above process to get the new transactions executed by MOA. As MOA has reduced the number of processing queries to 2; hence, user is allowed to do 2 transactions into the database. All the two user transactions are added to the queue. Here, threshold value is considered to be the 2. As the threshold value of query is reached, it starts updating the database. Then all those transactions which are added to the queue get executed. By tracking the time needed between previous and current time, more or less queries based on the load of the database are performed. As the previous time is greater than the current time, MOA has increased the number of processing queries to 3. In this way all new queries are get executed by MOA.

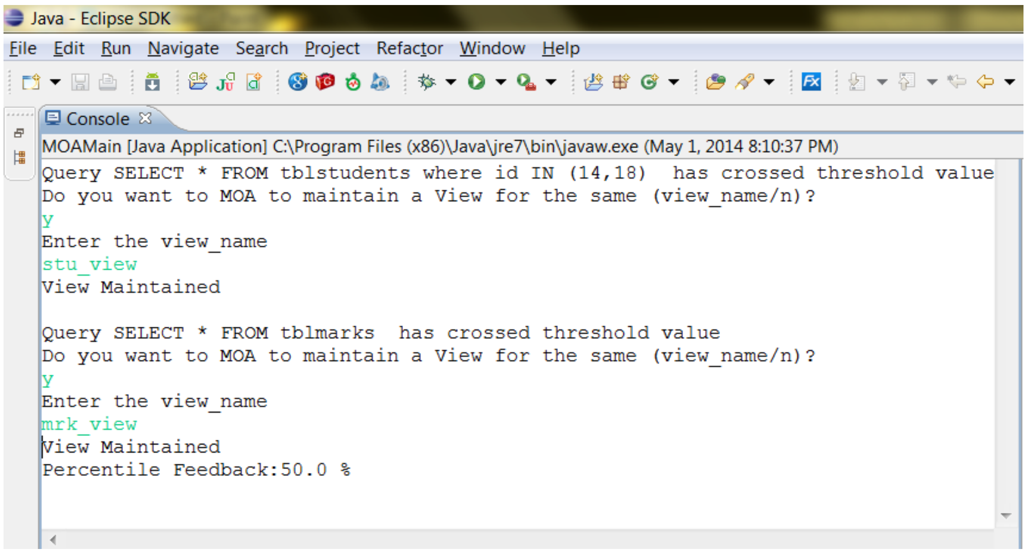
Choose a file containing select statement by applying MOA. All the queries get evaluated based on the three parameters frequency (f) of the query, the time (t) needed to execute the query and the memory requirement (c) of the query. After that normalized frequency (nf), normalized time (nt) and normalized cost (ns) are computed to calculate the total score of each of the query. Now the threshold value is calculated by considering total score of each of the query as depicted in Figure 4.



```
MOAMain [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (May 1, 2014 8:10:37 PM)
Contents of the Query File
SELECT * FROM tblattendance
SELECT * FROM tblstudents where id IN (14,18)
SELECT * FROM tblextraactivities
SELECT * FROM tblmarks
SELECT * FROM tblattendance WHERE id NOT IN (1,2)
SELECT * FROM tblstudents WHERE name='Rutuja'
Query SELECT * FROM tblattendance has frequency of 2, processing time:769 ms, Cost:198 bytes
Query SELECT * FROM tblstudents where id IN (14,18)  has frequency of 2, processing time:13 ms, Cost:37 bytes
Query SELECT * FROM tblextraactivities has frequency of 1, processing time:14 ms, Cost:255 bytes
Query SELECT * FROM tblmarks  has frequency of 1, processing time:14 ms, Cost:154 bytes
Query:SELECT * FROM tblattendance, Norm Freq.:1.0, Norm Time:1.0, Norm Storage:0.7764706, Total Score:0.61176467
Query:SELECT * FROM tblstudents where id IN (14,18) , Norm Freq.:1.0, Norm Time:0.016905071, Norm Storage:0.14509805, Total Score:1.4189985
Query:SELECT * FROM tblextraactivities, Norm Freq.:0.5, Norm Time:0.018205462, Norm Storage:1.0, Total Score:0.7408973
Query:SELECT * FROM tblmarks , Norm Freq.:0.5, Norm Time:0.018205462, Norm Storage:0.6039216, Total Score:0.9389365
Query Threshold:0.9276492
```

Figure 4: Transactions executed by applying MOA

As can be seen from Figure 4, after calculating the threshold value, the user is asked to maintain the view for those queries which have total score greater than threshold value. The user has selected the file containing six queries and from those queries, only four queries are executed. From the four executed queries, two views are maintained on tblstudents and tblmarks, as they have crossed the threshold value. View is maintained into the database to minimize the processing time and for faster execution later. After maintaining the view, the result of percentile feedback is evaluated as shown in Figure 5.



```
MOAMain [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (May 1, 2014 8:10:37 PM)
Query SELECT * FROM tblstudents where id IN (14,18)  has crossed threshold value
Do you want to MOA to maintain a View for the same (view_name/n)?
y
Enter the view_name
stu_view
View Maintained

Query SELECT * FROM tblmarks  has crossed threshold value
Do you want to MOA to maintain a View for the same (view_name/n)?
y
Enter the view_name
mrk_view
View Maintained
Percentile Feedback:50.0 %
```

Figure 5: Output Result (Percentile Feedback)

6.1. Discussion

The objective of the experiments is to show that the presented algorithm can provide guarantee to the overload management in real-time database system. For this reason, the behavior of the algorithm based on the set of performance metric is evaluated. In order to evaluate the performance of the system the standard parameters considered are the query threshold, number of views maintain, percentile feedback as compare to percentage results. This section focuses on experimentation carried out in proposed work.

➢ **Result Analysis**

Table 2. Experimental Results

Transactions	**Query Threshold**	**View Maintained**	**Percentile Feedback**	**Percentage Results**
Queries – 6 Executed – 4	0.92	1	25	16.66
Queries – 10 Executed – 7	0.82	5	71.42	50
Queries – 15 Executed – 13	0.85	8	61.53	53.33

From the above result analysis, it is seen that if 6 transactions are taken and from 6 only 4 transaction are executed then the query threshold value is 0.92. One view is maintained from the 4 executed transactions and the percentile feedback is 25. Figure 6 shows the graphical representation of given dataset.

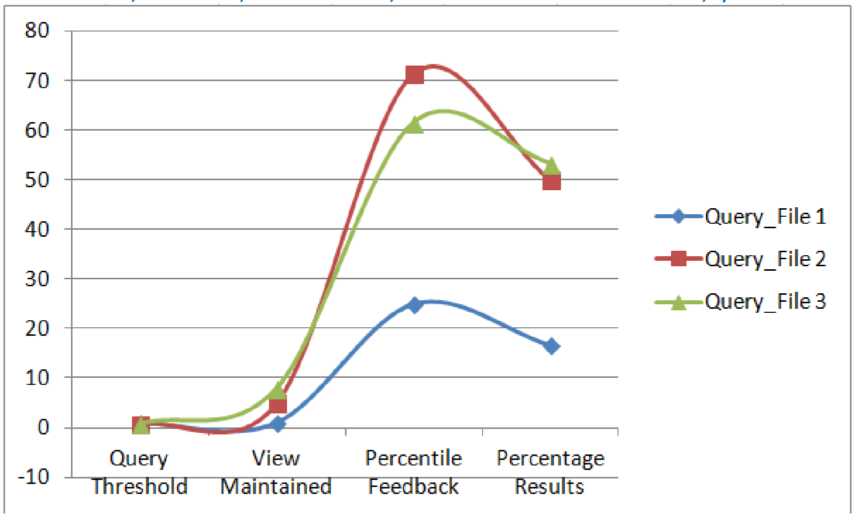


Figure 6. Graphical Representation of given dataset

Percentage is simply a representation of a proportion out of 100 which gives absolute results, whereas a percentile is a statistical measure of distribution which gives relative results. For a given set of data, it is the level below which a certain percentage of data falls. Relative results are very important when considering the dataset in real-time database systems. By getting the results in percentile, we can conclude if we either need or not need to make the view for the transaction, thereby making it easier to fetch the data from the database. As the results are evaluated in terms of percentile, views will be maintained for given set of transactions. Hence, as the views need less time for processing as compared to the actual table, the system performance increases.

VII. CONCLUSION

The proposed approach is for managing the overload in real-time database system. The focus is on the design of a model which defines a transactional behaviour adapted to the context of real time. The behavioural and the structural specifications of this model involving several execution modes for real-time transactions is an efficient solution to manage overload situations. The present work proposes a framework for dynamically resolving transient overloads in real-time database systems, yielding predictable behavior and graceful performance degradation during transient overloads. The framework consists of a strategy and a scheduling architecture.

- The multi-class overload architecture (MOA) has been discussed for overload management in real-time database system.
- MOA mainly consists of set of modules which act together in order to guarantee real-time properties of transactions that is predictability, overload resolution and service differentiation.
- The transaction scheduler applies dispatching algorithm called MOA which is used to schedule transaction in real-time database systems.
- MOA is suitable for multi-class scheduling and is starvation free.
- The results are evaluated in terms of percentile as the percentile gives the relative result which is important in real-time database systems.

VIII. FUTURE SCOPE

The MOA architecture will be extended in the near future to manage quality of service for both data and transactions. Hence, a database developer will be able to specify the requirements on the behaviour of the database even in the presence of unpredictable workloads and overloads. Values measured by the monitor will come to supply a quality of service manager which on the basis of the system utilization and the values specified by the database administrator, will detect a possible overload and will make the decision to reduce the transaction admission or to suspend it until the system turns over in a stable state.

- We can extend the system to integrate heterogeneous databases.
- We can also add templates which contain predefined configurations used by consumers to incorporate cloud services. The templates also include predefined database, security configurations and load balancing.
- We can improve performance feedback by making the database in active real-time system (ARTS) where the complete information about the workload must be known a priori.

REFERENCES

[1] J.R Haritsa, M. J. Carey, and M. Livny, "Data access scheduling in firm real-time database systems", In Journal of RTS, Volume 4, Issue 3, pp. 203-241, 1992.
[2] A. Bestavros, and S. Braoudakis, "Timeliness via speculation for real-time databases", In Proceedings 14th IEEE RTS Symposium (RTSS'96), San Juan, Puerto Rico, 1996.
[3] S. Braoudakis, "Concurrency control protocols for real-time databases", PhD dissertation, Computer Science Department, Boston University, USA, 1995.
[4] J.R Haritsa, M. J. Carey, and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control", In Proceedings of the Real-Time Systems Symposium, pp. 94-103, Dec. 1990.
[5] R. Gupta, J. Haritsa, K. Ramamritham, and S. Seshadri, "Commit processing in distributed real-time database systems", In Proceedings of the 17th IEEE Real-Time Systems Symposium, pp. 220-229, Washington D.C, Dec. 1996.
[6] R. Jayant, J. Haritsa, K. Ramamritham and R. Gupta, "The PROMPT Real_Time Commit Protocol", IEEE Transactions on Parallel and Distributed Systems, Volume 11, Issue 2, pp. 160-181, Feb. 2000.
[7] C. L. Liu, and J. Layland, "Scheduling algorithms for multiprogramming in hard real-time environments", In Journal of the ACM, 20(1): pp. 46-61, Jan. 1973.
[8] R. Abbot, and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation", In Proceedings of the 14th International Conference on Very Large Database Systems, Los Angeles, California, USA, Aug. 1988.
[9] R. Abbot, and H. Garcia-Molina, "Scheduling Real-Time Transactions with Disk Resident Data", In Proceedings of the 15th International Conference on Very Large Database systems, pp. 385–396, Amsterdam, The Netherlands, Aug. 1989.
[10] J.R Haritsa, M. J. Carey, and M. Livny, "Earliest-Deadline Scheduling for Real-Time Database Systems", In Proceedings of the 12th IEEE Real-Time Systems Symposium, San Antonio, Texas, Dec. 1991.
[11] Sang H. Son and Juhnyoung Lee, "A New Approach to Real-Time Transaction Scheduling", In Proceedings of the IEEE Conference on Real-Time Systems, pp. 177-182, Jun. 1992.
[12] Ozgur Ulusoy and Geneva G. Belford, "Real-Time Transaction Scheduling in Database System", In Proceedings of the ACM Journal on Information Systems, Volume 18, Issue 9, pp. 559-580, Dec 1993.
[13] Y-K. Kim, M. R. Lehr, D. W. George, and S. H. Song, "A database server for distributed real-time systems: Issues and experiences", In Proceedings of the Second IEEE Workshop on Parallel and Distributed Real-Time Systems, 1994.
[14] B. Adelberg, H. Garcia-Molina, and B. Kao, "Emulating Soft Real-Time Scheduling Using Traditional Operating System Schedulers", In IEEE Real-Time Systems Symposium, April 1994.
[15] B. Adelberg, H. Garcia-Molina, and B. Kao, "Applying Update Streams in a Soft Real-Time Database System", In Proceedings of the ACM Sigmod Record, 1995.
[16] B. Adelberg, B. Kao and H. Garcia-Molina, "Database Support for Efficiently Maintaining Derived Data", In EDBT Proceedings, 1996.
[17] B. Adelberg, B. Kao, and H. Garcia-Molina, "Overview of the STanford Real-Time Information Processor (STRIP)", In SIGMOD Record (ACM Special Interest Group on Management of Data), Volume 25, Issue 1, pp. 34-37, Mar 1996.
[18] S. F. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Eftring, "DeeDS Towards a Distributed and Active Real-Time Database System", In Proceedings of the ACM Sigmod Record, Volume 25, Issue 1, pp. 38-51, Mar 1996.
[19] J. Taina, and K. Raatikainen, "RODAIN: A Real-Time Object-Oriented Database System for Telecommunications", In Proceedings of the DART'96 workshop, pp. 12–15, 1996.

[20] A. Datta, S. Mukherjee, I. Viguier, and A. Bajaj, “Multiclass Transaction Scheduling and Overload Management in Firm Real-Time Database System”, In Proceedings of the ACM Journal on Information Systems, Volume 21, Issue 1, pp. 29 – 54, Mar 1996.

[21] A. Bestavros, and S. Nagy, "An admission control paradigm for value cognizant real-time databases", In the Proceedings of ACM on RTSS, Proceedings of the 17th IEEE Real-Time Systems Symposium, pp. 230-239, Jan 1996.

[22] J. A. Stankovic, S. H. Son, and J. Liebeherr, "BeeHive: Global Multimedia Database Support for Dependable, Real-Time Applications", Springer Link on Active, Real-Time and Temporal Database system, Volume 1553, pp. 51-69, 1999.

[23] J. Hansson, S. F. Andler and S. H. Son, “Value-Driven Multi-Class Overload Management”, In the Proceedings of 6th Conference on Real-Time Computing Systems and Applications, pp. 21-28, 1999.

[24] K.-D. Kang, S. Son, and J. Stankovic, "Service Differentiation in Real-Time Main Memory Databases", In the Proceedings of 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 119-128, 2002.

[25] G. C. Buttazzo, “Hard Real-Time Computing Systems – Predictable Scheduling, Algorithms and Applications”, Kluwer Academics Publishers, 2002.

[26] A. Cervin, J. Eker, B. Bernhardsson, and K. Arzn, “Feedback-feedforward Scheduling of Control Tasks”, In Proceedings of the Journal on Real-Time Systems, 2002.

[27] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, “Using Control Theory to Achieve Service Level Objectives in Performance Management”, In Proceedings of the Journal on Real-Time Systems, 2002.

[28] M. Amirijoo, J. Hansson, S. H. Son, “Error-driven QoS Management in Imprecise Real-Time Databases”. In Proceedings of 15th Euromicro Conference on Real-time Systems (ECRTS'03), 2003.

[29] M. Amirijoo, J. Hansson, S. Gunnarrson, S. H. Son, “Robust Quality Management for Differentiated Imprecise Data Services”, In Real-Time Systems Symposium, 25th IEEE International, pp. 265-275, 2004.

[30] Mehdi Amirijoo, Nicolas Chaufette, Jorgen Hansson, Sang H. Son and Svante Gunnarsson, “Generalized performance management of multi-class real-time imprecise data services”, In the Proceedings of 26th IEEE International Symposium on Real-Time Systems, pp.12-49, 2006.

[31] M. Amirijoo, J. Hansson, and S. Song, "Specification and Management of QoS in Imprecise Real-Time databases", In Proceedings of the IEEE Transactions on Computers, Volume 55, Issue 3, pp. 304-319, Mar 2006.

[32] Leila Baccouche, “An Overview of MOA, a Multi-Class Overload Architecture for Real-time Database Systems: Framework and Algorithms”, In Proceedings of the IEEE Conference on Computer Systems and Applications, pp. 756-763, 2006.

[33] R. K. Mishra and Dr. U. Shanker, “Overload Management and Admission Control in Real Time Distributed Database Systems”, IJCST, Vol. 3, Issue 2, pp. 29-34, June 2012.