How to Manage Real-Time Transactions Overload in ε-data Applications ?

Samia Saad-Bouzefrane
Laboratoire CEDRIC, Conservatoire National des Arts et Métiers
292, rue Saint Martin, 75141, Paris, FRANCE
samia.bouzefrane@cnam.fr

Abstract

Current applications are distributed in nature and manipulate time-critical databases with firm-deadline transactions. A transaction submitted to a master site is splitted into subtransactions submitted to participant sites which manage each a local database. In this paper, we propose the OCP-$\overline{S}$C (*Overload Control Protocol in non Strict Context*) for managing real-time distributed transactions with firm-deadline, in the context of possible overload situations and imprecise data acceptable utilization. This protocol has been experienced and its performances measured with a web application based on a 3-tiers architecture and developed on WebSphere.

Key words : commit processing, real-time transaction, overload, real-time scheduling, distributed database management system.

1. Introduction

Current applications, such as Web-based services, electronic commerce, mobile guidance by telecommunication systems, multimedia applications, etc. are distributed in nature and manipulate time-critical databases. In order to enhance the performance and the availability of such applications, the major issue is to develop protocols that manage efficiently real-time transactions while tolerating overload in the distributed system. In fact, if the system is not designed to handle overloads, the effects can be catastrophic and some primordial transactions of the application can miss their deadlines. While many efforts have been made in the management of transient overloads in centralized Real-Time Database Systems (RTDBSs) (Hansson et al., 1998; Bestavros et al., 1996; Hansson et al., 2001) few works control the overload in a distributed environment.

The objective of maintaining logical consistency in the database is difficult to reach ; some proposed works attempt to relax the isolation transaction property, i.e., serializability, usually considered as the transaction correctness criteria in traditionnal database management systems (Eswaran et al. 1976). Indeed, there exists many kinds of applications where strict serializability is not necessary; hence, that may tolerate data imprecision.

In this paper, we focus on the design of a commit processing protocol that cares of overload situations and that bounds database logical inconsistencies. The overload occurs when the computation time of transactions set exceeds the time available on the site processor and then the deadlines can be missed. ε-data concept initially proposed in (Saad et al., 2000) is employed here as a correctness criterium to guarantee the consistency of the distributed database. Our study is concerned by "firm-deadline" transactions because many current applications such as Web-based services use communication protocols with timeout features. In firm-deadline applications, each transaction that misses its deadline is useless and then aborted immediately. The remainder of this paper is organized as follows : Section 2 presents the related work. Section 3 introduces a notion close to epsilon serializability called *ε-data* that allows more execution concurrency between transactions. In Section 4 is described the mechanism used to control transactions overload. Section 5 presents the database model used whereas in Section 6 is described the principle of our protocol. Section 7 presents the architecture of a Stock Exchanged application that has been developed on WebSphere and that uses our protocol. The experimental results show that our protocol has good performances under overload and ε-data conditions.

2. Related work

Many authors have designed real-time scheduling algorithms that are resistant to the effects of system overload (Buttazo et al., 1998; Koren et al., 1995). However, designing algorithms to manage overload in RTDBSs has received comparatively little attention and the few efforts in this area have assumed a centralized real-time database system. For example, Hansson et al. in (Hansson et al., 1998; Hansson et al., 2001) propose an algorithm denoted OR-ULD (Overload Resolution-Utility Loss Density) that resolves transient overloads by rejecting non critical transactions and replacing critical ones with contingency transactions. Bestavros et al., in (Bestavros et al., 1996), consider overload management for soft-deadline transactions where primary transactions have compensating transactions. Transactions are guaranteed to complete either by successful commitment of the primary transaction or by safe transaction of the compensating transaction. Among the techniques that use the concept of importance, Saad et al., in (Saad et al. 2003), have proposed a protocol to control the transactions load in a replicated RTDBS. Transactions are assigned values used to define the importance degree of each transaction with respect to the application. In order to decrease the transactions load, only the transactions declared "important" by the application developer have their execution maintained, the other transactions considered as less important are rejected. In this paper, we will apply this principle to resolve transient overloads that may occur in distributed real-time database systems.

Furthermore, to increase concurrency execution between transactions without loss of data consistency we relax ACID properties (Atomicity, Consistency, Isolation, Durability) judged too restrictive in real-time context. Researchers have proposed techniques that take into account data semantics to relax these properties. These works have led to the development of forced wait and data similarity policies (Xiong et al., 1996) and epsilon serializability criterion (Pu, 1991). In the next section, we recall the principle of ε-data concept used to relax data properties in (Saad et al., 2000). Besides the overload control, the protocol that we present here integrates the ε-data concept in order to allow more transactions concurrency and therefore to reduce the number of firm transactions that miss their deadlines.

3. ε-data notion

ε-data, is a notion close to epsilon serializability (Pu, 1991). It introduces some levels of data-imprecision in order to deal with real-time applications that tradeoff consistency for timeliness, provided the amount of inconsistency introduced can be controlled. Many of the multi-media applications may tolerate some bounded imprecision (an image received is still useful even if some pixels are lost). The ε-data concept deals with data absolute-value imprecision which is a percentage of the current value of the data. For example, let a data item $d = 20$. If d tolerates an imprecision ε then it is called ε-data. Let $\varepsilon = 10\%$, then we tolerate the use of a value which is in the interval [20-2, 20+2. We denote by ε-data, a data item whose exact value is *v* and where each value in $[v-\varepsilon , v+\varepsilon]$ is acceptable. In strict context, if a write transaction already holds a lock on a data item *d*, a query transaction requesting a lock on *d* will wait or will be aborted. In ε-data applications, the isolation level tolerated is not maximal, that is, each read transaction may access a data while another transaction is writing it provided that the quantity of inconsistency introduced by the write transaction is less than a specified bound, ε. Therefore, locks management used by transactions is somewhat different from the one used in classical RTDBSs.

4. Overload Management

The arrival of a new subtransaction may cause an overload, and thus a timing fault, if the required computation times and deadlines exceed the computing capability of each site processor to fulfil all subtransactions deadlines. Our overload management policy aims to favour the execution of the most important transactions of the application. A favoured transaction must undergo less timing faults (i.e., deadline misses) and less abortions due to overload than other transactions. It is then necessary to have a means to designate the most important transactions. This means is presented in the form of a parameter called *importance value* as described in (Saad et al., 2003).

4.1 Transaction Importance

Each global transaction is associated with a positive integer that represents the importance value of the transaction in the transactions set. The importance value is intrinsic to the application, so it can be fixed by the application developer. Each transaction is characterized by a deadline which defines its urgency and by an importance value which defines the criticality of its execution, with respect to the other transactions of the real-time application. The importance of a transaction is not related to its deadline; thus two different transactions which have the same deadline may have different importance values.

4.2 Stabilization process

The stabilization process aims to manage system overload and consists in privileging the transactions that have high importance values. Transactions that have the lowest importance values are released from the ready-transactions queue and are aborted until the processor laxity recovers a positive value. The processor laxity, at time *t*, is the maximum time the processor may remain idle, after *t*, without causing a transaction to miss its deadline.

5. The database model

In our model, we consider only firm real-time transactions. A transaction is submitted to the master site where is executed the *coordinator* process. The result of the transaction is delivered before transaction deadline. Distinct parts of the database are located on different sites. The global transaction is decomposed into subtransactions that are sent to the participant sites, where they are managed by a *cohort* process. Each participant site includes three modules :

- a scheduler module that uses EDF scheduling algorithm,
- an overload-manager module that controls transient overloads each time a new subtransaction is inserted in the ready-transactions queue and
- a data-manager module that applies ε-data concept when subtransactions of the participant site conflict when accessing data.

The model does not consider database replica. Two types of subtransactions are allowed in our environment: query subtransactions and update subtransactions. A global transaction is characterized by its arrival time, its deadline and its importance value. In the same way, within a site a subtransaction is characterized by its arrival time, its execution duration, its deadline and its importance value. Note that the deadline and the importance value of a subtransaction are inherited from the global transaction to which it belongs.

6. Protocol description

Our protocol OCP- $\overline{\text{S}}$C (*Overload Control Protocol in non Strict Context*) is designed to manage distributed real-time transactions. It focuses on firm real-time transactions and uses the model described in previous section. OCP- $\overline{\text{S}}$C manages overload within each participant site while transactions manipulate ε-data. It is based on the 2PC (two Phase Commit) protocol for the communications between sites.

6.1 Integrating overload control

OCP- $\overline{\text{S}}$C augments 2PC protocol in order to handle overload conditions. When the coordinator process receives a transaction T_i to execute, it splits it into subtransactions. For each subtransaction ST_{ij}, the coordinator sends a message INITIATE(ST_{ij}) to execute ST_{ij} on the cohort process that manages data items needed by ST_{ij}. When the cohort receives an INITIATE(ST_{ij}) message, it applies the stabilization process. That is, if the site is overloaded because all the local subtransactions cannot be executed on time, the ready-transactions queue is stabilized by aborting the subtransactions that have the lowest importance values. When a subtransaction is aborted, the cohort sends a "NO" vote to the coordinator. As soon as it receives a "NO" vote, the coordinator broadcasts ABORT messages to all the cohorts for invalidating the local subtransactions. On the other hand, if the coordinator receives YES messages from all its cohorts then it broadcasts to them COMMIT messages.

6.2 The stabilization process description

The conditional laxity $\text{LC}_{ST_i}(t)$ of a transaction ST_i of the ready queue is the maximum time during which ST_i may be delayed without missing its deadline, with the assumption that all transactions with earlier deadline will finish their execution before ST_i may start running. The processor laxity *LP* is defined as a minimal value of the conditional laxity of each transaction of the ready queue. An overload situation is detected as soon as the site laxity $LP(t)$ is less than 0. The late transactions are those whose conditional laxity is negative. The overload value is equal to the absolute value of the processor laxity, $|LP(t)|$. The stabilization process consists in privileging, when the site is overloaded, transactions with high importance values. For this purpose, we remove from the $readyQueue_{s,\text{t}}$ the transactions that have the lowest importance values until the laxity is positive anew. Moreover, we should not remove a transaction *ST* from $readyQueue_{s,\text{t}}$ when this queue contains transactions that have lower importance values than *ST* and which rejection would have resulted in a positive laxity.

6.3 Integrating ε-data concept in the locking condition

Each time a subtransaction is submitted to a cohort, the site-processor laxity is computed. If the laxity is negative, one or more subtransactions with low importance values are aborted in order to maintain a positive laxity. Moreover, to increase concurrency between the subtransactions within a participant site, we apply the ε-data concept. The locking condition concerns a situation where a data item *d* is write-locked by a W^ε transaction and a query transaction $\text{Q}_\text{i}^\varepsilon$ requests to read-lock *d*. Instead of blocking or aborting $\text{Q}_\text{i}^\varepsilon$ transactions as in classical protocols, in our protocol all $\text{Q}_\text{i}^\varepsilon$ transactions pursue their execution concurrently with W^ε provided that the difference between the value written by W^ε and the value read by $\text{Q}_\text{i}^\varepsilon$ transactions does not exceed ε. Otherwise, if the value written by W^ε is out of range then the transaction manager behaves classically.

To increase concurrency between transactions, ε-data concept operates at two levels :

- *at the preparation phase of a write transaction* : all read transactions that request for a data item writelocked by a transaction that is in its preparation phase, may execute in parallel with the write transaction provided that the data-item inconsistency is bounded by ε.
- *at the uncertainty phase of a write transaction*: the uncertainty phase of a transaction begins at the time when it finishes its execution and remains waiting for a COMMIT or an ABORT message. Due to message exchanges, this uncertainty phase may last some significant time. Hence, if a writelocked data item provide a bounded inconsitency then all the read transactions that request this data item will access to its value and continue their execution in parallel with the

write transaction. During the uncertainty phase, read locks are released while write locks are kept until the transaction validation.

7. Performance

In our experimental platform, transactions are submitted via Web forms to the Web server that propagates them to EJB server that has the role of the master. The EJB server translates the transactions to SQL requests that are sent on the different ORACLE servers placed on different nodes. Each node includes an overload controller and a data manager that applies the ε-data concept.

In order to experiment our platform, we have implemented a Stock Exchange application. This application defines, through Web forms, three types of transactions :

- a buying order of stock exchange actions corresponding to one or more companies,
- a sale order of stock exchange actions as well as
- the posting of the current values of one or more actions.

Sale and buying orders have high importance values comparatively to a posting order.

The database is distributed over 3 nodes. Each node contains a distinct database that stores the names of companies, the number of stock exchange actions, the current value of each action and the associated shareholders. Each action has a value and an ε-value equal to 5% of the action value.

We perform some experiments on the platform in four cases :

- case 1: we experiment a situation where the overload is controlled as well as the use of ε-data concept,
- case2: we experiment a second situation where ε-data concept is not applied,
- case 3: the third experiment concerns a situation where the overload is not controlled,
- case 4: the fourth experiment is based on a situation where neither overload control nor ε-data concept are implemented.

Figure 1 shows better performances in the first situation, that is, when the overload is controlled and the ε-data concept is used. In theses experiments, we measure the number of important transactions that miss their deadlines. This number is minimal in the first situation.

8. Conclusion

In this paper, we have focused on the design of a commit processing protocol that bounds database logical inconsistencies when the distributed system is overloaded. When an overload is detected within a participant site, the transactions that are important for the application are favoured. The less important ones are discarded from the system. ε-data concept is employed here as a correctness criterium to guarantee the consistency of the distributed database. We have implemented our protocol and we have illustrated its efficiency by developing a Stock Exchanged application on WebSphere, that uses ε-data notion. The transactions of the clients are defined with importance values so that the most important ones are maintained in overload situations. The experimental results show that our protocol has good performances under overload and ε-data conditions. We are currently refining our simulation in order to integrate acceptance tests and to measure a defined efficiency ratio.

9. References

(Bestavros et al., 1996) A. Bestavros and S. Nagy,"Value-cognizant Admission Control for RTDB systems", *Proc. of the 17th Real-Time Systems Symp.*, pp. 230-239, IEEE Computer Society, dec. 1996.

(Buttazo et al., 1998) G. C. Buttazo, G. Lipari and L. Abeni, "Elastic Task Model for Adaptive Rate Control", in *Proc. of IEEE Real-Time Systems Symposium*, Madrid, dec. 1998.

(Eswaran et al., 1976) K. P. Eswaran, J. N. Gray, R. A. Lorie and I. L. Traiger, "The notion of consistency and predicate locks in a database system", CACM, **9(11)**, pp. 624-633, 1976.

(Hansson et al., 1998) J. Hansson, S. H. Son, J.A. Stankovic and S. F. Andler,"Dynamic Transaction Scheduling and Reallocation in Overloaded Real-Time Database Systems", *Proc. of the 5th Conference on Real-Time Computing Systems and Applications (RTCSA'98)*, pp. 293-302, IEEE Computer Press, 1998.

(Hansson et al., 2001) J. Hansson and S. H. Son,"Real-Time Database Systems: Architecture and Techniques", *K. Lam and T. Kuo (eds.)*, Kluwer Academic Publishers, pp. 125-140, 2001.

(Koren et al., 1995) G. Koren and D. Shasha, "Dover: An Optimal On-Line Scheduling Algorithm for Overloaded Uniprocessor Real-Time Systems", *SISAM J. Comput.*, **24(2)**, pp.318-339, 1995.

(Pu 1991) C. Pu, "Generalized Transaction Processing with Epsilon Serializability", in Proc. of the 4th Int. Workshop on High Performance Transaction Processing, sept. 1991.

(Saad et al., 2000) S. Saad-Bouzefrane and B. Sadeg, "Relaxing the Correctness Criteria in Real-Time DBMS", Int. Journal of Computers and their Applications, 7(4), pp. 209-217, 2000.

(Saad et al., 2003) S. Saad-Bouzefrane and C. Kaiser, "Distributed Overload Control Control for Real-Time Replicated Database Systems", 5th Int. Conf. On Enterprise Information Systems, april 2003, Angers, France.

(Xiong et al., 1996) M. Xiong, J. A. Stankovic, K. Ramamritham, D. Towsley and R. M. Sivasankara, "Maintaining Temporal Consistency : issues and algorithms", 1st Int. Workshop on RTDBS: Issues and Applications, pp. 1-6, California, 1996.

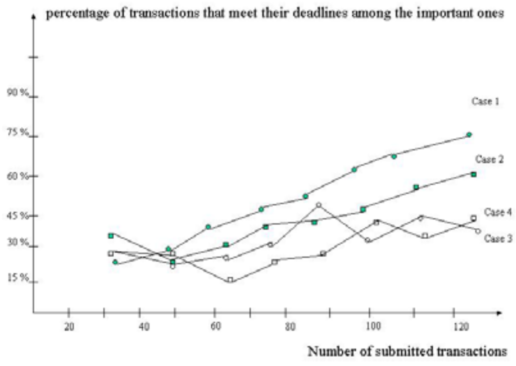

Figure 1: *Simulation results obtained in the situations described in Section 7*