QUALITY OF SERVICE MANAGEMENT IN DISTRIBUTED FEEDBACK CONTROL SCHEDULING ARCHITECTURE USING DIFFERENT REPLICATION POLICIES

Malek Ben Salem[1], Emna Bouazizi[1], Rafik Bouaziz[1],Claude Duvalet[2]

[1]MIRACL, HI of Computer Science and Multimedia, Sfax University, Tunisia
[2]LITIS, Université du Havre, 25 rue Philippe Lebon, BP 540, F-76058 ,
Le Havre Cedex, France

ABSTRACT

In our days, there are many real-time applications that use data which are geographically dispersed. The distributed real-time database management systems (DRTDBMS) have been used to manage large amount of distributed data while meeting the stringent temporal requirements in real-time applications. Providing Quality of Service (QoS) guarantees in DRTDBMSs is a challenging task. To address this problem, different research works are often based on distributed feedback control real-time scheduling architecture (DFCSA). Data replication is an efficient method to help DRTDBMS meeting the stringent temporal requirements of real-time applications. In literature, many works have designed algorithms that provide QoS guarantees in distributed real-time databases using only full temporal data replication policy.

In this paper, we have applied two data replication policies in a distributed feedback control scheduling architecture to manage a QoS performance for DRTDBMS. The first proposed data replication policy is called semi-total replication, and the second is called partial replication policy.

KEYWORDS

DRTDBMS, Quality of Service, Partial Replication, Semi-Total Replication.

1. INTRODUCTION

Distributed real-time database systems have a wide range of applications such as mobile and wireless communications, distance education, e-commerce treatment and industrial control applications as well as the control of nuclear reactions.

The DRTDBMS include a collection of sites interconnected via communication networks for distributed transactions processing. Each site includes one RTDBMS. In distributed environment, the presence of many sites raises issues that are not present in centralized systems.

In an application involving the use of DRTDBMS, user requests arrive at varying frequencies. When the frequency increases, DRTDBMS balance will be affected. During overload periods, DRTDBMS will potentially have fewer resources, and then real-time transactions will miss their deadlines. To address this problem, more studies focus on feedback control techniques have been proposed [1,5] to provide better QoS guarantees in replicated environment.

Data replication consists on replicating a data on participating nodes. Given that this technique increases the data availability, it can help DRTDBMS to meet the stringent temporal requirements. In literature, two data replication policies are presented: the full data replication policy and the partial data replication policy [8].

Wei et al [8] have proposed a QoS management algorithm in distributed real-time databases with full temporal data replication. They proposed a replication model in which only real-time data are replicated and the updating replicas data is propagate at the same time to all replicas in each other nodes. In replicated environment, user operations on data replicas are often read operations [8]. In this case, to guarantee the replicas data freshness, many efficient replica management algorithms have been added in literature. All proposed algorithms aim to meet deadlines of transactions and data freshness guarantees.

This proposed solution in [8] is shown to be appropriate for a fixed number of participating nodes, e.g., 8 nodes. However, in the presence of a high number of nodes (more than 8 nodes), using a full data replication policy is inefficient in these systems, and it may have many limitations. Those issues are related to communication costs between nodes, highly message loss and periodically collected data performance.

To address this problem, we proposed in this work a new data replication policy called a semi-total data replication policy and, we have applied it in feedback control scheduling architecture in replicated environment. Furthermore, we have applied the partial replication policy in this architecture, and we have presented a comparison between obtained results with these data replication policies and the existing results with full data replication policy proposed in previous work. Compared to previous work [8], we increment the number of nodes. Also, in our replication model, we proposed to replicate both types of data; the classical data and the temporal data. In this model, temporal replicas data are updating transparently, and the classical replicas data are updating according to the RT-RCP policy presented in [3].

The main objective of our work is to limit the miss ratio of the arrived transactions to the system. At the same time, our work aims to tolerate a certain imprecise value of replicas data and to control timely transactions, which must guarantee access only to fresh replicas data, even in the presence of unpredictable workloads.

In this article, we begin by presenting a distributed real-time database model. In the section 3, we present the related works on which, we base our approach. In section 4, we present the proposed QoS management approach based on DFCS architecture to provide QoS guarantees in DRTDBMS. Section 5 shows the details of the simulation and the evaluation results. We conclude this article by discussing this work and by focusing on its major perspectives.

2. RELATED WORK

In this section, we present the QoS management architecture proposed in [3], on which we based our work. We describe, also, an overview of the data replication policies presented in literature, which is used for the QoS enhancement.

2.1. Architecture for QoS management in DRTDBMS

The QoS can be considered as a metric which measures the overall system performance. In fact, QoS is a collective measure of the service level provided to the customer. It can be evaluated by different performance criteria that include basic availability, error rate, response time, the rate of successful transactions before their deadline, etc.

The DRTDBMS as RTDBMS have to maintain both the logical consistency of the database and its temporal consistency. Since it seems to be difficult to reach these two goals simultaneously because of the lack of predictability of such systems, some researchers have designed new techniques to manage real-time transactions. These techniques use feedback control scheduling theory in order to provide an acceptable DRTDBMS behaviour. They also attempt to provide a fixed QoS by considering the data and the transactions behaviour.

In RTDBMS, many works are presented to provide a QoS guarantees. Those works consist of the applicability of most of the management techniques of the real-time transactions and/or real-time data [1,2,9,11].

A significant contribution on QoS guarantees for real-time data services in mid-size scale of DRTDBMS is presented by Wei et al. [8] (cf. Figure 1). The authors have designed an architecture, which we base our work, that provides QoS guarantees in DRTDBMS with full replication of only temporal data with small number of nodes (8 nodes). This architecture, called Distributed FCSA (DFCSA), consists of heuristic Feedback-based local controllers and global load balancers (GLB) working at each site.

The general outline of the DFCSA is shown in Figure 1. In what follows, we give a brief description of its basic components.

The admission controller is used to regulate the system workload in order to prevent its overloading. Its functioning is based on the estimated CPU utilization and the target utilization set point. For each admitted transaction, its estimated execution time is added to the estimated CPU utilization. Therefore, transactions will be discarded from the system if the estimated CPU utilization is higher than the target utilization set by the local controller.

The transaction manager handles the execution of transactions. It is composed of a concurrency controller (CC), a freshness manager (FM), a data manager (DM) and a replica manager (RM).

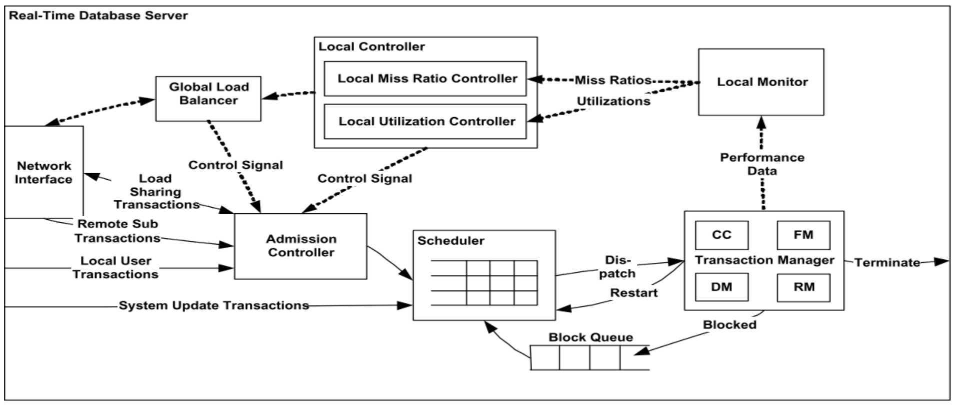

Figure 1 FCS architecture for QoS guarantee in DRTDBMS [8].

2.2. Distributed real-time database model

In this section, we present the distributed real-time database model on which we base our work. This model is issued from many works about QoS management in DRTDBMS [6]. The main difference is the applicability of different data replication policies. In our works, DRTDBMS model is defined by the interconnection between many centralized RTDBMS. We consider a main memory database model on each site, in which the CPU is the main system resource taken into account.

2.2.1. Data model

In this model, data objects are classified into either real-time or non real-time data. In our model, we consider both types of data objects. Non real-time data are classical data found in conventional databases, whereas real-time data have a validity interval beyond which they become stale. These data may change continuously to reflect the real world state (e.g., the current temperature value). Each real-time data has a timestamps which define its last observation in the real world state [9], a validity interval and a value.

2.2.2. Data replication model

In DRTDBMS, data replication is very attractive in order to increase the chance of distributed real-time transaction to meet its deadline, system throughput and provide fault-tolerance. However, it is a challenge to keep data copies consistent. For that purpose, two types of data replication policies have been developed in literature.

The first data replication policy is the full data replication; the entire database at each node is replicated to all other sites which admit transactions that can use their replicas data. Therefore, all the data of this database will be available in different sites which facilitates access by the various local and remote transactions.

The second policy data replication is the partial data replication policy, which is based on access history of all transactions in each node. In fact, for each node, if the number of access on one data object by current transactions reach a threshold then current node request to replicate locally this data object. Therefore, this policy consist of replicate the most accessed data object of the most accessed nodes to satisfy various user requests.

In this paper, we present the third policy. We called it the "Semi-total data replication". Wei et al. [8] have proposed a replication model that only temporal data had replicated and the updating replicas data is propagate at the same time to all replicas in each other nodes. In the replicated environment, the frequently user operations on data replicas are read operations. In this case, to guarantee the replicas data freshness, many efficient replica management algorithms was added in literature to manage the data replicas at every node that supported data replication. All proposed algorithms aims to meet transactions deadlines and data freshness guarantees.

Here, we discuss the difference between the two data replication policies. The full replication is characterized by a maximum number of replicated data. Then, full database replication means that all sites store copies of all data items. An analytical study in [12] has shown the scalability limits of full replication. Therefore, the time needed for updating replicated data is quite important. In some protocols update transactions are executed to preserve all the databases consistency. By taking into consideration the time of transmission of messages between sites, the chance to respect distributed real-time transactions decreases. However, partial data replication policy only assigns copies of a frequently accessed data item. When there is a high update workload, full replication has too much overhead to keep all copies consistent and the individual replicas have little resources left to execute read operations. In contrast, with partial replication, updating protocols for replica data only has to execute the updates for mostly accessed replica data items, and thus, it has more potential to execute read operations.

In the next section, we present our QoS management algorithms based on a different data replication policy.

2.2.3. Transaction model

In this model, we use the firm transaction model, in which tardy transactions are aborted because they can't meet their deadlines. In fact, transactions are classified into two classes: update transactions and user transactions. Update transactions are used to update the values of real-time data in order to reflect the real world state. These transactions are executed periodically and have only to write real-time data. User transactions, representing user requests, arrive aperiodically. They may read real-time data, and read or write non real-time data.

Furthermore, each update transaction is composed only by one write operation on real-time data. User transactions are composed of a set of sub-transactions which are executed at local node or at remote nodes that participate in the execution of the global system.

We consider that a user transaction may arrive at any node of the global system and define its data needs. If all data needed by the transaction exist at the current site, then the transaction is executed locally. Otherwise, the transaction, called real-time distributed user transaction, is split into sub-transactions according to the location of their data. Those sub-transactions are transferred and executed at corresponding nodes.

There are two sub-types of real-time distributed user transaction: remote and local [3]. Remote transactions are executed at more than one node, whereas the local transactions are executed only at one node. There is one process called coordinator which is executed at the site where the transaction is submitted (master node). There is also a set of other processes called cohorts that execute on behalf of the transaction at other sites accessed by the transaction(cohort node). The transaction is an atomic unit of work, which is either entirely complete or not at all. Hence, a distributed commit protocol is needed to guarantee uniform commitment of distributed transaction execution. The commit operation implies that the transaction is successful, and hence all of its updates should be incorporated into the database permanently. An abort operation indicates that the transaction has failed, and hence requires the database management system to cancel or abolish all of its effects in the database system. In short, a transaction is an all or nothing unit of execution.

2.3. Performance metrics

The main performance metric, we consider in our model, is the *Success Ratio* (*SR*). It is a QoS parameter which measures the percentage of transactions that meet their deadlines. It is defined as follows:

$$SR = 100 \times \frac{\#Timely}{\#Timely + \#Tardy} (\%)$$

Where #timely and #tardy represent, respectively, the number of transactions that have met and missed their deadlines.

3. QOS MANAGEMENT APPROACHES USING DIFFERENT DATA REPLICATION POLICIES

In this paper, the number of nodes is larger than the number used in experiments in [8] for full data replication policy. We have also proposed to apply other data replication policies that dedicated to mid-size scale system.

Our work consists of a new approach to enhance the QoS in DRTDBMS. We apply two data replication policies in conventional DFCS architecture; the semi-total replication and partial replication, on the DFCS architecture using a greater number of nodes (16 nodes) of overall system than classical DFCS architecture. Then, we have compared obtained results with those both data replication policies and the result obtained with full data replication policy used in [8]. Moreover, in our replication model, we proposed to replicate both types of data; the classical data and the temporal data. In this model, temporal replicas data are updating transparently, and the classical replicas data are updating as RT-RCP protocol presented by Haj Said et al. [3].

3.1 Approach using semi-total replication policy

In the first part, we propose a new algorithm replication called semi-total replication of real-time and non real-time data (cf. Algorithm 1). In this proposed data replication policy, the system execution start running without any database replication. Distributed real-time transactions require data located in local or in remote nodes. Each node define an access counter for transactions which require data located on remote sites which is define by an access counter of mostly accessed sites. In case where the number of accessed remote data at each node n_i reaches a maximum threshold, then, the current node request the full replication of the database from node n_i. In other case, if the number of accessed remote data at each node n_i reaches a minimum threshold, then the current node decide to remove the full replicated database from node n_i. The maximum and the minimum threshold parameters are defined as input parameter to the system.

Algorithm 1: The semi-total data replication policy

nbAccNode: number of accessed remote node by all transactions from current node
nbAccDataN$_i$: number of accessed remote data by all transactions from current node N_i
MaxAccNodeThres: maximum number of accessed remote node by all transactions.
MinAccNodeThres: minimum number of accessed remote node by all transactions.
N_i : Node i.

```
begin
    for i from 1 to $nbAccNode$ do
        if $nbAccDataN_i >= MaxAccNodeThres$ then
            Current node decide to replicate the full database of the $N_i$
        end
        if $nbAccDataN_i <= MinAccNodeThres$ then
            Current node decide to remove the replicated full database of the $N_i$
        end
    end
end
```

3.2 Approach using partial replication policy

In the second part of our work, we propose to apply a partial data replication policy of real-time and non real-time data (cf. Algorithm 2) in DFCS architecture. This technique may provide good management of database storage by reducing the updating replica data of the overfull system.

The principle of this algorithm is to start running system execution without any real-time and non real-time data replication. User's transactions require data located in local or in remote nodes. We use an accumulator of accessed remote data on other nodes by all transactions of the current node. In this policy, the access counter is calculated for each data, this means that it uses two accumulators; the first consists of the number of the frequently accessed remote nodes, and the second consists of the number of the accessed remote data on frequently accessed nodes. in the first case, if the number of accessed remote data at each node n_i reaches a maximum threshold, then the current node requests the data replication from node n_i. In another case, if the number of accessed remote data at each node n_i reaches a minimum threshold, then the current node decide to remove the current replicated data from node n_i. The maximum and the minimum threshold parameters are defined as input parameter to the system.

The both of those algorithms do not overload the system any more by updating replica data workload compared with full data replication policy. Each one of them has advantages to enhancing the QoS performance in DRTDBMS. This assertion for both algorithms is validated through a set of simulations.

Algorithm 2: The partial data replication policy

nbAccNode: number of accessed remote node by all transactions from current node
nbAccDataN$_i$: number of accessed remote data by all transactions from current node N_i
nbAccDataOcc: number of occurrences of accessed remote data from current node N_i
MaxAccNodeThres: maximum number of accessed remote node by all transactions.
MinAccNodeThres: minimum number of accessed remote node by all transactions.
MaxAccDataThres: maximum number of accessed remote data from current node by all transactions.
MinAccDataThres: minimum number of accessed remote data from current node by all transactions.
N_i : Node i.

```
begin
    for i from 1 to $nbAccNode$ do
        if $nbAccDataN_i >= MaxAccNodeThres$ then
            for j from 1 to $nbAccDataN_i$ do
                if $nbAccDataOcc >= MaxAccDataThres$ then
                    Current node decide to replicate the current data of the $N_i$
                end
            end
        end
        if $nbAccDataN_i <= MinAccNodeThres$ then
            for j from 1 to $nbAccDataN_i$ do
                if $nbAccDataOcc < MinAccDataThres$ then
                    Current node decide to remove the current data of the $N_i$
                end
            end
        end
    end
end
```

4. SIMULATIONS AND RESULTS

To validate our QoS management approach, we have developed a simulator. In this section, we describe the overall architecture of the simulator, and we present and comments the obtained results.

4.1. Simulation model

Our simulator is composed of:

- **Database**: it consists of a data generator that generates data randomly avoiding duplicate information. The consistency in the database is provided by update transactions. In our simulator, the database contains real-time data and classic data.

- **Generator of transactions**: it is composed of two parts.

    - User transactions generator: it generates user transactions using a random distribution, taking into account their unpredictable arrival.

  - Update transactions generator: it generates update transactions according to an arrival process that respects the periodicity of transactions.

- **Precision controller**: it rejects update transactions when the data to be updated are sufficiently representative of the real world, based on the value of MDE.

- **Scheduler**: it is used to schedule transactions according to their priorities.

- **Freshness manager**: it checks the freshness of data that will be accessed by transactions. If the accessed data object is fresh, then the transaction can be executed, and it will be sent to the transactions handler. Otherwise, it will be sent to the block queue. Then, it will be reinserted in the ready queue when the update of the accessed data object is successfully executed.

- **Concurrency controller**: it is responsible for the resolution of data access conflicts. Based on priorities (deadlines) of transactions, it determines which one can continue its execution and which one should be blocked, aborted or restarted.

- **Distributed commit protocol**: it manages the global distributed real-time transaction to ensure that all participating nodes agree with the final transaction result (validation or abortion) [9,10]. In our simulator, we use the commit protocol called "Permits Reading Of Modified Prepared-data for Timeliness" (PROMPT) defined in [4] which allows transactions accessing data not committed (optimistic protocol).

- **Admission controller**: the main role of this component is to filter the arrival of user transactions, depending on the system workload and respecting deadlines of transactions. Then, accepted transactions will be sent to the ready queue.

- **Handler**: it is used to execute transactions. If a conflict between transactions appears, it calls the concurrency controller.

- **Global load balancers** (GLB): it ensures that the load on each node does not affect the functioning of other nodes. So the GLB is used to balance the system load by transferring transactions from overloaded nodes to less loaded nodes in order to maintain QoS.

The input interface of our simulator allows choosing parameter values by which each simulation runs. It includes general parameters of the system, parameters for generating database, generating update transactions and generating user transactions. It also enables to choose the scheduling algorithm, the concurrency control protocol and the real-time distributed validation protocol. Once the simulation is well finished, results are saved in an output file. We can also show results in the form of curves, pie charts or histograms.

4.2. Simulation settings

The performance evaluation of the DRTDBMS is achieved by a set of simulation experiments, in which we varied some of the parameter values. Table 1 summarizes the general system parameters settings used in our simulations. The DRTDBMS is composed of 16 nodes. In each node we have 200 real-time data objects and 10000 classic data objects. Validity values of real-time data are distributed between 500 milliseconds and 1000 milliseconds. For real-time data, the value of MDE varies by 1 between 1 and 10. We choose the universal method to update real-time data. Within each queue in our system, transactions are scheduled according to the EDF algorithm. We use the 2PL-HP protocol for concurrency control. For the validation of transactions, the distributed algorithm PROMPT is chosen. Update transactions are generated according to the number of real-time data in the database.

Table 1. System parameter settings.

Parameter	Value
Simulation time	3000 ms
Number of nodes	16
Number of real-time data	200/node
Number of classic data	10000/node
Validity of real-time data	[500,1000]
MDE value	[1,10]
Method to update real-time data	Universal
Scheduling algorithm	EDF
Concurrency control algorithm	2PL-HP
Distributed validation algorithm	PROMPT

Parameter settings for user transactions are defined in Table 2. User transactions are a set of read and write operations (from 1 to 4). Read operations (from 0 to 2) can access real-time data objects and classic data objects, however, write operations (from 1 to 2) access only classic data objects. The time of one read operations is used to be 1 millisecond, and for one write operation is set to 2 milliseconds. We set the slack factor of transactions to 10. The remote data ratio, which represents the ratio of the number of remote data operations to that of all data operations, is set to 20%. We note that user transactions are generated at arrival times calculated according to the "Poisson" process, which uses the value of the lambda parameter to vary the number of transactions.

Table 2. User transactions parameter settings.

Parameter	Value
Number of write operations	[1,2]
Number of read operations	[0,2]
Write operation time (ms)	2
Read operation time (ms)	1
Slack factor	10
Remote data ratio	20 %

Parameter settings for the real-time and non real-time data replication parameter settings are given in Table 3. To simulate using the partial data replication policy, we have to fix a minimum and a maximum threshold for nodes supporting replication which are set respectively to 0.2 and 0.5. We have to define, also, the value of the minimum and the maximum threshold of the number of accessed remote data at each node, which are set to 0.1 and 0.2. In case of a simulation with semi-total replication, only maximum values of the threshold of accessed remote nodes to replicate their database have to be fixed.

Table 3. Data replication parameter settings.

Parameter	**Value**
Maximum threshold of nodes supporting replication	0.5
Maximum threshold of data to be replicated	0.2
Minimum threshold of nodes supporting replication	0.2
Minimum threshold of data to be replicated	0.1

4.3. Simulation principle

To evaluate the performance of the proposed QoS management approach to enhance the performance of the overall system, we conducted a series of simulations by varying values of some parameters. Each transaction, whatever its type (user or update), undergoes a series of tests since its creation to its execution. In experiments, the workload distribution is initially balanced between all participating nodes.

4.3.1. Simulation using semi-total data replication policy

The first set of experiments evaluates the QoS management using semi-total data replication policy. For each node, an accumulator counter is used to calculate the mostly accessed remote nodes which the current node requests to replicate fully their databases.

4.3.2. Simulation using partial data replication policy

In this set of simulations, we evaluate the QoS management using partial data replication policy. For each node, as with semi-total replication, an accumulator counter is used to calculate the mostly accessed remote nodes. For each frequently accessed node, an accumulator counter is used to calculate the mostly accessed data which are requested to be replicated in the current node.

4.4. Results and discussions

As shown in Figure 2, the transactions success ratio is not affected by the increase of incoming user transactions. In fact, the system workload remains balanced and the system behaviour is maintained in stable state. Also, it is shown that the number of successful transactions with partial and semi-total data replication policies is greater than with full data replication policy. Indeed, the QoS guarantees is defined by increasing the number of transactions that meet their deadlines using fresh data.

We can say that the use of semi-total and partial data replication policy is suitable when increasing the number of participating nodes in DRTDBMS. By this way, the applicability of these data replication policies provides an optimistic management of the database storage while using fresh data by reducing the time of updating replicas.

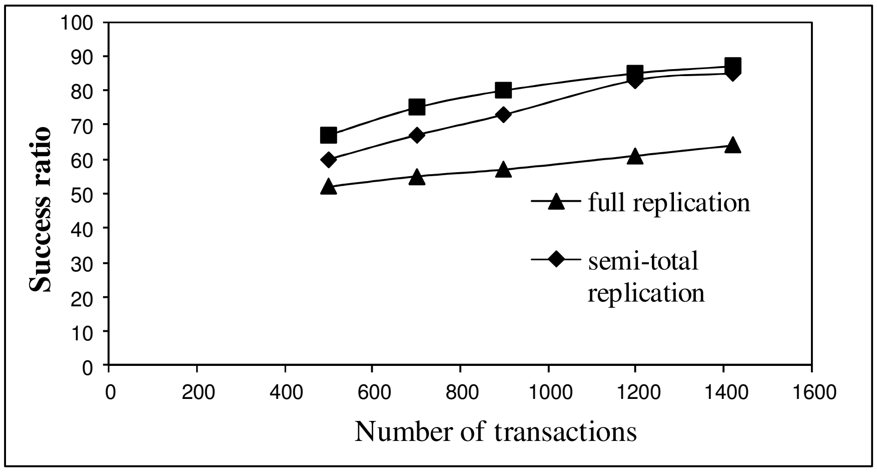

Figure 2. Simulation results for user transactions

The proposed QoS management, using semi-total and partial data replication policies, provides a better QoS guarantees than full data replication policy, ensuring stability and robustness of DRTDBMS.

5. Conclusion And Future Work

In this article, we presented our QoS management approach to provide QoS guarantees in DRTDBMS. This approach is an extension work of the DFCS architecture proposed by Wei et al. [8]. It consists on applying two data replication policies, semi-total replication and partial replication of both classical data and real-time data on the conventional DFCS architecture, in order to make DRTDBMS more robust and stable. Indeed, the proposed approach is defined by a set of modules for data and transaction management in distributed real-time environment. The proposed approach helps to establish a compromise between real-time and data storage requirements by applying different data replication policies.

In future work, we will propose an approach for QoS enhancement in DRTDBMS using multi-versions data with both semi-total and partial data replication policies.

References

[1] Amirijoo, M. & Hansson J. & Son, S.H., (2003) « Specification and Management of QoS in Imprecise Real-Time Databases », Proceedings of International Database Engineering and Applications Symposium (IDEAS).

[2] Amirijoo, M. & Hansson J. & Son, S.H., (2003) «Error-Driven QoS Management in Imprecise Real-Time Databases », Proceedings 15th Euromicro Conference on Real-Time Systems.

[3] Haj Said, A. & Amanton, L. & Ayeb, B., (2007) « Contrôle de la réplication dans les SGBD temps reel distribués», Schedae, prépublication n°13, fascicule n°2, pages 41-49.

[4] Haritsa, J. & Ramamritham, K. & Gupta, R., (2000) « The PROMPT Real-Time Commit Protocol », IEEE Transactions on Parallel and Distributed Systems, Vol 11, No 2, pp 160-181.

[5] Kang, K. & Son, S. & Stankovic, J., (2002) « Service Differentiation in Real-Time Main Memory Databases », 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC02).

[6] Ramamritham, K. & Son, S. & Dipippo, L., (2004) « Real-Time Databases and Data Services », Real-Time Systems journal,Vol 28, pp 179-215.

[7] Sadeg, B., (2004) « Contributions à la gestion des transactions dans les SGBD temps réel », University of Havre.

[8] Wei, Y. & Son, S.H. & Stankovic, J.A. & Kang, K.D., (2003) « QoS Management in Replicated Real Time Databases », Proceedings of the IEEE RTSS, pp 86-97.

[9] Shanker, U. & Misra, M. & Sarje, A.K., (2008) « Distributed real time database systems: background and literature review », Springer Science + Business Media, LLC.

[10] Jayanta Singh, J. & Mehrotra, Suresh C., (2009) « An Analysis of Real-Time Distributed System under Different Priority Policies », World Academy of Science, Engineering and Technology.

[11] Lu, C. & Stankovic, J.A. & Tao, G. & Son, S.H., (2002) « Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms », Journal of Real- Time Systems, Vol 23, No ½.

[12] Serrano, D. & Patino-Martinez, M. & Jimenez-Peris, R. and Kemme, B., (2007) « Boosting Database Replication Scalability through Partial Replication and 1-Copy-Snapshot-Isolation », IEEE Pacific Rim Int. Symp. on Dependable Computing (PRDC).