A TRANSACTIONAL MULTIMODE MODEL TO HANDLE OVERLOAD IN DISTRIBUTED RTDBSs

Samia Saad-Bouzefrane, Sofiane Bourenane
Laboratoire CEDRIC, Conservatoire National des Arts et Métiers, 292, rue Saint Martin, 75141 , Paris, France

Keywords Real-time database management system, real-time scheduling, commit process, overload, real-time transaction.

Abstract: Current applications, such as Web-based services, electronic commerce, mobile telecommunication systems, 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 efficient protocols that cooperate with the scheduler to manage the overload of the distributed system. In order to help real-time database management systems (RTDBS) to maintain data logical consistency while attempting to enhance concurrency execution of transactions, we introduce a transactional multimode model to let the application transactions adapt their behavior to the overload consequences. In this paper, we propose for each transaction several execution modes and we derive an overload controller suitable for the proposed multimode model.

1 INTRODUCTION

Current applications, such as Web-based services, electronic commerce, mobile telecommunication systems, 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 efficient protocols that cooperate with the scheduler to manage the distributed-system overload.

Data are structured and managed by a real-time database management system (RTDBS) which implements functionalities ensuring the data logical consistency and the respect of temporal constraints (Duvallet et al. 1999; Ramamritham 1993; Xiong et al., 1996) for transactions.

In a distributed real-time environment, the occurrence of an overload situation within a site is due to the shortage of processor time, at a given moment, to ensure the execution on time of all the transactions activated on this site. This situation generates an abortion of several transactions and involves, consequently, the loss of resources and a degradation of the quality of service. In order to avoid the undesirable effect of the overload on the distributed transactional treatment, many researchers propose to integrate mechanisms that manage and control the processor load by introducing the concept of importance. In fact, to each transaction is associated an importance value used to alleviate the processor load by removing the least-important transactions of the application (Kaiser et al. 1998, Saad et al.. 2003). Unfortunately, the concept of importance is not sufficient to express the semantic primordiality of the treatment carried out by the transactions. In fact, the abortion of the least-important transactions can also generate a loss in term of data logical and temporal performances. To resolve this problem, we present in this paper a multimode transactional model inspired by the multimode model of tasks proposed by J. Delacroix et al. in (Delacroix et al. 2000). Our model specifies a multimode transactional behavior that allows distributed real-time transactions to adapte their behavior to the overload consequences by starting palliative actions in a degraded mode, instead of being simply aborted.

Our study is concerned with “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 is then aborted immediately. The remainder of this paper is organized as follows : Section 2 presents the related work. Section 3 defines the architecture of the distributed RTDBS. Section 4 proposes a transactional model with several modes associated to transactions to adapte their behavior to the overload situation. Section 5 describes, at a

Saad-Bouzefrane S. and Bourenane S. (2004).
A TRANSACTIONAL MULTIMODE MODEL TO HANDLE OVERLOAD IN DISTRIBUTED RTDBSs.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 328-336
DOI: 10.5220/0002612303280336

global level, the consequences generated by a transaction that switches to another execution mode. Section 6 derives overload-management algorithms that take into account the multimode transactional model. Before concluding in Section 8, Section 7 presents a simulation framework.

2 RELATED WORK

Many authors have designed real-time scheduling algorithms that are resistant to the effects of system overload (Cottet et al. 2002). In fact, some algorithms deal with periodic task sets and allow the system to handle variable computation times which cannot always be bounded (Atlas et al. 1998, Buttazo et al. 1998, Mok et al. 1997). Other algorithms deal with hybrid task sets where tasks are characterized by an importance value (Delacroix 1996, Kaiser et al. 1998, Koren et al. 1995). Designing algorithms to manage overload in RTDBSs has received comparatively little attention, however, 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. The overload resolver cooperates with an EDF scheduling algorithm (Liu et al. 1973) that uses SRP (Stack Resource Policy (Baker 1991)) to handle blocking since the database is main-memory resident. Hansson et al., in (Hansson et al. 1999), improve OR-ULD algorithm to bias the execution among transaction classes such that the minimum completion ratio constraints are satisfied. OR-ULD algorithm has been also evaluated in (Hansson et al. 2000) for imprecise tasks where tasks are decomposed into one mandatory task that has hard deadline and one optional task that has firm deadline.

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. Datta et al., in (Datta et al. 1996), developed a scheduling mechanism to perform admission control both for managing transient overloads and to bias towards particular transaction classes. 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 distributed 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 aborted. However, even the abortion of the least-important transactions may generate a loss in term of data logical and temporal performances. In order to resolve this problem, we propose in this paper a model that specifies a multimode transactional behavior that allows the least-important transactions to start palliative actions in overload situations, instead of being aborted as in (Saad et al. 2003; Kaiser et al. 1998). This model will be used to derive algorithms that manage processor overload.

3 THE SYSTEM ARCHITECTURE

In this paper, we are interested only in “firm-deadline” transactions. Each transaction is submitted on a site called *master site*. It is splitted into subtransactions submitted to sites called *participant sites* (see Figure 1).

Each participant site is composed of four modules:

- a transactional manager that manages the validation of the subtransactions using a traditional two-phase commit protocol (2PC) (Samaras et al., 1995; Bernstein et al.. 1987),
- an EDF scheduler (Liu et al., 1973) that allocates the processor to the highest-priority transaction that is, to the transaction that has the nearest deadline,
- a data manager that manages the data-access conflicts using a two-phase locking strategy 2PL (Bernstein et al.. 1987) and
- an overload controller.

A global transaction, as well as a subtransaction, is characterized by its arrival time, its deadline and its importance value that expresses the criticality of the transaction compared to the other application transactions. The arrival time of a subtransaction corresponds to the time at which it is submitted to the participant site. The deadline and the importance value of a subtransaction are inherited from the global transaction to which the subtransaction belongs. The importance value is a fixed integer. It will be used by the overload controller to select the least-important transactions to consider during the overload-resorption process.

4 THE MULTIMODE TRANSACTIONAL MODEL

Within a participant site, an overload occurs when the time needed to execute all the ready subtransactions exceeds the available processor time. In this situation, one or more subtransactions may not finish their execution before the expiration of their deadline and consequently may be aborted. It becomes then necessary to have a multimode transactional model that defines a behavioral structure for each subtransaction so that a degraded mode is triggered each time a processor overload is detected. In the adapted model of transaction, a subtransaction is made up of at most four execution modes : the normal mode which is executed when the subtransaction begins to execute. It takes care of normal execution of the subtransaction. Three survival modes are executed when the normal mode is rejected, adjourned or revoked. The computation time of a survival mode should be short because it contains only actions that set the application in a safe state. Such specific actions, are for example, release of data locks, validation or abortion of update operations.

4.1 Normal mode

A global transaction T is in normal mode if all its subtransactions STi $\forall$ i $\in$ { 1, n } executed on participant sites are in normal mode. Initially, each subtransaction STi activated by the local transactional manager, starts its execution in normal mode. STi preserves its normal execution mode as long as the overload manager has not selected it as a victim. A subtransaction is considered as a victim if its normal mode is cancelled by the overload manager.

4.2 Survival mode

This mode is handled by subtransactions executed on an overloaded site. A subtransaction STi activates its survival mode when it undergoes, at the time of its execution in normal mode, an action of the overload resorption. The activation of this mode lets the subtransaction to execute palliative actions in a degraded mode. The survival mode is composed of three submodes: a rejection mode, an adjournment mode and a revocation mode. Each submode defines a program code with a level of degradation specified by the programmer. A subtransaction cannot switch to a survival mode as long as its global transaction does not support this type of mode in its multimode contract. The program code associated with the survival mode of a subtransaction must be executed imperatively. As illustrated in Figure 2, the submodes are described in the following :

- Rejection Mode : a subtransaction STi is switched from the normal mode to a rejection mode if the overload manager chooses STi as a victim when STi is requesting for the processor for the first time (STi has not yet begin its execution in normal mode). Indeed, its insertion into the ready queue with a normal mode generates an overload situation. The rejection mode is made up of a program code smaller than that of the normal mode. STi, in rejection mode, can be aborted if its insertion in the ready queue may generate also an overload.

- Adjournment Mode : a subtransaction STi switches from the normal mode to the adjournment mode when it undergoes an adjournment of its execution by the overload controller. An adjournment action is the operation of stopping the normal mode of STi when STi is executing before the preparation phase of 2PC protocol (Bernstein et al., 1987).

- Revocation Mode : a subtransaction STi switches from the normal mode to the revocation mode when it undergoes an action of the overload controller during its preparation phase and before beginning the commit phase. At this advanced execution level, the subtransaction switched to the revocation mode can:

- either execute the data requests in a degraded mode. The behavior of STi may be the same either in the revocation mode or in the adjournment mode.
- or stop the preparation phase and vote by YES in order to accelererate the passage to the commit phase so that the updates are directly posted on the database. The subtransaction STi, in revocation mode, ensures the temporal consistency without guaranteeing the data logical consistency.

Example :

The following transaction T defines two modes, a normal mode and a rejection mode. If T is rejected, it will execute a smaller program code.

```
T.NormalMode {Begin_Transaction
Req1 : Update Company_quotation
  SET Course = Course + Variation
       WHERE Company_identity = FR0006
or Company_identity = FR0011
      Company_identity   =   FR0028   or
Company_identity = FR0031
Req2 : Update Statistical_quotation
  SET High = Van [1], Low = Van [2]
      WHERE Company_identity = FR0006
or Company_identity = FR0011
    Company_identity   =   FR0028   or
Company_identity = FR0031
End_Transaction }
```

```
T.RejectionMode {Begin_Transaction
Req1 : Update Company_quotation
    SET Course= Course + Variation
      WHERE Company_identity >= FR0010
And Company_identity <= FR0030
 End_Transaction }
```

4.3 Multimode demarcation point

The multimode demarcation point is defined by an execution instant from which a subtransaction cannot support any changing of mode. A subtransaction STi reaches its demarcation point when it finishes its preparation phase and sends a message YES to the master site. At this execution level, the transaction cannot be cancelled to switch to a survival mode. Figure 2 locates the demarcation point relatively to the activation levels in the life time of a subtransaction.

4.4 A behavioural contract

A global transaction, as well as a subtransaction, is characterized by a contract determining the supported survival modes. The multimode contract includes three properties and is declared in a properties file that respects an XML format (see the following DTD). This file represents the multimode properties of all the transactions classes : a class gathers all the transactions which have the same behavioral contract.

The DTD of the XML file that defines the multimode contract

```
<!ELEMENT Contract(Class)>
<!ELEMENT Class(Rejection, Adjournment,
Revocation)>
<!ATTLIST Class type #PCDATA #REQUIRED>
<!ELEMENT Rejection (#PCDATA)>
<!ELEMENT Adjournment (#PCDATA)>
<!ELEMENT Revocation (#PCDATA)>
```

This is an example of a file that declares multimode classes.

```
< Contract>
<Class type = j >
< Rejection> No < / Rejection >
<Adjournment> Yes< /Adjournment>
<Revocation > Yes < /Revocation >
</Class >……
<Class type = k >
< Rejection> No < / Rejection >
<Adjournment>   No   <   /Adjournment>
<Revocation > Yes < /Revocation >
</Class>
</Contract>
```

In the example above, the class j gathers all the non-rejectable, adjournable and revocable transactions. The class k includes all the revocable, non-rejectable and non-adjournable transactions.

5 THE CONSEQUENCES OF A SUBTRANSACTION-MODE CHANGING

A subtransaction STk that changes its execution mode can generate one of the two following scenarios:

1. all the subtransactions STi $\forall i \in \{1,n\}-\{k\}$ executed on the other participant sites switch to the same survival mode m. We say that the global transaction T supports a synchronized multimode behavior. This cooperation model for a global switching is suitable for the transactions that require the atomicity of their distributed transactional treatments. STk that causes the global mode switching does not await for the agreement of the other subtransactions.

```
Mode (STi) = normal, for any STi i ∈
{1…n} of T
When STk goes to a survival mode m
     Mode (STk) = m
     For any STi i ∈ {1…n}-{k} of T do
        Switch Mode (STi) to mode m
EndFor
```

We introduce a protocol that allows the overload controller to synchronize the switching mode of subtransactions that belong to the same global transaction. This protocol implements three messages: ·

- *Mode_Changed* (STk, T, Mode = m): message sent by the overload controller of a participant site to inform the master site that subtransaction STk has switched to its survival mode m.
- *Change_Mode* (STi, T, Mode = m): message broadcasted by the master site to all the non-overloaded participant sites to invite them to switch their subtransaction to the mode m.
- *Change_Mode_Ok* (STi, T): message sent by each participant site to the master site once the local switching has been done.

2. A local switching of the execution mode of STk without changing the mode of the other subtransactions STi $\forall$ i $\in$ {1, n}-{k}. This local non-synchronized switching is supported by a transaction which a cancellation of the normal mode does not affect the logical consistency of the database but degrades the data QoS. This relaxation principle avoids the loss of resources.

6 THE OVERLOAD CONTROLLER

The overload controller is composed of two modules: an admission controller and an overload manager. These modules cooperate with the scheduler to set up a scheduling and an overload-management policy which, in overload conditions, favours the execution of the most important transactions and which removes the least important ones by switching them to the survival mode. Each overload controller manages the overload situations within each participant site without communicating its state to the other local overload controllers. However, with the introduction of the multimode transactional model, the overload controllers of the participant sites communicate to ensure the global mode switching by using the protocol described above. The overload controller manages two queues. We denote by:

ReadyQueue$_s$: the ready queue of a participant site S that contains the subtransactions that are waiting for the processor. They are sorted by increasing their deadlines. At time t, *ReadyQueue*$_s$ has the following state :

ReadyQueue$_s$(t) = {$ST_{0,m0}$, $ST_{1,m1}$, $ST_{2,m2}$,…,$ST_{n,mn}$}, m_i is the execution mode of subtransaction STi.

ImportanceQueue$_s$: the importance queue of a participant site S is composed of ready subtransactions executing in normal mode, sorted by increasing their importance values. The state of the queue at time t is the following : *ImportanceQueue*$_s$ = {$ST_{k,normal}$, Imp_{STk}} k $\in$ {1,…,d} (d≤n).

At the arrival of a new subtransaction STn requiring the processor on a participant site S, the scheduler invokes the admission controller. This latter has the role of detecting the overload and deciding the admission or the rejection of STn. The overload manager is called by the admission controller to resorb the overload situation generated by the insertion of STn into *ReadyQueue*$_s$. The scheduler is triggered at the end of each overload-management module to allocate the processor to the highest-priority ready subtransaction.

6.1 The admission controller

The admission controller is made up of two phases: a detection phase and an admission phase. The detection phase detects overload situations. At the arrival of a new subtransaction $ST_{n,normal}$, a parameter called processor laxity (denoted by LP(t)) is computed for the new configuration of *ReadyQueue*$_s$ $\cup$ {$T_{n,normal}$}. The processor laxity corresponds to the minimal value of the conditional laxities of the ready subtransactions.

LP(t) = Min ($LC_{STi,mi}$(t)) i $\in$ {0,.., n} / $\forall$ $ST_{i,mi}$ of (*ReadyQueue*$_s$ $\cup$ {$T_{n,normal}$})

$LC_{STi,mi}$(t) is the conditional laxity of the subtransaction $ST_{i,mi}$. It is equal to the time interval during which $ST_{i,mi}$ can be delayed, from time t, without missing its deadline. A negative value of $LC_{STi,mi}$ means that $ST_{i,mi}$ cannot finish its execution in mode m_i before its deadline expires. If LP(t) corresponds to a positive value, then the admission of $ST_{n,normal}$ in *ReadyQueue*$_s$ does not generate an overload situation. Thus, $ST_{n,normal}$ is accepted and inserted in *ReadyQueue*$_s$. Otherwise, the acceptance of $ST_{n,normal}$ generates an overload situation. In this case, the controller starts its second phase which consists in rejecting or accepting $ST_{n,normal}$. If $ST_{n,normal}$ is more important than one or several normal-mode subtransactions, then it is accepted by the admission controller. The overload manager is called then by the admission controller to stabilize the processor load (Section 6.2 will describe the principle of stabilization process). Otherwise, STn switches to the rejection mode. If the execution in this latter mode does not cause an overload situation $ST_{n,rejection}$ is accepted. Otherwise, it is aborted causing the abortion of the global transaction.

Module Overload_detection (*ReadyQueue*$_s$,$T_{n,normal}$)
Begin
Compute Processor_laxity (*ReadyQueue*$_s$, t)
If Processor_laxity (*ReadyQueue*$_s$, t) < 0 then
Overload_value= |Processor_laxity (*ReadyQueue*$_s$, t)| ;
Call Admission Phase ;
Else
Call the scheduler to insert $T_{n,normal}$ into *ReadyQueue*$_s$;
EndIf End

Admission Phase
Begin
If Imp_{Tn} > Imp_{ST} $\forall$ ST $\in$ *ImportanceQueue*$_s$ Then
$T_{n,normal}$ is admitted ;

Call the scheduler to insert $T_{n,normal}$ to $ReadyQueue_s$;
Call the overload manager to stabilize $ReadyQueue_s$;
Else
Tn is switched to rejection mode $T_{n,rejection}$
If ($Mode_{normal}(Tn)$-$Mode_{rejection}(Tn) \leq$ Overload_value)
Call the scheduler to insert $T_{n,Rejection}$ into $ReadyQueue_s$;
Else Tn is aborted
EndIf
EndIf
End

6.2 The overload manager

This module is called by the admission controller when the overload occurs. It proceeds to the resorption of the overload situation generated by the insertion of a new subtransaction STn in $ReadyQueue_s$. The overload manager implements a mechanism called a stabilization process. When an overload occurs, the stabilization process is executed to release the least-important transactions that are in normal mode, by switching them to the survival mode in order to alleviate the site processor. The stabilization process builds an overload-resolution plan denoted by P. The plan P is made up of a subset of $ReadyQueue_s$, denoted ORset, including the least-important subtransactions that execute in normal mode, that support the survival mode and that have not reached yet their demarcation point.

ORset = {STk} STk is a subtransaction of $ReadyQueue_s$ chosen in the plan P

The stabilization process removes all the subtransactions of ORset chosen in the plan P to alleviate the processor time, ensuring the deadline meeting for the most important subtransactions. In our transactional model, the removal of a subtransaction means its switching from the normal mode to a survival mode m which is adequate to its execution state. That is, (see Figure 2):

- its switching to the revocation mode if it is in the preparation phase of 2PC protocol,
- or its switching to the adjournment mode if it is in its data-processing phase. Processor time freed thanks to this switching is equal to the difference between the remainder execution time in normal mode and its computing time in survival mode m.

Processor time freed= [$Mode_{normal}$ (STk) - $Mode_m$ (STk)].

The use of the resolution plan P stabilizes $ReadyQueue_s$ and generates a new configuration of ready subtransactions denoted by $ReadyQueue_s^{stable}$.

P($ReadyQueue_s$) = $ReadyQueue_s^{stable}$ with Processor_laxity($ReadyQueue_s^{stable}$)>0

The stabilization process stops when one of the following conditions is satisfied:

- the plan P has stabilized $ReadyQueue_s$: P ($ReadyQueue_s$) = $ReadyQueue_s^{stable}$ and the processor laxity is a positive value : Processor_laxity ($ReadyQueue_s^{stable}$)>0. In this case, the overload situation is resorbed completely.
- all the subtransactions STk of ORset verifying the selection rules are inserted in the plan P and the processor laxity is always negative. In this case, the stabilization process cannot resorb the overload situation completely. Consequently, the subtransactions of $ReadyQueue_s$, having a negative conditional laxity, will be aborted.

7 THE SIMULATION FRAMEWORK

7.1 The Java platform

The simulation platform is based on Java technology and makes use of MySQL databases. Its architecture is composed of a master site and three participant sites over which the database system is distributed. Transactions are sent via HTTP requests to the master site which splits them into subtransactions and distributes them to appropriate participant sites, where they are processed into SQL statements and executed. Furthermore, the communication framework is based on socket primitives, rather than CORBA, for performance reasons and messages are modelled as objects.

The master site is implemented as a Java Servlet and sits on a TomCat Server. Transaction requests are made via HTTP and each request is handled by an instance of the Servlet. A transaction request contains one or more data operations. The number of the global transaction is calculated as an addition of the arrival time of the transaction at the master site and a random number ranging from 0 to one million. The transaction deadline is calculated from the arrival time of the transaction at the master site and the execution time of all its data operations. The execution time of each data operation is determined by its type (read or write). An additional time is also included to cater for communication time between sites. The importance value of the transaction is determined from the number of the data operations of the transaction and from the type of each

operation. To each subtransaction is associated a normal mode and a survival mode.

The participant site is composed of various modules, implementing the Overload Manager, the EDF Scheduler and the Data Manager. All these modules are defined like threads and work concurrently.

The application developed to evaluate the performance of our platform under various working conditions is composed of three modules:

- the *Execution Module* : that gets the transactions to be executed from a file and sends them simultaneously, in the form of HTTP requests, to the master site and waits for the execution results.
- the *Configuration Module* that enables the user to change the execution parameters of the distributed system. These parameters allow the user to determine the conditions under which the transactions would be executed. For example, the user can determine whether the importance value of transactions should be considered during the stabilization process or not, etc.
- the *Statistics Module* has the purpose of analysing the series of execution grouped according to the type of configuration under which they have been executed and to display graphically the results of the analysis.

7.2 Simulation analysis

During the simulation phase, several series of execution have been used to measure the performance of the system under different working conditions. The series vary in terms of the number of transactions (50, 100, 150, 200, 250, 300) in order to allow one to evaluate the behavior of each simulation configuration vis-à-vis a linear increase of the number of requests. The series have also been designed in a way to ensure several conflicting data access between read and write operations and few conflicts between write operations. All the tests have been carried out on a platform consisting of participant sites, each having a local database of 30 records. Also, each distributed transaction contains three subtransactions, one for each participant site, and each sub-transaction has at most 3 data operations. The tests presented in Figure 3 have been carried out to evaluate the system under various configurations, comprising the multimode concept and that of the importance value.

The results of Figure 3 show that the system achieves the best performance when the multimode concept is applied and when the importance value of transactions is considered during the stabilization process.

8 CONCLUSION

In this paper, we have focused 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. A simulation platform based on a commit processing protocol that manages transient-overload situations of the distributed system has been developed. When an overload is detected within a participant site, the transactions that are important for the application are favoured. The less important ones are switched into degraded modes or discarded if the degraded modes are not sufficient to resorb overload. The simulation platform is based on Java technology and makes use of MySQL databases. Transactions are sent via HTTP requests to the master which is implemented as a Java Servlet on TomCat server. Each participant site implements an overload manager, an EDF scheduler and a data manager. This platform integrates a graphical interface to submit transactions, a configuration module to fix a certain number of parameters and a statistical module that displays the simulation tests in a graphical way. The simulation results show good performances under overload and multimode execution.

REFERENCES

(Atlas et al. 1998) A. Atlas and A. Bestavros, "Statistical Rate Monotonic Scheduling", in proc. of IEEE Real-Time Systems Symposium, Madrid, dec. 1998.

(Bernstein et al. 1987) P. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems", Addison Wesley, 1987.

(Baker 1991) T. P. Baker, "Stack-based Scheduling of Real-Time Processes", in Real-Time Systems Journal, 3(1), pp. 67-99, march 1991.

(Bernstein et al., 1987) P. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems", Addison Wesley, 1987.

(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.

(Cottet et al., 2002) F. Cottet, J. Delacroix, C. Kaiser and Z. Mammeri, "Scheduling in Real-Time Systems", Edition J. Wiley & Sons, 2002.

(Datta et al., 1996) A. Datta and et al., "Multiclass Transaction Scheduling and Overload Management Real-Time Database Systems", Information Systems, 21(1), pp. 29-54, 1996.

(Delacroix et al. 2000) J. Delacroix and C. Ménival, "Intégration d'un Contrôle de Charge par Importance au sein du système RT-Linux", RTS'2000 Conference, pp. 47-63, march 2000, Paris.

(Duvallet et al. 1999) Claude DUVALLET, Zoubir MAMMERI, Bruno SADAG., « les SGBD Temps réel », edition Hermes, 1999.

(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., 1999) J. Hansson, S. F. Andler and S. H. Son, "Value-driven Multi-class Overload Management", Inter. Conf. on Real-Time Systems and Applications, dec. 1999, Hong Kong.

(Hansson et al., 2000)J. Hansson, M. Thuresson and S. H. Son,"Imprecise Task Scheduling and Overload Management using OR-ULD", Inter. Conf. on Real-Time Computing Systems and Applications}, Korea, dec. 2000.

(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.

(Kaiser et al., 1998) C. Kaiser, C. Santellani, "Pétrarque. Une Plate-forme d'Expérimentation pour l'ordonnancement temps réel strict d'applications réparties", Technique et Science Informatique Journal, 17(1), pp.39-62, 1998 (French).

(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.

(Liu et al., 1973) C. Liu and J. Leyland, "Scheduling Algorithms for Multiprogrammig in Hard Real-Time Environment", Journal of the ACM, 20(1), 1973.

(Mok et al., 1997) A. K. Mok and D. Chen, "A multiframe model for real-time tasks", IEEE transactions on Software Engineering, 23(10), p. 635-645, 1997.

(Ramamritham 1993) Ramamritham K., "Real-time databases", J. of Distributed and Parallel Databases, 1(2), pp. 199-226, 1993.

(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.

(Samaras et al., 1995) G. Samaras et al., "Two-Phase Commit Optimization in a Commercial Distributed Environment", Journal of Distributed and Parallel Databases, 3(4), 1995.

(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.

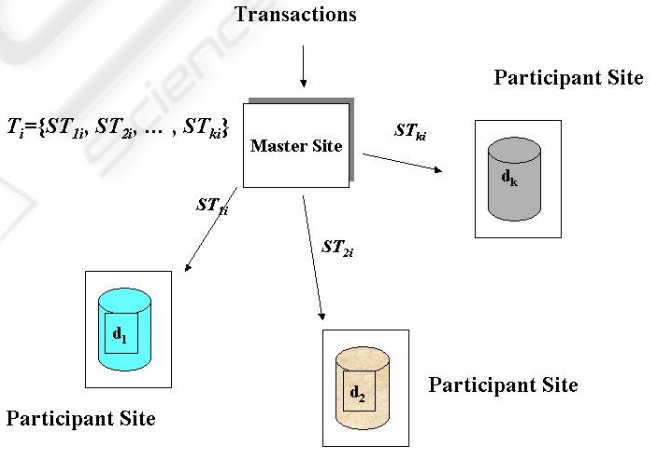

Figure 1: The transactional model

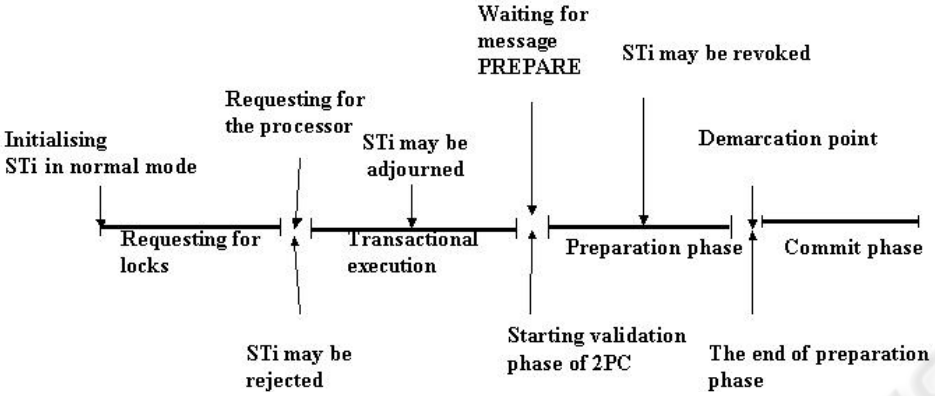

Figure 2: The life time of a multimode subtransaction STi

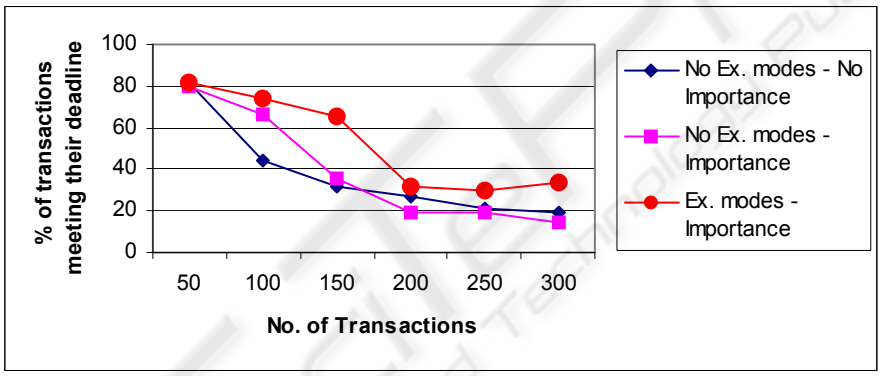

Figure 3: Percentage of transactions meeting their deadline in different situations