UNIVERSITE DU HAVRE
UFR DES SCIENCES ET TECHNIQUES
Laboratoire d'Informatique du Havre (LIH EA 3219)

Habilitation à Diriger des Recherches

Spécialité : Informatique

Contributions à la gestion des transactions dans les SGBD temps réel

Présentée par :

BRUNO SADEG

devant le jury composé de :

ALEXANDRE BERRED, Professeur, LMAH, Université du Havre, examinateur
CHRISTINE COLLET, Professeur, IMAG, INP Grenoble, examinateur
FRANCIS COTTET, Professeur, LISI/ENSMA, Futuroscope, rapporteur
MOHAND-SAID HACID, Professeur, LIRIS, Université Lyon 1, rapporteur

ROBERT LAURINI, Professeur, LIRIS, INSA Lyon, président

MICHEL MAINGUENAUD, Professeur, PSI, INSA Rouen, examinateur
ZOUBIR MAMMERI, Professeur, IRIT, Université Paul Sabatier (Toulouse 3), examinateur
CHANTAL SOULÉ-DUPUY, Professeur, IRIT, Université Toulouse 1, rapporteur

26 Novembre 2004

Table des matières

Liste des figures

Introduction générale

Depuis l'avènement des ordinateurs et de l'informatique, des progrès très importants ont été accomplis aussi bien au niveau du matériel que du logiciel. Par exemple, la vitesse des processeurs double presque tous les 18 mois. Les mémoires principale et secondaire, qui étaient un frein important au développement d'applications avec beaucoup de calculs et de grands volumes de données, n'est presque plus un problème et on raisonne en giga-octets à propos de mémoire principale et en téra-octets en ce qui concerne la mémoire secondaire.

En termes de logiciels, de grands progrès ont également été effectués. Ils sont relatifs à l'amélioration de la gestion des applications, qui deviennent de plus en plus sophistiquées : miniaturisation, distribution sur plusieurs sites, travail dans des environnements non filaires et mobiles, manipulation de grands volumes de données, hétérogénéité, prise en compte de contraintes temporelles auxquelles sont assujetties les actions et les données, aspect réactif des systèmes, etc.

Depuis un peu plus d'une décennie, un nouveau défi s'était présenté aux chercheurs en informatique. Il s'agit de gérer les nouvelles applications, caractérisées à la fois par la manipulation de quantités importantes de données de différentes natures, et par des contraintes temporelles auxquelles peuvent être assujetties ces données et les actions qui les manipulent. Les actions ainsi que les données peuvent posséder des échéances (représentées par des intervalles de validité, dans le cas des données).

Des travaux ont été menés et d'autres sont en cours sur des techniques qui puissent garantir à la fois le respect des contraintes temporelles et des contraintes logiques des tâches critiques[1], et l'amélioration de la qualité des résultats obtenus avec les autres types de tâches (depuis quelques années, on parle de qualité de service - QdS). Nos travaux entrent dans le domaine d'étude de la QdS, puisque l'on s'intéresse à l'un de ses aspects : la qualité des transactions et la qualité des données.

Des recherches ont été initiées vers la fin des années 80 sur des systèmes qui puissent gérer de manière efficace ce genre d'applications : ce sont les systèmes de gestion de bases de données temps réel ($SGBDTR$). L'objectif de ces systèmes est, à la fois, de maintenir la cohérence logique des bases de données (respect des contraintes d'intégrité) et de garantir le respect des contraintes temporelles (respect des contraintes temporelles individuelles des transactions et utilisation des données temporelles durant leurs intervalles de validité). Pour parvenir à développer ce type de systèmes, il est nécessaire de s'inspirer des systèmes existants. Il s'agit en particulier de s'inspirer des systèmes de gestion de bases de données ($SGBD$) pour la structuration et la gestion au sens large des données (cohérence, reprise après panne, manipulation, ...). Pour que les transactions puissent respecter leurs échéances et utiliser des données durant leurs intervalles de validité, il est naturel d'exploiter les mécanismes que les systèmes temps réel utilisent pour respecter les contraintes temporelles des tâches, et trouver les moyens de les adapter à la gestion des transactions dans les SGBDTR.

En raison de la nouveauté du domaine, en particulier en France, nous faisons dans ce document quelques rappels généraux sur la gestion des tâches dans les systèmes temps réel et quelques rappels généraux sur la gestion des transactions dans les SGBD.

Nous constatons que beaucoup d'applications actuelles ont souvent besoin d'offrir une garantie absolue (100%) de respect des échéances pour les tâches critiques. Cela requiert une flexibilité pour fonctionner dans des environnements non déterministes. Ce besoin passe également par l'incorporation de composants logiciels basés sur des algorithmes complexes. C'est le cas des SGBDTR qui sont des

[1] Ayant des cons´equences sur l'Homme ou sur son environnement socio-´economique, etc.

systèmes imprévisibles, notamment à cause des accès non maîtrisés aux données. C'est l'objet du travail présenté dans ce document.

Les applications ayant un besoin de traitement temps réel des données prennent de plus en plus de place dans notre environnement. Les premières applications dites *temps réel* sont apparues principalement dans le domaine de l'informatique industrielle comme le contrôle de processus. Mais, avec le développement fulgurant de l'informatique, de nouvelles applications ont vu le jour où les volumes de données manipulées deviennent de plus en plus importants et qui nécessitent des traitements en temps réel (avec des contraintes temporelles, souvent sous forme d'échéances).

Les systèmes temps réel *traditionnels* disposent de mécanismes adéquats pour gérer les contraintes temporelles. Les applications sont modélisées sous forme de tâches ayant des caractéristiques précises. À partir d'un modèle de tâches particulier, des algorithmes d'ordonnancement de tâches sont élaborés de telle manière que si certaines conditions sont satisfaites, alors la configuration de tâches sera ordonnançable, c'est-à-dire que chaque tâche va respecter son échéance.

Dans ces systèmes, la principale ressource gérée est le processeur (temps CPU). Les données sont peu nombreuses et l'accès à une donnée s'effectue sous forme de section critique (gardée par des sémaphores).

Les SGBD traditionnels de leur côté sont très efficaces pour la gestion et la structuration de quantités importantes de données, mais leur objectif est de minimiser le temps de réponse moyen des transactions (DeWitt, 1985; Gray, 1993) ; les temps d'exécution des transactions individuelles sont indéterminés. Donc, il peut arriver que des transactions soient exécutées durant une durée très longue. Ce qui n'est pas acceptable dans beaucoup d'applications temps réel.

Dans les applications actuelles, il est souvent nécessaire de garantir le respect des échéances individuelles de toutes les transactions. Malheureusement, cet objectif est difficile, voire impossible, à atteindre dans l'état actuel de la technologie. Un autre critère de performance a alors été proposé : la maximisation du nombre de transactions qui respectent leurs échéances[2]. Réussir à concevoir un système qui permet de connaître à l'avance le nombre de transactions qui respectent leurs échéances, sera une avancée importante dans ce domaine.

C'est dans ce contexte que sont nées les recherches sur les SGBD temps réel, qui sont des systèmes qui doivent à la fois gérer de grands volumes de données et tenir compte des échéances individuelles des transactions (et de celles des données[3]).

Plusieurs problèmes se posent lors de la conception et de la réalisation de ces systèmes. L'objectif de nos travaux est d'appréhender le problème général de la gestion des transactions dans les SGBDTR et de proposer des solutions à certains types de problèmes moyennant certaines hypothèses.

Nous commencerons par discuter dans la suite de ce rapport de plusieurs problèmes qui se sont posés et qui se posent dans les SGBDTR, en particulier celui de la gestion des transactions. Puis nous décrirons notre contribution, notamment en ce qui concerne les points suivants :

– La relaxation de certaines propriétés ACID des transactions traditionnelles, en appliquant des critères relaxant la sérialisabilité.

[2] Ou la minimisation du nombre de transactions ayant manqu´e leurs ´ech´eances.
[3] Exprim´ees sous forme de dur´ees de validit´e.

- La proposition des concepts d'*epsilon-donnée* et de *delta-échéance* pour classifier les différents types d'applications temps réel gérées par les SGBDTR, puis la proposition de protocoles de contrôle de concurrence et d'ordonnancement des transactions temps réel, en fonction des valeurs de delta et de epsilon.
- L'application de ces notions dans les SGBD multimédia temps réel pour tolérer la présentation de scènes audio/vidéo non parfaites, ainsi que dans les SGBDTR mobiles pour atténuer l'effet des déconnexions.
- L'étude de la validation (commit) des transactions temps réel distribuées, en particulier la proposition d'adaptation de protocoles existants pour les SGBDTR, puis la proposition de nouveaux protocoles utilisant des techniques algorithmiques employées dans les systèmes distribués : l'ordre causal.
- L'étude des transactions temps réel imbriquées.
- L'étude des SMA (Systèmes Multi-Agents) temps réel, en particulier l'utilisation de techniques *Anytime*.
- Des travaux préliminaires sur l'ordonnancement par rétroaction des transactions temps réel et l'étude probabiliste du comportement des transactions temps réel.

Pour des raisons de lisibilité, ce dossier est structuré en trois parties correspondant à trois entités logiques (documents) distinctes. Dans cette partie, seront décrits les travaux effectués au sein de l'équipe SGBDTR[4] du LIH[5] (EA 3219), sur la problématique précédente. Deux autres parties seront consacrées aux annexes.

Cette partie est décomposée en plusieurs chapitres, dont la description est donnée ci-dessous.

- Dans le chapitre 1, nous présenterons les systèmes dont sont issus les SGBDTR. Ce sont les systèmes temps réel (STR) et les systèmes de gestion de bases de données (SGBD) *traditionnels*. Nous décrirons notamment la gestion des transactions dans les SGBD et la gestion des tâches dans les STR.

- Dans le chapitre 2, nous discuterons de la problématique des SGBDTR, en décrivant brièvement les différents problèmes rencontrés lors de leur conception et de quelques solutions proposées, et nous présenterons quelques applications potentielles et quelques prototypes de recherche.

- Dans le chapitre 3, nous discuterons de la gestion des transactions temps réel.
  Après avoir rappelé la problématique et quelques résultats obtenus sur le contrôle de concurrence et l'ordonnancement des transactions, nous présenterons notre contribution dans ce domaine.

- Ensuite, nous rappellerons brièvement le problème de la validation (Commit) des transactions distribuées dans le chapitre 4. Nous présenterons quelques travaux effectués dans un contexte temps réel. Nous décrirons ensuite nos travaux sur la validation des transactions temps réel distribuées, notamment ceux exploitant des techniques de duplication de ressources ainsi que ceux utilisant l'ordre causal.

- Nous décrirons ensuite notre contribution à l'utilisation de modèles de transactions imbriquées dans un contexte temps réel (chapitre 5).

[4] Syst`emes de Gestion de Bases de Donn´ees Temps R´eel.
[5] Laboratoire d'Informatique du Havre.

– Les travaux effectués sur les algorithmes *Anytime* sont présentés dans le chapitre 6.

– Dans le chapitre 7, nous décrirons nos travaux préliminaires sur la qualité des données et la qualité des transactions dans les SGBDTR. Ces travaux exploitent des résultats sur la théorie du contrôle par rétroaction appliqués à l'ordonnancement des transactions, et utilisent la création et la manipulation de plusieurs versions des données.

– Le chapitre 8 sera consacré à la description de nos travaux sur l'étude probabiliste du comportement des transactions temps réel, et à quelques résultats préliminaires que nous avons obtenus.

– Dans le chapitre 9, nous concluons en donnant quelques reflexions globales sur la conception des SGBD temps réel et en exposant les grandes lignes de notre projet de recherche.

Les deux autres parties contiennent les annexes, et constituent deux documents séparés :

1. L'annexe 1 sera consacrée à la description de mon curriculum vitae.
2. Les textes de mes principales publications seront donnés dans l'annexe 2.

1 Introduction aux $SGBD$ et aux systèmes temps réel

Les Systèmes de Gestion de Bases de Données Temps Réel ($SGBDTR$) sont nés du rapprochement entre les systèmes de gestion de bases de données (SGBD) (Bernstein et al., 1987; Gardarin, 2000; Gray, 1993) et les systèmes temps réel (STR) traditionnels (Liu and Leyland, 1973; Elloy, 1988). Pour cette raison, dans ce chapitre, nous commencerons par décrire brièvement les caractéristiques principales de ces deux types de systèmes. Nous rappellerons en particulier des résultats connus sur la gestion des transactions dans un $SGBD$ et sur celle des tâches dans un système temps réel. La transaction étant l'unité de travail et l'unité de recouvrement dans un SGBD, et la tâche l'abstraction utilisée dans les systèmes temps réel.

1.1 Introduction aux $SGBD$

Les $SGBD$ sont apparus au début des années 1960 pour pallier les problèmes des systèmes de gestion de fichiers classiques. Depuis, quatre générations de SGBD se sont succédées : les $SGBD$ *hiérarchiques*, les $SGBD$ *réseaux*, les $SGBD$ *relationnels* et les $SGBD$ *orientés objets*[1]. Chaque génération améliore les fonctionnalités de la précédente (Date, 1985; Adiba and Collet, 1993; Gardarin, 2000; Delobel and Adiba, 1982) et certains de ces systèmes sont utilisés dans les processus décisionnels ou pour l'intégration de plusieurs bases de données sous forme d'entrepôts de données (*data warehouse*) centralisés ou distribués (Garcia-Molina et al., 1998; Bouzeghoub et al., 1999; Khrouf and Soulé-Dupuy, 2001). Les $SGBD$ les plus répandus actuellement sont de type *relationnel*.

Les $SGBD$ ont apporté beaucoup d'améliorations à la gestion des données : moyens de structuration, redondance minimisée, indépendances logique et physique, reprise après incident, facilité d'utilisation, etc. Au niveau de la conception de ces systèmes, l'apport le plus important concerne sans doute la gestion des transactions (Date, 1997). Dans la suite, nous allons passer en revue certains éléments relatifs à cet aspect des SGBD. Le lecteur intéressé peut se référer à l'abondante littérature sur le sujet (Bernstein et al., 1987; Gray and Reuter, 1993; Adiba and Collet, 1993; Gardarin, 2000; Hameurlain et al., 1996; Date, 1997).

1.1.1 Gestion des transactions dans les $SGBD$

Une transaction est composée d'une ou plusieurs opérations sur les données de la base. Dans l'étude des transactions, une opération est généralement réduite à une *lecture* ou à une *écriture* sur disque. Une transaction dans un $SGBD$ doit posséder un certain nombre de propriétés, désignées par l'acronyme ACID (Atomicité, Cohérence, Isolation et Durabilité). Ces propriétés distinguent une transaction de n'importe quelle autre séquence d'actions sur la base.

[1] Des $SGBD$ exploitant des donn´ees semi-structur´ees (utilisant XML) sont apparus (Gardarin, 2002; Milo et al., 2003; Gardarin et al., 2002; Hacid et al., 2001).

– Atomicité : la propriété d'atomicité signifie qu'une transaction est considérée comme une unité atomique d'exécution, c'est-à-dire que toutes les opérations de la transaction doivent être exécutées. Si une des opérations n'a pas pu s'exécuter alors toute la transaction est annulée.

– Cohérence : la propriété de cohérence stipule que, étant donnée une base de données cohérente, l'exécution d'une transaction fait passer la base dans un autre état cohérent.

– Isolation : les transactions sont dites isolées les unes des autres, c'est-à-dire que les modifications effectuées par l'une d'elles ne deviennent visibles et exploitables par les autres transactions qu'après sa phase de validation (Commit).

– Durabilité : la propriété de durabilité indique qu'une fois la transaction validée (Commit), les modifications qu'elle a effectuées sur la base pendant son exécution deviennent permanentes, même si un incident survient après sa validation.

Parmi les propriétés $ACID$, le maintien des propriétés AID est du ressort du SGBD, alors que la propriété C (cohérence), elle est assurée non seulement par le $SGBD$, mais également par l'utilisateur, qui doit avoir programmé correctement sa transaction.

Une transaction a une marque de début et une marque de fin. Sa fin peut être négative (annulation, abandon, abort ou rollback) ou positive (validation, confirmation, engagement ou commit). Dans la suite, nous utiliserons indifféremment les termes précédants mis entre parenthèses pour une fin négative ou positive respectivement, sauf indication contraire.

Un des problèmes posés par la gestion des transactions est le contrôle des accès simultanés aux mêmes données de la base par plusieurs transactions, pour éviter les accès dans des modes incompatibles[2]. En effet, si les transactions s'exécutaient dans un $SGBD$ sans contrôle, différents types de problèmes pourraient survenir comme la perte de mise à jour, les lectures inconsistantes, les lectures non renouvelables, l'analyse incohérente ou les lectures fantômes. Le lecteur intéressé pourra consulter (Date, 1997; Gardarin, 2003) pour davantage de détails sur ces problèmes.

La solution évidente à ces problèmes est de verrouiller les objets accédés jusqu'à ce que la transaction se termine. Cependant, dans un souci de performance, il est intéressant de laisser s'exécuter le maximum de transactions en concurrence, tout en veillant à ne pas remettre en cause la cohérence de la base. Pour cela, des protocoles dits de *contrôle de concurrence* ont été proposés. La plupart de ces protocoles ont comme idée de base de veiller à ce que l'exécution des transactions concurrentes soit *sérialisable*. La *sérialisabilité* est le critère généralement accepté pour l'exécution correcte des transactions dans un SGBD. La sérialisabilité de deux transactions peut être définie de la manière suivante (Bernstein et al., 1987) : *L'exécution de deux transactions est dite sérialisable si l'exécution entrelacée de leurs opérations fournit les mêmes résultats que leur exécution en série (l'une après l'autre*).

Plusieurs classes de protocoles de contrôle de concurrence ont été proposées dans la littérature. Dans la suite, nous allons donner un bref panorama de ces protocoles, étant donné que beaucoup d'entre-eux ont servi de base à des protocoles de contrôle de concurrence pour les transactions temps réel.

[2]Lecture/Ecriture, Ecriture/Lecture ou Ecriture/Ecriture

1.1.2 Protocoles de contrôle de concurrence

1.1.2.1 Protocoles de verrouillage à deux phases (2PL)

Parmi les protocoles de contrôle de concurrence (CC) des transactions dans les SGBD, il y a la classe des protocoles basés sur le verrouillage. Le principal représentant de cette classe est le protocole 2PL[3] (Verrouillage à 2 phases) (Bernstein et al., 1987; Gardarin, 2003; Date, 1997). Il s'agit du protocole le plus étudié et le plus implémenté dans les $SGBD$ commerciaux. Le protocole 2PL est basé sur l'utilisation de verrous. Deux principaux types de verrous sont utilisés, qui ont servi de base à d'autres types de verrous comme les verrous intentionnels (Date, 1997) :

- Les verrous partagés (ou en lecture) : dénotés par S (*Share*).
- Les verrous exclusifs (ou en écriture) : dénotés par X (*eXclusive*).

Le protocole $2PL$ stipule que, lors de son exécution, une transaction passe par deux phases :

- l'acquisition de verrous : au cours de cette phase, la transaction acquiert les verrous au fur et à mesure de ses besoins d'accès aux données,
- la libération de verrous : en général pendant sa phase de validation (Commit), la transaction libère ses verrous et ne peut plus en acquérir d'autres.

Le protocole 2PL constitue une *condition suffisante* pour qu'une exécution de transactions soit sérialisable. Donc, si l'exécution des transactions respecte les processus précédents d'acquisition et de libération des verrous, alors l'exécution de ces transactions est sérialisable et la base de données est exempte d'incohérence.

Plusieurs variantes du protocole $2PL$ ont été proposées pour améliorer les performances des $SGBD$, dont certaines sont implémentées dans des $SGBD$ commerciaux. Parmi ces variantes, certaines définissent d'autres types de verrous comme les verrous intentionnels, d'autres libèrent certains types de verrous avant la validation (Commit) de la transaction, etc. (Bernstein et al., 1987).

Le protocole 2PL et ses variantes sont dits *pessimistes*, car ils partent de l'hypothèse que dans le SGBD, des conflits potentiels sont toujours possibles dès lors que des transactions accèdent simultanément aux données. Ils imposent donc la pose de verrous sur les objets auxquels accèdent les transactions avant toute utilisation de ces objets.

Nous rappelons dans la table de la figure 1.1 la matrice de compatibilité des verrous classiques (modes de verrouillage S et X). Dans cette table, le symbole '—' représente l'absence de verrou, O désigne la possibilité d'acquérir le verrou et N désigne l'interdiction d'acquérir le verrou.

L'un des avantages de ce protocole est sa facilité d'implémentation. C'est la raison principale de sa large utilisation. Un autre avantage est qu'il permet de conserver l'exclusivité d'un objet aussi longtemps que nécessaire (Date, 1997). La cohérence des données manipulées est ainsi renforcée. En effet, l'usage d'une donnée verrouillée est soumis à l'approbation du protocole, qui décide, selon la matrice de compatibilité, de donner l'accès ou non à d'autres transactions qui en font la demande.

Ce protocole présente également des inconvénients non négligeables. En effet, il s'agit d'un protocole bloquant : deux transactions peuvent indéfiniment s'attendre mutuellement à cause de leurs verrous

[3] Two-Phase Locking

Verrou demand´e \ Verrou d´etenu	X	S	—
X	N	N	O
S	N	O	O
—	O	O	

TAB. 1.1 – Matrice de compatibilité des verrous partagé et exclusif

respectifs (verrous mortels). L'interblocage peut être direct ou indirect. Soit $(T_i)_{i=1,n}$ des transactions. L'interblocage est direct si T_i attend T_j, qui, de son côté, attend T_i. L'interblocage est indirect si T_1 attend T_2, qui attend T_3, ... , qui attend T_1. On dit que l'on a un cycle dans le *graphe d'attente* des transactions. Des méthodes existent pour prévenir ou *guérir* les interblocages (Gardarin, 2003; Date, 1997).

1.1.2.2 Protocoles par estampillage

Le protocole par estampillage (Date, 1997) stipule que l'ordre d'exécution des transactions est donné par leurs estampilles. L'estampille d'une transaction est un nombre entier positif affecté à une transaction dès son arrivée dans le système. Il correspond à l'ordre chronologique d'arrivée de la transaction. Notons que dans le protocole 2PL, l'ordre d'exécution des transactions était donné en fonction de l'ordre d'accès des transactions aux objets de la base.

Basé sur les estampilles des transactions, un protocole de contrôle de concurrence a été proposé. La résolution des problèmes de conflit d'accès aux données par plusieurs transactions s'effectue en se servant des estampilles. Lorsque des transactions rentrent en conflit d'accès à une donnée dans un mode incompatible, la transaction "vainqueur"[4] est la transaction de plus petite estampille (la plus ancienne). Les autres transactions sont abandonnées puis redémarrées Les transactions redémarrées se voient affecter de nouvelles estampilles.

L'avantage de ce protocole est, contrairement au verrouillage, l'absence de problème de blocage. Cependant, il possède au moins deux inconvénients, qui sont le *livelock* (une transaction peut être sans cesse redémarrée) et le problème d'abandon en cascade (quand une transaction est abandonnée, toutes les transactions qui ont utilisé les valeurs qu'elle a mises à jour seront également abandonnées).

1.1.2.3 Protocoles optimistes (par certification)

Des protocoles avec une approche optimiste, parfois désignés par OCC, ont également été proposés. Cette approche part de l'hypothèse optimiste que, si dans un SGBD, plusieurs transactions s'exécutent en concurrence, il n'existe qu'une faible probabilité pour qu'elles rentrent en conflit d'accès aux mêmes données. Dans les protocoles optimistes, les transactions s'exécutent entièrement dans leur espace local. Les conflits éventuels sont détectés lorsqu'elles passent le test de *certification*.

Il existe deux principales stratégies de certification (Kung and Robinson, 1981) :

– *Certification en arrière* : lorsqu'une transaction arrive à son terme, le test pour détecter les conflits éventuels s'effectue par rapport aux transactions récemment validées,

[4] celle qui sera autoris´ee `a acc´eder `a la donn´ee

– *Certification en avant* : lorsqu'une transaction arrive à son terme, le test pour détecter les conflits éventuels s'effectue par rapport aux transactions en cours d'exécution.

Avec le protocole par certification, une transaction passe par trois phases durant son exécution, quelle que soit la stratégie. Les trois phases sont les suivantes (Kung and Robinson, 1981) :

– Lecture : elle consiste à lire les données de la base de données, faire une copie dans son espace local et exécuter les opérations nécessaires. Les opérations de mise à jour sont enregistrées dans un fichier temporaire de mise à jour.
– Certification : elle consiste à tester la transaction pour vérifier si elle respecte l'intégrité et la cohérence de la base, i.e. si elle est sérialisable avec les transactions récemment validées ou avec les transactions actives, selon le type de certification. D'après le résultat de ce test, la transaction est, soit validée (et passe à la dernière phase vu qu'elle n'est en conflit avec aucune autre transaction), ou bien elle est abandonnée puis redémarrée.
– Écriture : elle applique les changements de manière permanente sur la base de données.

Les méthodes optimistes présentent de meilleures performances que les méthodes par verrouillage dans les $SGBD$ où les transactions rentrent peu en conflit d'accès aux données. Il y a alors peu de redémarrages. Cependant, lorsque les conflits deviennent nombreux, les limites de ce type de protocoles apparaissent rapidement.

Nous avons vu brièvement comment s'effectue la gestion des conflits d'accès aux données par des transactions dans les $SGBD$ centralisés. Nous avons, en particulier, passé en revue quelques techniques de contrôle de concurrence proposées pour gérer ces conflits.

1.1.3 SGBD et contraintes temporelles

Depuis l'apparition des SGBD, en plus des différentes génération de systèmes citées précédemment, il existe des $SGBD$ plus spécialisés. Il y a les $SGBD$ temporels (Snodgrass, 1987; Tansel et al., 1993), qui permettent de gérer les historiques des données, i.e. chaque donnée possède plusieurs versions datées. Les transactions sollicitent ensuite les versions des données en fonction de certains critères tels que la disponibilité.

Il y a également les $SGBD$ actifs (Dayal, 1989; Coupaye and Collet, 1998), qui sont des systèmes qui généralisent les mécanismes de *triggers* qui existent dans les $SGBD$ traditionnels. Généralement, le mécanisme qui implémente l'aspect actif de ces $SGBD$ se présente sous forme de règles, dites *règles actives*, qui ont la forme suivante :

QUAND *un événement survient,*
SI *une condition est satisfaite,*
ALORS *déclencher une action (ou plusieurs).*

Ces règles sont appelées règles ECA (Événement-Conditions-Actions).

Cependant, aucun $SGBD$ des types cités précédemment ne prend en compte les contraintes temporelles, qui consistent principalement à exécuter des transactions ayant des échéances et/ou manipulant des données ayant des intervalles de validité. Et comme on le verra dans le chapitre 2, il est nécessaire de concevoir un nouveau type de $SGBD$ qui puisse satisfaire ces contraintes : ce sont les $SGBD$ temps réel.

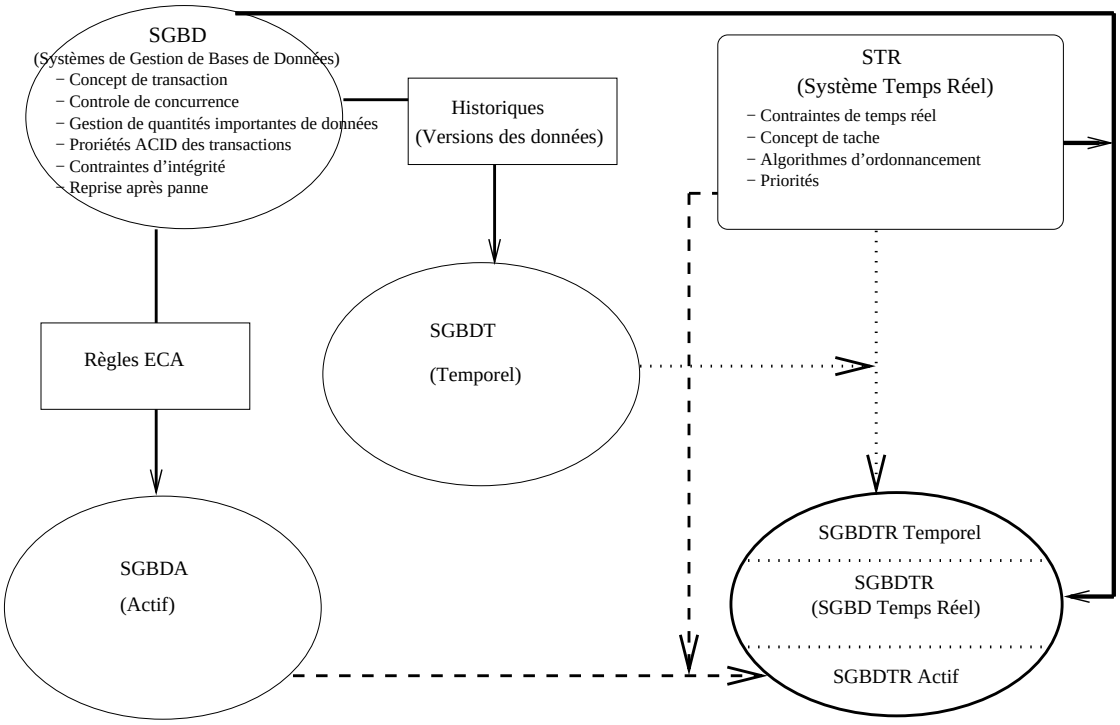

FIG. 1.1 – Des SGBD et STR vers les SGBDTR

Le schéma de la figure 1.1 illustre une manière dont les SGBD et les systèmes temps réel intègrent les différents composants relatifs aux règles actives, aux aspects temporels (au sens *historiques* du terme) et aux contraintes temps réel, pour aboutir à différents systèmes.

Dans la section 1.2, nous allons présenter brièvement les caractéristiques principales des systèmes temps réel. Ensuite, nous détaillerons un mécanisme important spécifique à ces systèmes : l'ordonnancement des tâches par priorités. Quelques protocoles seront présentés, qui se basent sur un certain nombre de paramètres : périodicité des tâches, algorithmes en ligne ou hors ligne, statiques ou dynamiques, etc.

Les protocoles de CC dans les $SGBD$ vus précédemment, combinés aux protocoles d'ordonnancement de tâches temps réel, sont utilisés pour concevoir des protocoles de contrôle de concurrence et d'ordonnancement (CC&O) des transactions dans les $SGBDTR$, ou simplement *transactions temps réel*. Quelques uns de ces protocoles seront présentés dans le chapitre 3.

1.2 Introduction aux systèmes temps réel (STR)

1.2.1 Généralités sur les STR

Le temps joue un rôle de plus en plus important dans les services et les applications actuelles (informatique industrielle, télécommunications, multimédia, ...). La maîtrise du temps a d'abord commencé par plus de puissance de calcul dans les ordinateurs, puis par l'apparition de mécanismes "légers" (moins de temps gaspillé pour des parties non liées à l'application), et enfin par l'apparition de méthodes pour prédire le comportement temporel des applications. Plusieurs mécanismes à mettre en oeuvre dans un système temps réel (STR)[5] ont été proposés, dont le principal est l'ordonnancement des tâches qui composent l'application (Cottet et al., 2000; Bonnet and Demeure, 1999).

[5]qui supporte les applications temps r´eel

On ne peut pas parler de temps réel sans donner la définition suivante, qui résume ce que doit être un système temps réel : *c'est un système dont le comportement dépend non seulement de l'exactitude des traitements effectués, mais également de l'instant où les résultats de ces traitements sont fournis. Dans ces systèmes, un retard dans la production d'un résultat est considéré comme une erreur, pouvant souvent entraîner de graves conséquences (Stankovic, 1988; Kavi, 1992)*.

On distingue, en général, deux principaux types de tâches temps réel : (1) à échéances dures ou strictes (*hard*) : la tâche qui ne se termine pas avant son échéance peut provoquer une exception dans le système, et peut avoir de graves conséquences sur l'application, (2) à échéances molles, lâches ou relatives (*soft*), où le retard d'une tâche ne provoque pas d'exception, c-à-d que les tâches de ce type peuvent manquer leurs échéances occasionnellement sans engendrer de graves conséquences.

Le déterminisme est le but à atteindre pour tout système temps réel, pour assurer la prévisibilité, i.e., enlever toute incertitude sur le comportement des activités (individuelles et mises ensemble). Dans un STR dur (à contraintes strictes), on cherche à ce que toutes les échéances soient respectées, alors que dans un STR mou (à contraintes relatives), on cherche à minimiser le retard moyen des activités, ou maximiser le nombre de tâches qui respectent leurs échéances par exemple.

Les systèmes classiques ont des limites qui les rendent inutilisables pour le temps réel car ils reposent sur un SE (Système d'Exploitation) qui offre des mécanismes mal adaptés au temps réel, par exemple :

- Les politiques d'ordonnancement visent le partage équitable du temps d'exécution. L'équité est mal adaptée quand il y a des tâches plus critiques que d'autres.
- Les mécanismes d'accès aux ressources partagées sont à adapter pour éliminer les incertitudes temporelles.
- La gestion des E/S (entrées/sorties) engendrent de longues attentes (parfois non bornées).
- La gestion des interruptions n'est pas optimisée.
- Les mécanismes de gestion de la mémoire virtuelle sont à revoir (la pagination, le *swapping*, ...).
- Les temporisateurs qui organisent le temps n'ont pas une granularité assez fine (pour beaucoup d'applications temps réel).

Tous ces problèmes ont poussé à la proposition de systèmes qui puissent tenir compte de la dimension *temps* dans la gestion des applications : ce sont les systèmes temps réel. Le mécanisme le plus important dans ces systèmes est l'ordonnancement des tâches, c-à-d l'exécution des tâches par ordre de priorités de telle manière que les tâches critiques puissent respecter leurs échéances, et à minimiser le retard ou maximiser le nombre de tâches non critiques qui s'exécutent avant échéance. Nous résumons dans le paragraphe suivant les principaux algorithmes d'ordonnancement de tâches temps réel. Une très bonne introduction sur le sujet peut être trouvée dans (Cottet et al., 2000).

1.2.2 Gestion des tâches dans les STR

Une application temps réel est modélisée sous forme de tâches. Les caractéristiques temps réel d'une tâche peuvent être schématisées comme le montre la figure 1.2. La terminologie utilisée pour l'ordonnancement des tâches temps réel est la suivante :

- La tâche : il s'agit de l'unité de base de l'ordonnancement temps réel. Les tâches peuvent être périodiques, sporadiques ou apériodiques, à contraintes temporelles strictes ou relatives.

– Le modèle de tâches : il peut comporter des paramètres chronologiques (des dates) ou chronométriques (des périodes). Les paramètres de base d'une tâche sont :

. r : sa date de réveil, moment de déclenchement de la 1^{ere} requête d'exécution (r_0, r_1, ... pour une tâche périodique)
. C : sa durée maximale d'exécution (si elle dispose du processeur à elle seule)
. D : son délai critique (délai maximal acceptable pour son exécution)
. P : sa période (si la tâche est périodique)

Remarque : Si la tâche est à contraintes strictes, alors l'échéance $d_i = r_i + D$, représente la date dont le dépassement entraîne une faute temporelle.

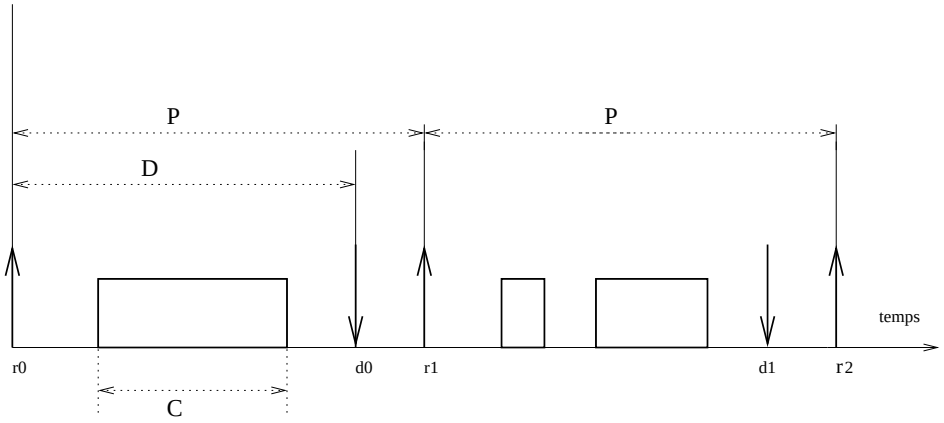

FIG. 1.2 – Caractéristiques d'une tâche temps réel

1.2.3 Quelques algorithmes d'ordonnancement de tâches

Comme nous l'avons dit précédemment, l'ordonnancement des tâches est le mécanisme fondamental dans les systèmes temps réel. L'une des contributions les plus significatives dans le domaine est celle effectuée par Liu et Layland (Liu and Leyland, 1973) qui ont développé des algorithmes d'ordonnancement statiques et dynamiques pour les tâches temps réel strictes critiques (*hard*), qui sont optimaux sous certaines conditions.

Pour tester la qualité d'un algorithme d'ordonnancement, des tests d'ordonnançabilité sont effectués. Dans un système temps réel, étant donnés un ensemble de tâches avec leurs caractéristiques d'une part, et un algorithme d'ordonnancement d'autre part, l'objectif du test d'ordonnançabilité est de prédire à l'avance si toutes les tâches vont respecter leurs échéances. Nous verrons dans la suite les conditions nécessaires et/ou suffisantes d'ordonnançabilité des tâches périodiques.

1.2.3.1 Quelques éléments caractérisant un ordonnancement

Un ordonnancement temps réel doit posséder les caractéristiques suivantes :

– L'ordonnancement doit permettre le respect des contraintes temporelles des tâches d'une application quand elle s'exécute en régime courant (charge normale).

– L'ordonnancement doit être vérifiable, c-à-d que l'on doit pouvoir prouver a priori le respect des contraintes temporelles des tâches d'une application (en régime courant).

– L'ordonnancement doit offrir une certaine tolérance en régime de surcharge (permettre une exécution dégradée mais sécuritaire du système). Le régime de surcharge survient lorsque des tâches supplémentaires arrivent suite à des anomalies (alarmes, variations des temps d'exécution) et provoquent des fautes temporelles.

– Configuration de tâches : c'est la mise en jeu d'un ensemble de n tâches qui s'exécutent.

– Tâches à départ simultané : ce sont des tâches qui ont la même date de réveil. Dans le cas contraire, on parle de tâches à départ échelonné.

– Le facteur d'*utilisation* du processeur pour n tâches périodiques :

$$U = \sum_{i=1}^{i=n} \frac{C_i}{P_i}$$

– Le facteur de *charge* du processeur pour n tâches périodiques :

$$CH = \sum_{i=1}^{n} \frac{C_i}{D_i}$$

– La laxité du processeur à l'instant t, $LP(t)$, représente l'intervalle de temps à partir de t pendant lequel le processeur peut rester inactif sans remettre en cause le respect des échéances ($LP(t) \geq 0, \forall\, t$). $LP(t)$ est égale au minimum des laxités conditionnelles $LC_i(t)$ des tâches i.

– $LC_i(t)$ = D_i - $\sum_j C_j(t)$. Les tâches j sont celles qui sont déclenchées à l'instant t et qui devancent i dans la séquence de planification.

1.2.3.2 Typologie des algorithmes d'ordonnancement

Les ordonnancements classiques ont pour objectifs :

– de maximiser le taux d'utilisation du processeur et minimiser le temps de réponse moyen des tâches (durée séparant l'instant de soumission au système de l'instant de fin d'exécution), comme pour les transactions dans les $SGBD$ traditionnels, ou
– de respecter d'autres critères : minimiser la durée moyenne d'attente des tâches (état *prêt*), augmenter le débit du processeur (nombre de tâches traitées par intervalle de temps), minimiser la moyenne du temps de réponse d'un ensemble de tâches, etc.

On distingue plusieurs types d'algorithmes d'ordonnancement de tâches : hors ligne, en ligne, statiques, dynamiques, préemptifs, non préemptifs, distribués ou non.

. Algorithmes hors ligne
Ils construisent la séquence complète de planification des tâches sur la base de tous les paramètres temporels des tâches. Cette séquence est connue avant l'exécution. Cette approche est très efficace mais, comme elle est statique, elle est rigide. En effet, elle suppose que tous les paramètres, y compris les dates de réveil, sont figés. Elle ne s'adapte donc pas aux changements de l'environnement.

. Algorithmes en ligne
Ils sont capables à tout instant de l'exécution d'une application de choisir la prochaine tâche à ordonnancer, en utilisant les informations des tâches déclenchées à cet instant. Ce choix peut être remis en cause par l'occurrence d'un nouvel événement. Cette approche offre des solutions moins bonnes (car elle utilise moins d'informations) et entraîne des surcoûts de mise en oeuvre, mais elle présente un avantage non négligeable : elle permet de tenir compte de l'arrivée imprévisible des tâches. Elle autorise donc la construction progressive de la séquence d'ordonnancement.

Remarque : Pour pouvoir traiter les tâches apériodiques (et les surcharges anormales), l'ordonnancement se fait souvent en ligne.

. Algorithmes préemptifs
Ils permettent à l'ordonnanceur de déposséder la tâche *élue* du processeur au profit d'une autre tâche jugée plus prioritaire. Il ne sont utilisables que si toutes les tâches sont préemptibles. La tâche ayant perdu le processeur passe à l'état *prêt* pour être *élue* ultérieurement sur le même processeur ou sur un autre.

. Algorithmes non préemptifs
Ils n'arrêtent pas l'exécution d'une tâche *élue*. Il peut en résulter des fautes temporelles qu'un algorithme préemptif aurait évitées.

. Meilleur effort ($best\ effort$)
Dans cette approche, l'algorithme essaie de faire au mieux avec les processeurs disponibles.

. Ordonnancement centralisé
Si l'algorithme d'ordonnancement s'exécute sur une architecture centralisée (ou sur un site privilégié d'une architecture distribuée, qui contient tous les paramètres des tâches).

. Ordonnancement distribué
Lorsque les décisions d'ordonnancement sont prises sur chaque site par un ordonnanceur local après une éventuelle coopération pour effectuer un ordonnancement global (Si l'ordonnancement est global, peuvent intervenir le placement des tâches sur les sites et la migration de tâches d'un site vers un autre).

Remarque : Pour une application à contraintes relatives, la stratégie d'ordonnancement est celle du meilleur effort. Pour une application à contraintes strictes, l'ordonnanceur doit garantir le respect des contraintes temporelles. Il y a obligation de réussite et non-tolérance aux fautes temporelles (Cottet et al., 2000).

1.2.3.3 Principales politiques d'ordonnancement classique

Elles servent à déterminer quelle tâche *prête* doit être *élue* (qui disposera du processeur).

– 1. Premier arrivé, premier servi : les tâches de faible temps d'exécution sont pénalisées si elles sont précédées dans la file d'attente par une tâche de plus longue durée.

– 2. Plus court d'abord : il remédie à l'inconvénient précédent. Il minimise le temps de réponse moyen. Cependant, il pénalise les travaux longs. Cette méthode impose d'estimer les durées

d'exécution des tâches (difficiles à obtenir).

- 3. Temps restant le plus court d'abord : la tâche en exécution restitue le processeur lorsqu'une nouvelle tâche ayant un temps d'exécution inférieur à son temps d'exécution restant devient *prête*.

- 4. Tourniquet ($Round\ Robin$) : on définit un quantum de temps (10ms à 100ms, par exemple). Chaque tâche de la file acquiert le processeur pendant au maximum un quantum de temps, puis le cède à la suivante dans la file, etc. Cette méthode est utilisée souvent dans les systèmes à temps partagé. Ses performances dépendent du quantum de temps : s'il est trop grand, les temps de réponse sont rallongés, s'il est trop petit, les commutations de contextes deviennent nombreuses.

- 5. Priorités constantes : des priorités sont affectées aux tâches et, à un instant donné, la tâche *élue* est celle qui a la plus forte priorité. Les tâches de faible priorité peuvent donc ne pas disposer du processeur (famine). Une solution est de faire "vieillir" la priorité des tâches en attente (à augmenter en fonction du temps d'attente). Mais, la priorité devient ainsi variable et on n'est plus dans le cas des algorithmes à priorités constantes.

- 6. Files de priorités constantes multi-niveaux : on définit plusieurs files de tâches *prêtes*. Chaque file correspond à un niveau de priorité (n files de priorités variant de 0 à $n-1$: la file i pour les tâches de même priorité). Les files sont gérées soit par ancienneté, soit par tourniquet. Le quantum peut être différent selon la file. L'ordonnanceur sert d'abord les tâches de la file 0, puis celles de la file 1 (dès que la file 0 est vide), etc. Il existe deux variantes :

  . Les priorités des tâches sont constantes tout au long de leur exécution : une tâche en fin de quantum est réinsérée dans la même file.

  . Les priorités des tâches évoluent dynamiquement en fonction des services dont elles ont déjà bénéficiés. Une tâche de la file i, qui n'a pas terminé son exécution à la fin de son quantum, est réinsérée dans la file $i+1$ (moins prioritaire), etc.
  On minimise ainsi les risques de famine des tâches de faible priorité en diminuant au fur et à mesure de l'exécution les priorités des tâches ayant de fortes priorités initiales.

- Remarque : aucune des politiques présentées précédemment ne permet de remplir les deux objectifs d'un ordonnancement temps réel. Notamment, elles ne prennent pas en compte l'urgence des tâches (leur délai critique).

1.2.3.4 Ordonnancement de tâches temps réel indépendantes

Dans un premier temps, nous présentons quelques algorithmes d'ordonnancement de tâches temps réel périodiques. Les tâches ont certaines caractéristiques : elles sont indépendantes et chaque tâche est composée d'occurrences (instances ou *jobs*), qui surviennent à chaque date de réveil. Ensuite, nous présentons quelques algorithmes d'ordonnancement de tâches périodiques indépendantes en présence de tâches apériodiques.

(i) **Tâches périodiques seules**

– Ordonnancement en ligne, à priorité constante

1. *Rate Monotonic* (ou RM) (Liu and Leyland, 1973) :

   La priorité d'une tâche est fonction de sa période : la tâche de plus petite période est la plus prioritaire. L'algorithme est optimal dans la classe des algorithmes à priorité constante pour une configuration de tâches à échéances sur requête (dont l'échéance est égale à la période). Dans ce cas, une **condition suffisante** d'existence d'une borne minimale pour l'acceptation d'une configuration de n tâches est donnée par la formule :

   $$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$$

   Cette borne représente le pire cas et tend vers ln 2 (69%) quand n est très grand. Elle peut donc être dépassée (en moyenne elle est de 88%)
   Nous donnerons une illustration graphique de ce protocole. Pour les autres protocoles, le lecteur intéressé peut consulter (Cottet et al., 2000). Exemple : la figure 1.3 montre l'exemple d'un ordonnancement *Rate Monotonic* pour 3 tâches périodiques à échéance sur requête.
   $T_1(r_0 = 0, C = 3, P = 20)$, $T_2(r_0 = 0, C = 2, P = 5)$, $T_3(r_0 = 0, C = 2, P = 10)$.
   La tâche de plus haute priorité est T_2 et la tâche de plus basse priorité est T_1.
   La séquence d'étude est l'intervalle [0, 20][6].

   Les 3 tâches respectent leur contraintes temporelles car la condition suffisante est vérifiée : on a bien $\frac{3}{20} + \frac{2}{5} + \frac{2}{10} \leq 0.77$.

   Remarque : l'utilisation de la période comme critère d'ordonnancement limite l'application de *Rate Monotonic* aux seules tâches où l'échéance est égale à la période (échéances sur requête).

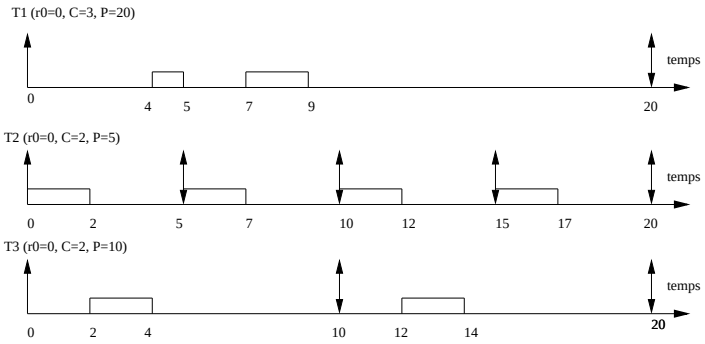

FIG. 1.3 – Ordonnancement *Rate Monotonic*

[6] Configuration `a d´epart simultan´e : p´eriode d'´etude [0, PPCM(P_i)]

2. Ordonnancement *Inverse Deadline* ou *Deadline Monotonic* (DM) (Audsley et al., 1991) :

   La priorité d'une tâche est fonction de son délai critique. La tâche la plus prioritaire est celle de plus petit délai critique. Cet algorithme fournit les mêmes performances que l'algorithme *Rate Monotonic* pour les tâches à échéances sur requête, et il est meilleur pour les autres types de configurations de tâches. Une **condition suffisante** d'acceptabilité est donnée par :

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{\frac{1}{n}} - 1)$$

– Algorithmes *en ligne* à priorité variable

1. *Algorithme Earliest Deadline*[7]*, (ED)* (Liu and Leyland, 1973) :

   La plus haute priorité à l'instant t est accordée à la tâche dont l'échéance est la plus proche. Pour les tâches à échéances sur requête, une **condition nécessaire et suffisante** d'ordonnançabilité est donnée par :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

   Pour des tâches périodiques quelconques, une **condition suffisante** d'ordonnançabilité est :

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$$

   et une **condition nécessaire** est :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

2. Ordonnancement *Least Laxity (LL) :*

   La tâche de plus haute priorité à l'instant t est celle qui a la plus petite laxité dynamique $L_i(t)$. Cet algorithme est optimal et les conditions d'ordonnançabilité des tâches sont les mêmes que pour l'algorithme *Earliest Deadline*. Les séquences produites par LL sont équivalentes à celles produites par ED lorsque les valeurs des laxités sont calculées aux dates de réveil. Par contre, si on calcule la laxité à chaque instant, alors la séquence produite par LL entraîne plus de changements de contextes qu'avec celle produite par ED.

   Rappel : $L(t) = D(t) - C(t)$ est la laxité nominale résiduelle et représente le retard maximum possible pour reprendre l'exécution d'une tâche si elle s'exécute seule ($L(t)$ est aussi égale à $D + r - t - C(t)$).

[7] ou EDF : Earliest Deadline First.

(ii) **Présence de tâches apériodiques**

Le but est que les contraintes temporelles (CT) des tâches périodiques soient respectées et, (i) soit que le temps de réponse des tâches apériodiques à contraintes relatives (*Best Effort*) soit minimisé, (ii) soit que le nombre de tâches apériodiques à contraintes strictes qui respectent leurs CT soit maximisé.

Nous allons voir dans la suite quelques algorithmes d'ordonnancement qui tiennent compte de l'arrivée de tâches apériodiques à contraintes relatives de différentes manières : en arrière-plan, par serveurs de tâches, etc.

1. Traitement en arrière-plan (*Background Processing*) :

   Les tâches apériodiques sont ordonnancées lorsque le processeur est oisif (il n'y a pas de tâche périodique prête). S'il existe plusieurs tâches apériodiques, alors le traitement s'effectue en les ordonnançant par dates de réveil ($FIFO$).

   La figure 1.4 montre un tel ordonnancement sur un intervalle égal à 2 fois la période d'étude d'une configuration de 2 tâches périodiques à échéances sur requête $T_1(r_0 = 0, C = 2, P = 5)$, $T_2(r_0 = 0, C = 2, P = 10)$.
   L'application de la politique d'ordonnancement *Rate Monotonic* à cette configuration laisse le processeur oisif sur les intervalles [4,5], [7,10], [14,15] et [17,20].
   La tâche apériodique $Ta_3(r = 4, C = 2)$ survenant à $t = 4$ peut immédiatement commencer son exécution, qu'elle finit sur le temps creux suivant, c-à-d entre les instants $t = 7$ et $t = 8$.
   La tâche apériodique Ta_4 (r=10, C=1) qui survient à t=10 doit attendre le temps creux [14,15] pour s'exécuter.
   La tâche apériodique Ta_5 (r=11, C=2) doit attendre le temps creux [17,20] pour s'exécuter.

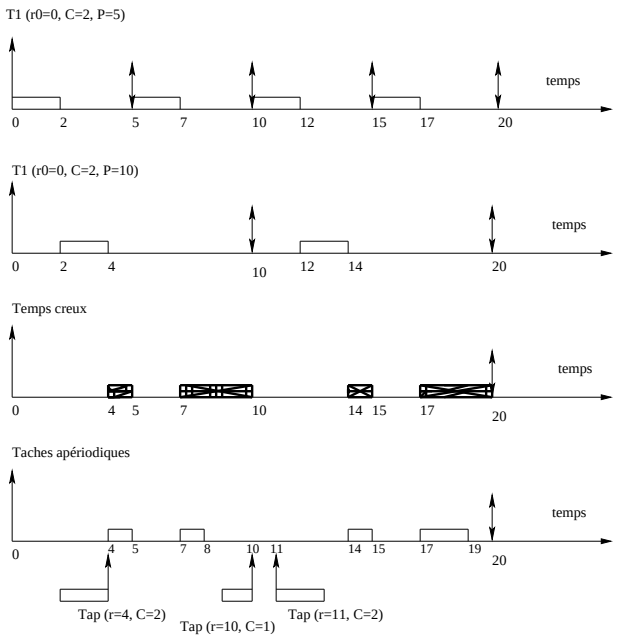

FIG. 1.4 – Ordonnancement "en arrière plan"

Remarque : La méthode d'ordonnancement en arrière-plan est simple et peu coûteuse et est applicable quelque soit l'algorithme d'ordonnancement des tâches périodiques. Mais les temps de réponse des tâches apériodiques peuvent être mauvais surtout si la charge des tâches périodiques est importante (Cottet et al., 2000).

2. Les serveurs de tâches :

Un serveur est une tâche périodique créée spécialement pour veiller à l'ordonnancement des tâches apériodiques. Elle est caractérisée par une période et un temps d'exécution (la capacité du serveur). Elle est souvent ordonnancée avec le même algorithme que les autres tâches périodiques.
Une fois active, la tâche serveur sert les tâches apériodiques dans la limite de sa capacité. L'ordre de service des tâches apériodiques ne dépend pas de l'algorithme d'ordonnancement des tâches périodiques (il peut être FIFO, fonction de l'échéances, fonction du temps d'exécution, ...).

Il existe plusieurs types de serveurs. Le plus simple est le serveur par scrutation. Les autres (serveur ajournable, à échange de priorité, sporadique) en sont des améliorations.

2.1. Traitement par 'scrutation' (*Polling*) :

À chaque activation, le serveur traite les tâches apériodiques en attente depuis son activation précédente, jusqu'à épuisement de sa capacité (ou il n'y a plus de tâches en attente). Si, lors d'une nouvelle activation, il n'y a aucune tâche apériodique en attente, le serveur se suspend jusqu'à la prochaine occurence : sa capacité (temps d'exécution) est récupérée par les tâches périodiques.

Remarques : La faiblesse de cette approche est que lorsque le serveur est prêt et qu'il n'y a pas de tâches à traiter, sa capacité est perdue. Si, ensuite, des tâches apériodiques surviennent, alors elles doivent attendre l'occurence suivante du serveur pour être traîtées.
Une amélioration a été apportée en utilisant un serveur *ajournable*. Dans cette approche, s'il n'y a pas de tâche apériodique à servir, alors le serveur conserve sa capacité. Il est donc prêt à servir d'éventuelles tâches apériodiques qui surviennent. Cependant, il viole le principe de l'algorithme *Rate Monotonic* qui stipule que la tâche prête de plus haute priorité est celle qui doit s'exécuter. Il diminue le taux d'utilisation du processeur par les tâches périodiques.

2.2. Serveur sporadique :

Il améliore les temps de réponse des tâches apériodiques sans diminuer le taux d'utilisation du processeur des tâches périodiques. Ce serveur est également une tâche périodique de plus haute priorité, qui conserve sa capacité de traitement si, à son réveil, il n'y a pas de tâche apériodique à servir. Mais ici, le serveur sporadique ne retrouve pas sa capacité initiale à chaque occurence.

1.2.4 STR et gestion des données

Les algorithmes d'ordonnancement que l'on a vu précédemment prennent en compte les tâches périodiques avec différentes configurations, d'autres algorithmes permettent d'ordonnancer les tâches

périodiques, en présence de tâches apériodiques qui, par définition, peuvent survenir à tout moment, perturbant ainsi la régularité d'exécution des tâches périodiques seules. Il existe d'autres algorithmes qui s'appliquent aux systèmes distribués temps réel où il faut non seulement s'occuper des la gestion des tâches (périodiques ou non), mais également tenir compte de la gestion des messages échangés entre les différents sites (Cottet et al., 2000).

Dans ces algorithmes, la seule ressource partagée est le processeur. Il existe d'autres algorithmes d'ordonnancement de tâches périodiques et apériodiques où il y a d'autres ressources que le processeur : les données. Cependant, l'accès à une donnée par une tâche temps réel se fait sous forme de section critique, et les données, qui ne sont pas très nombreuses, ne représentent pas une structure telle que l'on doive s'occuper de la cohérence de l'ensemble. La cohérence concerne uniquement la valeur de la donnée.

La gestion d'une quantité importante de données nécessite des mécanismes qui ne sont fournis que par les $SGBD$: structuration des données, maintien de la cohérence des données, langage de requête puissant sur la base, etc. (Ramamritham, 1993).

Dans le chapitre suivant, nous aborderons la problématique des $SGBD$ temps réel, en indiquant dans un premier temps pourquoi les $SGBD$ traditionnels, même améliorés, ne sont pas capables de gérer les contraintes temporelles (Stankovic et al., 1999). Ensuite, nous aborderons de manière assez succincte quelques problèmes qui se posent dans les $SGBD$ temps réel, ainsi que quelques solutions proposées (Duvallet et al., 1999b). Puis nous décrirons nos contributions relatives à la gestion des transactions dans les $SGBD$ temps réel.

2 Problématique des SGBD temps réel

Ces dernières années, l'intérêt croissant pour les systèmes temps réel a conduit à de nombreux travaux de recherche sur le calcul temps réel dans les domaines de l'informatique et de l'ingénierie. Mais, plus les systèmes temps réel évoluent, plus leurs applications deviennent complexes, et requièrent souvent des accès et l'exécution *dans les temps* de grandes masses de données. D'où un besoin de fonctionnalités de gestion de données avancées dans les systèmes temps réel, qui constitue un formidable défi intellectuel et d'ingénierie. Pour relever ce défi, des solutions doivent être trouvées dans la conception et le développement de *systèmes de gestion de bases de données temps réel (SGBDTR)*.

Il était communément admis que pour respecter les contraintes temporelles, il suffit d'augmenter suffisamment le débit du système. Mais les recherches sur les systèmes temps réel ont remis en cause cette idée. Ainsi, les défis et problèmes rencontrés par les concepteurs de SGBDTR sont très différents de ceux rencontrés par les concepteurs de SGBD généraux. Pour respecter les besoins fondamentaux des contraintes temporelles et de prévisibilité, il est nécessaire non seulement de revoir les méthodes conventionnelles d'ordonnancement et de gestion des transactions, mais également de considérer de nouveaux concepts à ajouter dans les SGBD et dans les STR conventionnels.

La genèse des recherches sur les SGBDTR a eu lieu en 1988 avec l'article de Abbot et Garcia-Molina (Abbot and Garcia-Molina, 1988). La confirmation que les recherches sur les SGBDTR ont atteint un point important est l'organisation du premier *Workshop* sur les SGBDTR (Bestavros et al., 1996b), puis la publication d'une session spéciale de ACM SIGMOD Record (Bestavros et al., 1996a), où les chercheurs des deux communautés SGBD et STR y ont contribué. Depuis, les recherches sur les SGBDTR ont pris de l'importance, témoin le nombre de publications de haut niveau sur le domaine, dont voici quelques références : (Abbot and Garcia-Molina, 1992; Ramamritham, 1993; Graham, 1993; Gupta et al., 1996; Andler et al., 1996; Bestavros and Braoudakis, 1996; Andler and Hansson, 1997; Lam and Yan, 1998; Duvallet et al., 1999a; Haritsa et al., 2000; Haritsa and Ramamritham, 2000b; Datta and Mukherjee, 2001; Hansson and Son, 2001; Kuo et al., 2001; Kang et al., 2002; Liu et al., 2002; Amirijoo et al., 2003b; Lindström, 2003; Shu, 2003; Qin and Liu, 2003).

Dans la suite de ce chapitre, nous commençons par décrire quelques caractéristques des SGBDTR. Nous décrivons ensuite quelques applications où l'utilisation des SGBDTR est fortement recommandée. Puis nous passons en revue quelques problèmes auxquels les chercheurs dans le domaine des SGBDTR se sont intéressés et nous en présenterons brièvement quelques solutions apportées. Ensuite, suivra un paragraphe qui présentera quelques prototypes de $SGBDTR$ implantés, principalement dans des laboratoires de recherche.

2.1 Idées fausses sur les bases de données temps réel

En informatique, une certaine méconnaissance existe lorsque l'on parle de calcul en général et de calcul temps réel. Stankovic avait donné dans un article (Stankovic, 1988) les arguments réfutant le fait que l'on peut effectuer des calculs temps réel moyennant des machines rapides. Dans le même ordre

d'idée, Stankovic et al. (Stankovic et al., 1999) ont réfuté les arguments qui soutiennent l'idée que les bases de données temps réel puissent être gérées avec les SGBD traditionnels moyennant quelques améliorations (rapidité, mise en mémoire centrale, ...). Parmi les idées fausses sur les bases de données temps réel, on peut citer les suivantes :

1. Les avancées dans la technologie matérielle des ordinateurs vont satisfaire les besoins des bases de données temps réel.
2. Les avancées dans les méthodes de conception et développement des bases de données vont satisfaire les besoins des bases de données temps réel.
3. Le calcul temps réel est équivalent à du calcul rapide.
4. Une base de données temps réel doit résider entièrement en mémoire centrale.
5. Une base de données temporelle est une base de données temps réel.
6. Il est impossible de garantir en même temps le respect des contraintes logiques et temporelles dans un SGBD temps réel.
7. Une base de données temps réel est une base de données spécialisée.

Le développement des arguments sur les points précédents peuvent être trouvés dans Stankovic et al. (Stankovic et al., 1999).

Les arguments ainsi présentés défendent l'idée qu'il est nécessaire de poursuivre les recherches sur des systèmes spécifiques qui puissent gérer les bases de données sujettes aux contraintes temporelles.

2.2 Transactions temps réel

L'un des principaux problèmes rencontrés lors de la conception de SGBDTR est relatif à la gestion des transactions : contrôle de concurrence et ordonnancement, résolution de conflits, validation, gestion de la surcharge, etc. Il s'agit également du problème le plus largement étudié. Il sera développé dans le chapitre 3, dans lequel nous présenterons également notre contribution relativement à cet axe de recherche.

Dans un SGBDTR, les transactions doivent non seulement respecter la cohérence logique de la base (respect des contraintes d'intégrité), mais doivent également respecter leurs contraintes temporelles individuelles, données souvent sous forme d'échéances. Certaines transactions dans les SGBDTR sont sujettes à une contrainte supplémentaire : elles doivent accéder à des données temporelles, i.e. qui possèdent des durées de validité au delà desquelles elles deviennent obsolètes.

Selon les conséquences du manquement des échéances des transactions sur l'application et sur son environnement, les transactions dans un SGBD temps réel sont classées en trois catégories (Ramamritham, 1993; Haritsa et al., 1992; Duvallet et al., 1999b) :

- Transactions à échéances strictes et critiques (*hard deadline transactions*) : une transaction qui manque son échéance peut avoir des conséquences graves sur le système ou sur l'environnement contrôlé. Par exemple (Mammeri, 1998), dans un système d'interception de missiles, les transactions qui permettent de contrer des missiles adverses sont à échéances strictes et critiques.

- Transactions à échéances strictes et non critiques (*firm deadline transactions*) : si une transaction manque son échéance, elle devient inutile pour le système. Ses effets sont nuls. Elle est donc abandonnée dès qu'elle manque son échéance. Prenons l'exemple d'une usine automatisée où un robot capte des informations sur des objets défilant sur un convoyeur. Si la transaction qui s'occupe de l'acquisition des caractéristiques d'un objet n'est pas terminée avant la disparition de l'objet du champ de vision du robot, elle est abandonnée et redémarrée pour prendre

en compte le prochain objet. On suppose que les objets dont l'acquisition d'informations a été abandonnée sont recyclés plus tard.

– Transactions à échéances non strictes (*soft deadline transactions*) : si une transaction manque son échéance, le système ne l'abandonne pas immédiatement. En effet, elle peut avoir une certaine utilité pendant un certain temps encore après l'expiration de son échéance, mais la qualité de service qu'elle offre est moindre. Par exemple, dans un système multimédia, on fixe des échéances pour la réception du son et de l'image, afin de garantir une bonne synchronisation de ces deux paramètres au moment de leur présentation à l'utilisateur final. Si le son et l'image arrivent sur deux canaux différents, il peut y avoir des décalages entre la réception des données *son* et des données *image*. Quand une image arrive légèrement en retard, elle peut toujours être présentée à l'utilisateur si le décalage par rapport au son n'est pas perceptible. Ainsi, une réponse à une requête (pour obtenir une image) peut être exploitée au-delà de l'échéance fixée. Il est évident que si le décalage est trop important, le système ne peut plus exploiter l'image reçue, sinon la scène devient incompréhensible.

Parfois une fonction *valeur* est utilisée pour modéliser les types de transactions (voir Figure 2.1). Quand elles manquent leurs échéances, les transactions de type *Hard* apportent une valeur négative pour le système, les transactions de type *Firm* apportent une valeur nulle et les transactions de type *Soft* apportent une valeur qui décroît avec le passage du temps jusqu'à devenir nulle.

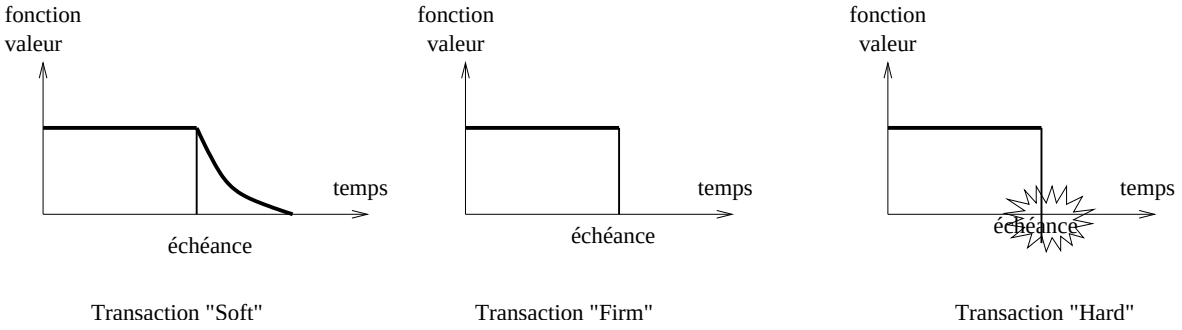

FIG. 2.1 – Une fonction *valeur* pour modéliser les types de transactions

La majorité des travaux sur la gestion des transactions porte sur le contrôle de concurrence et l'ordonnancement, et concernent surtout les transactions de type strict non critique (*firm*).

2.3 Propriétés à prendre en compte

Divers besoins temps réel sont à prendre en compte lors de la conception d'un SGBDTR. Parmi ces besoins, on peut citer :

– **Le contrôle d'admission**. Il s'agit de l'admission dans le système de nouvelles transactions. Le composant d'un SGBDTR dédié à ce rôle, le contrôleur d'admission, est en effet utile pour contrôler l'arrivée des transactions. L'objectif est de veiller à ce que le système ne soit pas sur-utilisé, donc surchargé. Ce qui provoquerait des manquements d'échéances d'un nombre important de transactions.

– **Le contrôle de concurrence et l'ordonnancement**. Il s'agit de faire collaborer le plus étroitement possible ces deux mécanismes. En effet, en cas de conflit d'accès à une donnée par plusieurs transactions, le contrôleur de concurrence résout le conflit en étroite collaboration avec l'ordonnanceur, en utilisant des techniques qui se basent sur les priorités et l'importance des transac-

tions.

– **Le mécanisme de recouvrement** (reprise), dont le fonctionnement doit être revu car en cas de reprise du système, certaines données temporelles deviennent obsolètes et il est inutile de les récupérer. Dans ce cas, des actions alternatives pourraient être prises.

2.4 Quelques applications des SGBDTR

Dans cette section, nous décrivons quelques applications pour lesquelles les SGBDTR sont appropriés (Locke, 2001). Nous prendrons un exemple dans quelques domaines : l'avionique et l'espace, le contrôle du trafic aérien, le contrôle industriel, le contrôle/commande, le multimédia et la bourse. Il existe de traditionnels et grands travaux sur la conception des applications dans ces domaines. Cependant, les solutions sont souvent basées sur des techniques 'ad hoc' au lieu de se servir des technologies de bases de données. Dans ce qui suit, nous considérons les besoins fondamentaux de chaque application et les effets que ces besoins auraient eu si ces applications ont été implémentées en utilisant une base de données temps réel.

Chaque application s'exécute dans un environnement qu'elle contrôle ou modifie selon des contraintes temporelles bornées. Cela implique que l'application utilise des capteurs et autres organes d'entrée pour accéder à l'information décrivant cet environnement. Chaque élément d'information (donnée) doit donc être stocké et retrouvé *dans les temps* avec un niveau approprié de cohérence des données relativement aux autres informations sur l'environnement.

2.4.1 Avionique

En général, une application d'avionique et espace peut être caractérisée par un ensemble de capteurs et d'actionneurs qui gèrent l'information décrivant et contrôlant un environnement externe. Par exemple, un système de contrôle de vol d'un avion possède des capteurs qui décrivent les actions du pilote comme le contrôle du gouvernail d'un avion, des interrupteurs, de la position, etc. Si, de plus, l'application consiste en un processeur réalisant une certaine mission, d'autres capteurs sont nécessaires pour la navigation et d'autres spécifiques à la mission considérée.

Les données gérées par ce type de systèmes comme pour la plupart des données temps réel peuvent être classifiées en deux catégories : (1) des données relatives à la mesure des paramètres de l'environnement comme la vitesse et l'altitude. La particularité de ces données est que leurs valeurs peuvent être estimées à partir de valeurs acquises précédemment, donnant ainsi un niveau de redondance qui permet de tolérer quelques incohérences temporelles, (2) les données de l'autre catégorie n'exhibent pas de propriétés dynamiques. On peut citer l'état des équipements qui peut prendre différentes valeurs, l'état d'un périphérique (ON/OFF), etc.

Taille de la BD	3 000 entités
Durée d'une lecture	0.05 ms. minimum
Durée d'une écriture	1 ms. maximum
Cohérence externe	0.05 sec.
Cohérence temporelle	0.20 sec.
Durabilité	4 heures

TAB. 2.1 – Caractéristiques des données dans un système d'un avion en mission (Locke, 2001)

La dynamicité des données temps réel fait qu'elles ne requièrent pas la propriété d'atomicité pour obtenir un niveau acceptable de cohérence, car l'erreur commise durant les différentes mesures

périodiques peut être souvent ignorée. Considérons, par exemple, un capteur qui reporte les valeurs de la latitude et la longitude d'un avion. Si l'atomicité des transactions est relaxée, alors la latitude lue durant une période et la longitude lue durant la période précédente pourraient toujours servir à calculer la position actuelle de l'avion ; l'erreur résultant de ces valeurs se trouvera bornée par la distance parcourue entre les deux périodes. Cela signifie que, pour les transactions qui manipulent cette catégorie de données et dans certaines circonstances, il n'est pas toujours nécessaire d'assurer les propriétés ACID. Cependant, pour les données du système de contrôle, il est grandement souhaitable que les transactions respectent la propriété d'atomicité pour atteindre un niveau acceptable de cohérence.

La table 2.1 illustre un exemple de caractéristiques des données temps réel gérées dans un système d'avion en mission. La taille de la base reflète le besoin de gérer une quantité importante d'informations sur les objets à surveiller sur terre et dans les airs, sur les facilités de navigation, sur les informations pour l'opérateur, etc. Les durées de lecture/écriture répondent au besoin de stocker et retrouver ces données avec suffisamment de latence telles que combinées avec les temps d'exécution et d'entrée/sortie, les contraintes temporelles *de bout en bout* soient respectées. La cohérence externe décrit un niveau acceptable d'erreur sur l'information concernant la latitude et la longitude (basée sur les valeurs d'autres paramètres comme la vitesse). La cohérence temporelle reflète un niveau acceptable d'erreur pour la plupart des données sensorielles (acquises par des capteurs). La durabilité est déduite de la durée de la mission.

2.4.2 Surveillance du trafic aérien

Le principe d'un système de surveillance de trafic aérien est de repérer les positions de tous les avions dans une portion de l'espace correspondant à une zone géographique donnée. Par exemple, aux États Unis, il existe une vingtaine de tels systèmes, appelés des ARTCC (Air Route Traffic Control Center), qui contrôlent tout l'espace aérien du pays, en plus de plusieurs terminaux de contrôle qui gèrent les avions survolant les alentours des villes ayant un ou plusieurs aéroports. Un ARTCC peut contenir des centaines de contrôleurs de trafic aérien, chacun d'eux s'occupant d'un secteur spécifique de l'espace aérien et disposant d'ordinateurs qui enregistrent et renseignent sur l'identification des avions, leur position, les plans de vol, les lignes attribuées, la météo, etc.

Quand un avion quitte un aéroport, son équipage communique avec les contrôleurs du trafic dans l'aéroport jusqu'à ce que l'avion décolle et quitte la zone de l'aéroport où il rentre sous le contrôle d'un ARTCC. Ensuite, en fonction de la distance à parcourir, l'avion est pris en charge par un secteur (ARTCC) et le secteur suivant, etc. puisqu'il traverse plusieurs secteurs, à différentes altitudes.

Avant son départ de l'aéroport, si l'avion opère sous le contrôle d'instruments de vol (IFR[1]), alors il aura renseigné un plan de vol décrivant le type d'avion, l'identification, la destination, la route durant le vol, et autres informations relatives au vol. Durant son vol, l'avion acquiert des renseignement sur sa latitude et sa position utilisant des informations *radar* et reste en contact permanent avec l'ARTCC de la zone survolée.

De plus, chaque contrôleur dispose d'un unique identificateur associé à certaines préférences : modes d'affichage, configurations, types de contrôles à utiliser et autres informations. Durant un changement de contrôleur, en cas de panne d'affichage ou d'ordinateur par exemple, le contrôleur doit aiguiller rapidement ses affichages sur un écran disponible, qui doit alors être immédiatement fonctionnel (correctement configuré et ajusté au secteur selon les préférences).

La table 2.2 illustre un exemple de données manipulées dans la surveillance de trafic aérien et leurs caractéristiques, notamment temporelles.

[1] Instrument Flights Rules

Taille de la BD	20 000 entités
Durée d'une lecture	0.05 ms. minimum
Durée d'une écriture	5 ms. maximum
Cohérence externe	1.5 sec.
Cohérence temporelle	3 sec.
Durabilité	12 heures

TAB. 2.2 – Caractéristiques des données dans un système de surveillance du trafic aérien (Locke, 2001)

2.4.3 Contrôle industriel

Un système de contrôle industriel implique généralement la gestion d'un ensemble de capteurs et d'actionneurs relatifs à des processus de fabrication de produits. Les informations manipulées sont par exemple la localisation de divers matériels, les moyennes des flots d'arrivée, les positions et vitesses des convoyeurs, les conditions d'activation de réactions chimiques, les informations de contrôle de robots, etc. La plupart de ces informations sont assujetties à des contraintes temporelles qui doivent être maintenues continuellement. De plus, les systèmes de contrôle industriels impliquent souvent la gestion d'autres informations telles que les dessins industriels, des spécifications et modélisations de processus, des indicateurs de statuts, des mesures de productivité des employés et beaucoup d'autres informations.

Les données utilisées dans ces contrôles sont considérées comme critiques pour toute la gestion du système pour plusieurs raisons. Plusieurs processus de fabrication dans des usines ont des implications sécuritaires pour les employés et pour l'environnement. Les usines sont soumises à des surveillances de plus en plus accrues si bien que que le processus de fabrication devient dépendant de la manière dont le système est capable de minimiser les risques.

Ceci implique par exemple que l'information critique telle que la localisation de parties dangereuses, les maxima de sûreté des températures et pression, la localisation des substances dangereuses, soient maintenues dans un état stable et cohérent. Au niveau du système, ceci implique également que les propriétés d'atomicité et de cohérence des transactions soient assurées. Par contre, le critère de durabilité de l'information mesurée n'est généralement pas critique sur le long terme, à l'exception des informations périodiques.

Taille de la BD	100 000 entités
Durée d'une lecture	1 ms. minimum
Durée d'une écriture	10 ms. maximum
Cohérence externe	0.20 sec.
Cohérence temporelle	0.50 sec.
Durabilité	5 jours

TAB. 2.3 – Caractéristiques des données dans un système de contrôle industriel (Locke, 2001)

La table 2.3 montre un exemple de valeurs typiques d'une base de donnée de contrôle industriel. La taille de la base reflète le nombre d'assemblage/parties qui pourraient être présents dans une grande usine. Les temps de lecture/écriture doivent refléter la vitesse qui pourrait être exigée pour que des valeurs telles que la pression ou la température soient maintenues à jour. Les valeurs des cohérences externe et temporelle pourraient être celles résultant des mesures de la position et de la vitesse d'un convoyeur, alors que la valeur de la durabilité pourrait être dérivée du temps total mis par un convoyeur pour se déplacer d'une zone à une autre de l'usine.

2.4.4 Contrôle/commande

Les systèmes de contrôle/commande se distinguent généralement par la manipulation de quantités importantes d'informations de différentes natures, provenant d'un environnement externe. Ces informations doivent être gérées par le personnel de commande de telle manière que les valeurs des données de l'environnement externe représentant une situation donnée amènent à une conclusion satisfaisante (succès).

Taille de la BD	50 000 entités
Durée d'une lecture	0.50 ms. minimum
Durée d'une écriture	5 ms. maximum
Cohérence externe	0.05 sec.
Cohérence temporelle	0.10 sec.
Durabilité	1 heure

TAB. 2.4 – Caractéristiques des données dans un système de contrôle/commande (Locke, 2001)

Dans ces systèmes, les informations proviennent en général de systèmes de messageries, de sources *intelligentes*, d'autres sources comme les chars ou les avions. Ces informations transitent par des lignes de communications qui peuvent être numériques ou non, avec ou sans fils, ou de type radio. Sans tenir compte de sa source, toute cette information doit être regroupée et intégrée dans un seul système qui doit pouvoir la transmettre aux décideurs. Le système de contrôle/commande doit donc gérer ces données de telle manière qu'elles restent *fraîches* et disponibles pour les décideurs durant un intervalle de temps donné. Ce type de systèmes est souvent appelé "système de systèmes" car il fournit un contrôle centralisé d'un grand nombre d'autres systèmes comme le système de messagerie, le système de contrôle de lutte contre le feu.

Un exemple de caractéristiques des données manipulées dans un système de contrôle/commande est donné dans la table 2.4.

2.4.5 Gestion des données multimédia

Des applications de SGBD temps réel se trouvent également dans le domaine du multimédia. Il s'agit, par exemple, d'un système de distribution de vidéo mobile où des clients sont connectés à un serveur vidéo par des liaisons sans fil. Les requêtes auprès du serveur s'effectuent pendant que les clients sont mobiles. Ces requêtes sont plus ou moins bien servies en fonction de certains paramètres : volumes de données, largeur de la bande passante, mobilité des clients, charge du réseau, etc.

Pour assurer une meilleure qualité de service aux clients, il est indispensable que le système puisse s'adapter aux changements. Il doit par exemple disposer d'un contrôleur d'admission qui filtre les transactions à servir. Il doit également disposer d'un ordonnanceur basé sur les priorités pour pouvoir satisfaire en priorité les transactions les plus urgentes. Il faut en effet tenir compte de l'urgence des paquets à transmettre pour que des coupures ne se produisent pas à la reception. Des algorithmes tels que EDF ou RM sont étendus à la gestion des flux multimédia. L'algorithme EDF ainsi que d'autres algorithmes sont également étendus pour une utilisation dans un environnement mobile (Shakkottai and Srikant, 1999; Rammanathan and Agrawal, 1998).

Dans la majorité de ces applications, la proportion des transactions *à lecture seule*, i.e. interrogation, est de loin supérieure à celle des transactions de mise à jour. L'objectif est donc d'améliorer la qualité de service offerte aux clients qui effectuent des requêtes auprès d'une base de données globale située sur le serveur.

2.4.6 Application boursière

Dans (Adelberg and Kao, 2001), Adelberg et al. ont travaillé sur la conception d'un système de bases de données spécialement dédié à des environnements hétérogènes. L'application sur laquelle ils ont testé leur système consiste en un programme d'application boursière. Ce système permet de surveiller les prix des actions de produits ou d'autres éléments financiers, et de déterminer les opportunités du marché en fonction des caractéristiques ou des modèles à court terme du marché.

En général, les programmes de gestion de la bourse sont développés en direction des clients et restent des secrets de l'entreprise. L'exemple présenté est donc simplifié d'une part par nécessité, car très peu d'informations sont disponibles, et d'autre part pour focaliser l'attention sur les caractéristiques principales de gestion des données sans se soucier des détails relatifs à la modélisation du marché.

Un programme de gestion de marché doit maintenir trois types de prix : les cours des actions, les prix indexés et les prix théoriques des options. Les cours des actions constituent les données de base du système et sont mis à jour dans la base de données selon l'état du marché. Les prix indexés sont des données dérivées qui doivent être calculés à partir des cours des actions. En réalité, la tendance actuelle des fournisseurs de produits est d'envoyer d'autres éléments que les cours des actions, comme les prix composés populaires (données populaires), i.e. le DJIA[2] aux États Unis, et d'autres valeurs.

La base de données utilisée est composée des tables suivantes (Adelberg and Kao, 2001) :
- *Action (symbol, prix)* : table de base contenant les actions et leurs prix.
- *Action-Ecart-Type (symbol, ecart-t)* : table de base contenant les écarts-type des prix des actions sur une année.
- *Composite-Prix (comp, prix)* : table qui représente une vue matérialisée contenant les moyennes des prix de composés calculés, e.g. DJIA, qui sont des données dérivées.
- *Composite-liste (comp, symbol, poids)* : table de base représentant les occurrences d'association *plusieurs-à-plusieurs* entre les actions et les prix des composés indexés sur ces actions. Les valeurs de cette table changent très peu souvent.
- *Option-Prix (opt-symbol, prix)* : table qui représente une vue matérialisée contenant les prix calculés pour chaque option listée. Ce sont des données dérivées.
- *Option-Liste (opt-symbol, action-symbol, prix-v, expir)* : table de base représentant les occurrences d'association *un-à-plusieurs* entre les actions et les options. Les valeurs de cette table doivent être mises à jour tous les trimestres quand une option de change crée une nouvelle option et efface les options ayant expiré. L'attribut *prix-v* est le prix auquel le détenteur d'actions peut vendre l'action *action-symbol* avant son expiration.

Les tables *Composite-Prix* et *Option-Prix* sont des vues matérialisées, dont les définitions sont les suivantes :

create view *Composite-Prix* **as**
select *comp,* **sum***(prix*poids)* **as** *prix*
from *Action, Composite-liste*
where *Action.symbol = Composite-liste.symbol*
group by *comp*

create view *Option-Prix* **as**
select *opt-symbol,* f_{BS}*(prix, prix-v, expir, ecart-t)* **as** *prix*
from *Action, Action-Ecart-Type, Option-Liste*

[2] Dow Jones Industrial Average

where *Action.symbol = Option-liste.action-symbol*
and *Action.symbol = Action-Ecart-Type.symbol*

La fonction f_{BS} calcule le prix théorique d'une option en utilisant un modèle très commun en marché boursier (Black and Scholes, 1973).

Parmi les problèmes intéressants que les auteurs ont rencontré lors de l'implémentation de cette application, nous pouvons citer les suivants :

- Le maintien de la cohérence temporelle : dans une application boursière, l'environnement à modéliser est complètement séparé de la base de données. Les cours des actions changent sans se référer aux valeurs de la base, forçant l'application à scruter l'environnement et à notifier les changements à la base de données. Par conséquent, les valeurs stockées dans la base sont toujours en décalage avec les valeurs réelles (à cause des délais dus aux échantillonnages, aux transmissions et aux mises à jour).

- L'ordonnancement des mises à jour : dans un SGBD autonome, les seules transactions qui s'y exécutent proviennent des utilisateurs et sont typiquement ordonnancées selon leurs échéances ou selon des valeurs de fonctions. Dans un système qui importe et exporte des données, il existe d'autres sources de transactions. Si le volume des données échangées n'est pas trop important, alors des priorités élevées peuvent être affectées à ces transactions sans grave conséquence sur les transactions *utilisateur*. Par contre, dans les applications où les transactions d'import/export sont nombreuses comme la bourse, de nouveaux algorithmes sont à concevoir qui permettront d'équilibrer les deux types de travaux.

  Le principal compromis est à trouver entre les temps de réponse des transactions *utilisateur* et la fraîcheur des données. La définition de la fraîcheur varie selon l'application. Par exemple, si dans la base sont stockées les valeurs décrivant un changement continu de l'environnement, i.e. *la température* dans une centrale, alors la donnée pourrait devenir obsolète presque immédiatement. Dans le cas où les changements de l'environnement surviennent à des instants discrets, comme les prix des actions, la valeur d'une donnée reste stable jusqu'à l'occurrence d'un autre prix pour l'action. Le comportement des applications est également différent vis à vis des données obsolètes : certaines applications peuvent choisir d'utiliser la valeur disponible la plus récente, d'autres choisissent d'abandonner la transaction si la donnée est non fraîche.

- La gestion des données dérivées : lorsque les valeurs des données de base changent fréquemment, la maintenance des données dérivées peut nécessiter une part non négligeable du temps *CPU* de la machine. Cependant, la propriété de *localité* des données est utilisée pour alléger la charge du système. La localité des mises à jour signifie que quand une donnée est mise à jour, alors il est très probable que la même donnée ou une donnée qui lui est associée soit mise à jour juste après. Par exemple, une mise à jour du cours d'une action montre l'intérêt porté à ce commerce. La même action fera alors probablement l'objet d'autres activités commerciales et d'autres mises à jour. La localité des mises à jour implique une explosion des calculs de données dérivées. La méthode utilisée par les auteurs pour éviter cette explosion est de différer les re-calculs de données dérivées. Ils utilisent pour cela une stratégie, appelée *retard forcé*, qui consiste à ne pas re-calculer immédiatement après la mise à jour d'une donnée de base, les valeurs dérivées. Ils utilisent une règle active, qui est une sorte d'extension de *trigger* de SQL-3. Pour la gestion des applications boursières, les auteurs ont intégré dans leur SGBD à base de règles actives le concept de *transaction unique*. Ce concept permet de (i) regrouper des mises à jour au lieu d'exécuter une transaction à la fois, et (ii) de partitionner cet ensemble de transactions de n'importe quelle manière pour réduire le coût de re-calcul des données dérivées.

– Gestion de l'application boursière : pour tenir compte des changements fréquents des prix des actions, la meilleure solution est l'utilisation de règles actives. Les auteurs ont créé des règles qui mettent à jour les prix des options dès que le cours d'une action change. Pour éviter une explosion des mises à jour des tables dérivées, une fenêtre de temps a été définie de manière à regrouper toutes les modifications à effectuer pendant cette fenêtre de temps et les accomplir en une seule fois.

Les détails sur le fonctionnement de l'application boursière et la manière dont sont implémentés les règles, les processus et les ordonnanceurs peuvent être trouvés dans (Adelberg and Kao, 2001).

Les auteurs ont testé une application boursière particulière et ont conclu que les caractéristiques relatives à la gestion des données temps réel (aspect actif, maintien de la cohérence temporelle, algorithmes d'ordonnancement adéquats) ont permis d'améliorer ses performances. Par exemple, le concept de *transaction unique* a réduit la charge du système de 50% si la fenêtre de temps utilisée pour les mises à jour de données dérivées de la table *Composite-Prix* est de 0.5 seconde, et de 67% pour une fenêtre de temps de 3 secondes. Pour la table *Option-Prix*, la réduction de la charge a été de 20% pour une fenêtre de temps de 3 secondes.

2.5 Quelques axes d'étude sur les SGBDTR

Depuis le début des recherches sur les SGBDTR, beaucoup de travaux sur différents aspects de ces systèmes ont été effectués. Dans la suite, nous allons décrire les grandes lignes de quelques uns de ces travaux comme la gestion de la surcharge, les mécanismes de réaction, la gestion des tampons en mémoire, et le problème de la sauvegarde restauration. L'axe spécifique concernant la gestion des transactions temps réel sera traité dans le chapitre 3 car il est en relation directe avec nos principaux travaux.

2.5.1 Gestion de surcharge

2.5.1.1 Introduction

Des états de surcharges transitoires se produisent souvent dans les SGBD temps réel. Ces états correspondent à des moments dans le système où la demande de ressources est supérieure aux ressources disponibles. La conséquence est que le système n'est plus capable de satisfaire entièrement les besoins de cohérence de la base (temporelle, logique et externe). Des travaux ont été effectués pour tenter de minimiser les effets des ces surcharges transitoires. Le gestionnaire de ressources doit assurer une dégradation graduée et contrôlée des performances qui passe, entre autres, par l'utilisation de politiques d'ordonnancement adéquates. Tandis que les politiques classiques d'ordonnancement tels que EDF ou RM produisent des résultats optimaux dans des conditions normales, elles fournissent généralement des performances médiocres dans des conditions de surcharge.

Les SGBDTR opèrant souvent dans des situations de surcharge, le système doit posséder des mécanismes non seulement pour les détecter, mais également pour les résoudre. La gestion des ressources dans les SGBDTR consiste en plusieurs activités : le contrôle d'admission qui détermine quelles transactions peuvent détenir des ressources, un ordonnanceur qui détermine l'ordre d'exécution des transactions admises, un gestionnaire de surcharge qui détermine comment résoudre les surcharges transitoires.

Parmi les politiques de résolution de surcharge, il y a celles qui rejettent ou abandonnent les transactions durant la surcharge, celles qui font migrer les transactions d'un processeur à un autre ou d'un

site à un autre (systèmes multiprocesseurs ou distribués), et celles qui remplacent des transactions par d'autres moins gourmandes en ressources. D'autres politiques facilitent l'ajout de ressources au système ou en modifient les critères de correction lors des phases de surcharge.

2.5.1.2 Approche étudiée

Dans cette section, nous décrivons une méthode proposée par Hansson et Son (Hansson and Son, 2001), car elle nous paraît innovante parmi les politiques existantes. En effet, l'idée de base derrière *le résolveur de surcharge* qu'ils ont proposé diffère des précédentes approches puisqu'elle est basée sur la génération de plans de résolution de surcharges (PRS). Ces PRS résolvent la surcharge transitoire en désallouant des ressources à des transactions préalablement admises dans le système. Deux actions peuvent être entreprises en cas de surcharge : ou bien substituer à des transactions leurs transactions *contingentes* moins gourmandes en ressources, mais fournissant un résultat incomplet et/ou imprécis, ou bien rejeter ces transactions. Par conséquent, les transactions admises n'ont pas la garantie absolue d'obtenir toutes les ressources dont elles ont besoin. Dans les deux cas précédents, il en résulte *une perte d'utilité* relativement au souhait initial. L'objectif est de minimiser cette perte d'utilité.

Chaque cas précédent constitue un PRS. Le résolveur de surcharge compare les PRS générés et en choisit le meilleur (celui qui minimise la perte d'utilité). Plus précisément, le résolveur de surcharge fonctionne de la manière suivante :

- il détermine l'*intervalle critique de surcharge* (ICS)[3] et calcule la durée qu'il est nécessaire de retirer de l'ICS pour résoudre la surcharge,
- il génère un ou plusieurs PRS, où un PRS consiste en un ensemble d'*actions de résolution de surcharge* (ARS) qui désallouent des ressources à des transactions déjà admises,
- il décide s'il est avantageux d'exécuter un des PRS en considérant la perte ou le gain d'utilité relative en exécutant le PRS et en acceptant la nouvelle transaction, ou bien s'il est préférable de rejeter simplement la transaction ou de lui substituer une autre moins gourmande en ressources.

La génération des plans de résolution de surcharges (PRS) suit les étapes suivantes :

- Génération de l'ensemble des ARS possibles dans l'intervalle ICS et calcul de leur perte d'utilité.
- Tri des ARS selon le critère *perte d'utilité*.
- Itération sur les ARS dans l'ordre décroissant de la perte d'utilité, et ajout de ces ARS au PRS jusqu'à ce que le temps total "gagné" par le nouveau PRS soit supérieur ou égal au temps dont a besoin la nouvelle transaction.

Pour mettre en oeuvre l'idée précédente, les auteurs ont proposé OR-ULD[4], un algorithme qui permet de générer des PRS, selon le schéma décrit précédemment. Ils ont conduit des simulations dans un système où s'exécutent des transactions sporadiques critiques, ayant des transactions contingentes, et des transactions apériodiques de type *firm*. Ils ont comparé la gestion du contrôle d'admission en utilisant OR-ULD avec un autre contrôleur basé sur EDF. Ils ont conclu que le protocole OR-ULD permet un meilleur fonctionnement dégradé des performances du système en cas de surcharge (Hansson et al., 1998).

Le problème de la surcharge du système est crucial dans les SGBDTR. Dans ces systèmes comme dans tout système temps réel, il est nécessaire de disposer d'un contrôleur d'admission qui filtre les accès des transactions dans le système en fonction d'un certain nombre de paramètres : le temps estimé d'exécution de la transaction qui arrive, son urgence, les ressources requises, la charge courante du

[3] qui détebute à l'instant où une surcharge est détectée et l'instant d'un manquement d'échéance

[4] Overload Resolver-Utility Loss Density

système, etc.

Des travaux ont apporté des améliorations en décomposant les transactions en parties obligatoires et optionnelles, pour ensuite garantir l'exécution avant échéance des parties obligatoires de toutes les transactions et maximiser le nombre de parties optionnelles qui s'exécutent avant leurs échéances. D'autres travaux exploitent des résultats sur la théorie du contrôle par rétroaction pour concevoir des algorithmes d'ordonnancement qui opèrent dans des environnements où la charge est imprévisible comme les SGBD (*feedback control scheduling*) (Amirijoo et al., 2003a; Lu et al., 2002; Kang et al., 2002).

2.5.2 Mécanismes de réaction

Des travaux ont déjà été effectués sur les mécanismes réactifs dans les SGBD traditionnels. L'implantation de ces mécanismes a donné lieu à des prototypes de SGBD Actifs comme NAOS (Collet and Coupaye, 1998) ou d'autres (Dayal, 1989). Ils ont été implantés sous forme de règles ECA (Événement-Conditions-Actions). Ces travaux sur les SGBD Actifs ont servi de base à l'utilisation de ces mécanismes dans les SGBD temps réel. On remarquera que l'aspect réactif est inhérent à beaucoup d'applications temps réel. Donc, le fait d'intégrer les mécanismes réactifs aux contraintes temporelles est une voie naturelle pour construire des systèmes qui gèrent beaucoup d'applications temps réel actuelles.

2.5.2.1 Introduction

Dans les SGBD actifs, sont introduits la gestion des événements et la composition d'événements. Les premiers travaux ont été proposés par exemple dans le cadre du projet HiPAC (Dayal et al., 1988), puis dans (Liu et al., 1998), Liu et al. y ont présenté une approche unifiant l'aspect actif et la gestion des contraintes temporelles dans les SGBD. Afin de satisfaire l'exécution des transactions avant échéance, les règles ECA sont étendues avec des *attributs temporels*. Les règles sont déclenchées quand l'événement spécifié dans ces règles survient. Quand le gestionnaire de règles reçoit la notification de l'événement, toutes les règles concernées doivent être recherchées depuis la base de règles et préparées pour l'exécution. Ce temps de recherche des règles doit être minimisé car il constitue une charge imprévisible qui risque de conduire les transactions à manquer leurs échéances. Une fois les règles retrouvées, leurs conditions sont évaluées afin de déterminer celles qui vont être exécutées, ou bien au sein de la même transaction, ou bien dans une transaction séparée.

2.5.2.2 Distribution des règles

Mellin et al. (Mellin et al., 2001) ont étudié ces mécanismes de réaction sur un SGBD où la base de données est distribuée sur plusieurs sites, et les mécanismes réactifs le sont également. Ils ont étudié comment un événement détecté sur un noeud (site) peut-il déclencher des règles sur un autre noeud, sachant que l'on considère qu'un événement composite consiste en des événements détectés sur différents noeuds. Au niveau d'un noeud, une règle devrait être déclenchée et peut engendrer l'exécution d'actions sur d'autres noeuds. Cette approche a été utilisée dans la réalisation du prototype de $SGBDTR$ actif DeeDS (Andler et al., 1996).

2.5.2.3 Spécification des événements et des règles

Ce paragraphe décrit la manière dont les événements et les règles devraient être spécifiés, comme cela a été réalisé dans le prototype DeeDS. On distingue trois parties dans la spécification des événements : i) l'expression des événements, ii) les souscripteurs, et iii) les contraintes temporelles. Au contraire des SGBD actifs non temps réel, la spécification des événements dans DeeDS doit permettre de tenir compte des contraintes temporelles.

Pour limiter les besoins en ressources dans la gestion des événements, il est nécessaire d'ajouter des contraintes temporelles qui placent le système dans des états intermédiaires, par exemple, sans ces contraintes, les files d'attente internes pourraient s'agrandir indéfiniment.

Les règles ECA des SGBD actifs ne tiennent pas compte des attributs temporels tels que l'échéance, le temps maximal d'exécution, la date de réveil "au plus tôt" ou la criticité. Quelques propositions ont été faites pour inclure quelques attributs temporels simples dans les règles (Dayal et al., 1988; Chakravarthy et al., 1994). Andler et al. ont dans le prototype DeeDS (Andler et al., 1996) tenu compte complètement des attributs temporels.

Considérons l'exemple suivant :

```
constant tank.period = 15s
    event read_tanklevel_period periodic tank.period
    rule read_tanklevel
    on read_tanklevel_period
    if true
    do deadline tocc + tank.period criticality firm execution_time 600ms
    begin
        dbstore("tanklevel", tank.read_level());
    end
```

La règle spécifiée lit périodiquement le niveau d'eau dans le char et stocke sa valeur dans la base de données. Les opérations de lecture et de sauvegarde sont évaluées de telle manière qu'elles ne prennent pas plus de 600ms. Comme la criticité est de type *firm*, ces opérations doivent être terminées avant le début de la nouvelle période, sinon elles seront abandonnées. L'échéance est spécifiée relativement à l'instant de l'occurrence de l'événement déclencheur (*tocc*). La règle est déclenchée par l'événement prédéfini *read_tanklevel_period* qui est un événement périodique qui survient toutes les 15s. La condition est fixée à *true*, car la règle doit toujours être exécutée. L'action a donc des attributs temporels tels que l'échéance, la criticité et le temps d'exécution.

2.5.2.4 Exécution

L'exécution des événements consiste en plusieurs catégories :

1. La souscription d'événements : quand un événement survient, il est envoyé au gestionnaire d'événements. Des instances d'événements sont alors générées. De plus, les priorités entre événements sont gérées. Une manière de réaliser cela est de "signaler" une instance d'événement une seule fois pour la priorité à laquelle il est utilisé. Il peut arriver que le système soit inondé d'événements, mais l'utilisation de pointeurs sains (Mellin, 2000) réduit la gêne ainsi occasionnée.
2. Des algorithmes de composition d'opérateurs : la plupart des algorithmes possèdent des opérateurs binaires comme la conjonction ou la disjonction. Il y a également des opérateurs d'intervalle (pour activer/désactiver un opérateur sur un intervalle de temps). La sémantique des opérateurs utilisés pour le temps réel est : chronique, récent, continu, car ils ne génèrent qu'une instance d'événement pour chaque occurrence signalée par un opérateur.
3. Des mécanismes utilisés pour la réalisation des algorithmes des opérateurs : ce sont notamment les graphes colorés (Chakravarthy and Mishra, 1994), les réseaux de Petri (Gatziu and Dittrich, 1993), les automates d'états finis (Gehani et al., 1993) ou la logique temps réel (Liu et al., 1998).
4. Évaluation des conditions des événements des opérateurs : les conditions servent à détecter si les événements ont expiré. Elles sont utilisées pour des raisons de performances, car elles évitent de

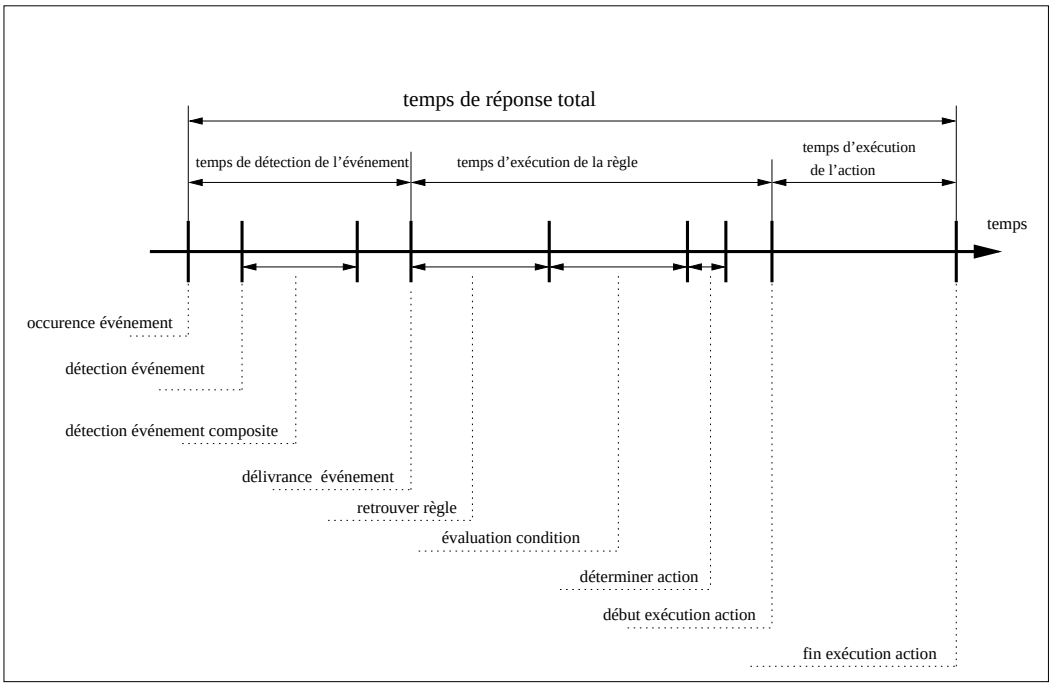

FIG. 2.2 – Calcul du temps de réponse en présence de règles ECA

signaler l'expiration des événements aux règles.

5. Gestion de la mémoire utilisée dans la composition d'événements : en combinant l'utilisation des pointeurs sains et la mémoire auto-référencée, cela permet de réduire d'environ 20% la contention par rapport à sa non-utilisation (Mellin, 2000).
6. Service de *timeout* : ce service permet notamment d'éviter de rallonger indéfiniment les files d'attente internes d'événements.

L'exécution des règles implique plusieurs étapes comme indiqué sur la figure 2.2. Pour garantir la prévisibilité dans l'exécution d'une règle, chaque étape doit être prédictible, i.e. minimiser les aléas de l'exécution. Le temps de réponse est divisé en trois parties : le temps de détection de l'événement, le temps d'exécution de la règle et le temps d'exécution des actions. En général, l'exécution des actions s'effectue au sein de nouvelles transactions pour éviter de rallonger le temps d'exécution de la règle, quelque soit le mode de couplage (immédiat ou différé).

2.5.3 Gestion des tampons

La gestion des tampons en mémoire centrale (MC) est un problème important dans les SGBD. L'utilisation de tampons est due au fait que les bases de données sont trop volumineuses pour tenir entièrement en MC. Le principe de la "mise en tampons" intelligente est de maintenir la partie stratégique d'une BD en MC de telle manière que les transactions n'ont pas à effectuer souvent des accès disque pour chercher leurs données en mémoire secondaire (MS), réduisant ainsi les temps de réponse.

Bien qu'il existe beaucoup de travaux sur la gestion des tampons dans les SGBD traditionnels, il y'en a très peu sur la gestion des mémoires-tampons (désignées simplement par le terme *tampons* dans la suite) dans les SGBDTR. On peut citer les travaux de Jauhari et al. ainsi que ceux de Datta et al. (Jauhary et al., 1990; Datta and Mukherjee, 2001). Dans ce paragraphe, nous donnons les grandes lignes de ce dernier travail.

Dans (Datta and Mukherjee, 2001), Datta et al. ont proposé une politique de gestion de tampons dans les SGBDTR actifs. Dans ces systèmes, le scénario le plus commun est que des actions de contrôle doivent être exécutées suite à une réaction incorrecte du système. Ce comportement réactif (voir paragraphe précédent) est souvent modélisé sous forme de règles ECA. Par exemple, *SUITE à un rapport d'événements, SI la température du réacteur est* $> 1000°$*, ALORS réduire la pression de 10 atm, DANS les 10 sec*. Cet exemple illustre l'aspect actif et l'aspect temps réel des systèmes qui gèrent ce type d'applications, qui sont des SGBD temps réel particuliers, puisqu'ils intègrent cet aspect réaction à des événements.

Dans ces systèmes, nous disposons en général de connaissances importantes sur les transactions : la périodicité et les ensembles de données lues et écrites. Cette connaissance est exploitée dans leurs travaux (Datta and Mukherjee, 2001) par Datta et al. pour proposer PAPER[5], un algorithme de gestion de tampons pour les transactions temps réel actives.

Le modèle de transactions utilisé est le suivant. Chaque transaction se voit assigner un espace local de travail. Tous ces espaces de travail sont vus comme des tampons privés affectés aux transactions actives, sous l'hypothèse qu'il y a suffisamment d'espace pour toutes les transactions concurrentes. Entre les tampons privés et le disque (MS) se trouve un *tampon global*, auquel chaque transaction peut accéder.

En quelques mots, la philosophie générale de l'algorithme PAPER est (1) que la politique de remplacement est basée sur les priorités (des priorités sont assignées aux tampons de l'espace global). La priorité d'une page donnée est modifiée en fonction de ses références *anticipées* aux données et des priorités des transactions qui vont accéder à cette page-mémoire, (2) sachant que les données accédées et les schémas de déclenchement sont connus (grâce à un graphe de déclenchement), il est possible d'anticiper avec quelque incertitude le schéma d'arrivée de nouvelles transactions, en fonction de l'état courant du système. L'algorithme PAPER exploite cette connaissance pour rechercher par anticipation des données dont les futures (prochaines) transactions auront besoin.

PAPER intègre donc deux composants : un composant de remplacement de pages basé sur les priorités et un composant de recherche de données par anticipation.

Les auteurs ont implémenté deux versions de l'algorithme : PAPER$^-$, qui intègre uniquement le premier composant (donc l'aspect temps réel seulement), et PAPER$^+$ qui intègre les deux aspects : temps réel et actif.

Les auteurs ont comparé par simulation les performances de ces algorithmes avec trois autres algorithmes de remplacement de pages : le plus connu LRU (Least Recently Used)[6] (Effelsberg and Harder, 1984) et deux autres : Priority LRU et Priority-Hints (Jauhary et al., 1990; Carey et al., 1989). Les critères de performance utilisés sont le nombre de transactions qui ratent leurs échéances (à minimiser) et le nombre de fautes de pages (à minimiser). Ils ont déduit que l'algorithme PAPER présente de meilleures performances sous diverses conditions de charge. Ils ont cependant constaté que la présence du composant de recherche de données par anticipation n'apporte pas d'amélioration significative, sauf dans quelques cas.

2.5.4 Problème de sauvegarde/restauration

Une des particularités d'un SGBD est sa capacité à remettre une base de données dans un état cohérent, suite à la survenue d'un incident, notamment les incidents du système d'exploitation. Le plus

[5]Prefetching Anticipatorially and Priority basEd Replacement.

[6]La page la moins r´ecemment r´ef´erenc´ee est remplac´ee par une nouvelle.

évident étant la coupure de courant, qui provoque la perte de tout le contenu de la mémoire volatile. Le système aura donc perdu les contenus de tous les tampons qu'il avait en mémoire et qui contiennent des informations aussi importantes qu'une partie des dernières modifications effectuées sur la base, et qui ne sont peut être pas encore reportées sur les fichiers physiques. De même, certains fichiers de la base sur disque pourraient contenir des modifications effectuées par certaines transactions, qui ont été finalement annulées.

Les états décrits précédemment peuvent survenir sur une base de données car les opérations d'écriture sur la base de données sont souvent effectuées de manière asynchrone et ne sont pas effectuées immédiatement au moment de l'opération de Commit (Bernstein et al., 1987).

Suite à un incident donc, le système doit redémarrer avec une base de données incohérente. Pour résoudre les différents problèmes, le SGBD déroule des algorithmes de reprise, qui permettent de "défaire" le travail effectué par des transactions qui n'ont pas encore été validées au moment de l'incident, et *refaire* le travail des transactions qui ont été validées (commit) avant l'incident (rejouer les transactions) mais dont les modifications n'ont pas pu être reportées sur les fichiers de la base. Ceci est possible grâce aux fichiers *journaux* (log) que le SGBD maintient et où toutes les modifications sur la base sont reportées, avant toute opération (images *avant* et images *après*). Ce fichier est stocké sur un support non volatile.

En ce qui concerne les SGBDTR, peu de travaux ont été effectués sur la restauration du système après incident. Ceux qui existent concernent la sauvegarde régulière sur support stable de bases de données temps réel localisées en mémoire centrale. Á cause de la volatilité de la mémoire centrale, la base de donnée pourrait être perdue entièrement en cas d'incident (erreur logicielle, matérielle, coupure de courant, etc). Le processus de journalisation (*logging*) constitue l'une des activités qu'un SGBD en mémoire centrale doit effectuer durant le mode de fonctionnement normal pour garantir la restauration de la base en cas d'incident.

C'est dans ce cadre que Le Gruenwald et al. (Huang and Gruenwald, 1994) ont proposé deux pistes pour effectuer la sauvegarde/restauration dans ces systèmes. Ils ont d'abord proposé dans (Huang and Gruenwald, 1996) d'utiliser une politique de mise à jour immédiate pour les données temporelles, de base ou dérivées (Duvallet et al., 1999b) et une politique de mise à jour différée des données persistantes. Chacune des politiques est employée séparément

Dans (Gruenwald and Cheng, 1997), les auteurs ont amélioré leur précédente technique de sauvegarde en distinguant des types de transactions. Par exemple, ils ne traitent pas les données temporelles de la même manière. Ils distinguent en effet les données temporelles ayant de courts intervalles de validité de celles qui ont des intervalles de validité plus longs. C'est ainsi par exemple qu'ils considèrent qu'il n'est pas nécessaire d'user du temps processeur pour restaurer les données temporelles ayant une courte durée validité. Dans leurs travaux, ils considèrent dans un premier temps deux types de transactions : périodiques et normales. Les transactions périodiques sont responsables soit de la mise à jour des données sensorielles (données de base qui doivent refléter l'état de l'environnement), soit de la lecture de ces données pour dériver, après certains calculs, de nouvelles données (données dérivées).

Ensuite, une catégorisation plus fine a été adoptée pour utiliser leur nouvelle technique de sauvegarde/restauration. Ils considèrent en effet six types de transactions : les transactions sensorielles périodiques qui mettent à jour des données ayant un court intervalle de validité (par rapport à un seuil de référence), les transactions du même type que précédemment qui mettent à jour des données de plus long intervalle de validité, les transactions périodiques dérivant des données ayant un court intervalle de validité, les même que les précédentes mais qui dérivent des données de plus long intervalle de validité,

des transactions normales à contraintes non strictes (*soft*), des transactions normales à contraintes strictes non critiques (*firm*).

Les auteurs ont utilisé pour chaque type de transaction une technique spécifique de sauvegarde/restauration. Les résultats des simulations qu'ils ont effectuées ont montré que leur proposition est appropriée pour la sauvegarde/restauration des bases de données temps réel localisées en mémoire centrale.

Un autre travail sur la sauvegarde/restauration a été effectué par Kiviniemi et al. dans (Kiviniemi et al., 1997). Ils ont en particulier décrit le processus de journalisation qu'ils ont utilisé dans RODAIN[7], un prototype de SGBDTR pour les télécommunications.

RODAIN est un SGBDTR distribué, dont l'architecture consiste en des noeuds-BD autonomes en interaction. Chaque noeud-BD peut communiquer avec une ou plusieurs applications, et une application peut communiquer avec un ou plusieurs noeuds-BD. Un noeud-BD est constitué d'un noeud primaire en mémoire centrale, d'un noeud miroir en mémoire centrale et d'un sous-système secondaire de stockage. Deux types de données sont pris en compte dans une base RODAIN : les données *froides* (*cold*) et les données *chaudes* (*hot*). La copie primaire des données chaudes est localisée en mémoire centrale, la copie primaire des données froides est localisée en mémoire secondaire.

Une base de données RODAIN peut être restaurée suite à un incident de copie primaire ou de copie miroir sans perte de données validées (commit). La copie des données chaudes sur le noeud miroir est mise à jour en utilisant les journaux produits par les transactions validées sur le noeud primaire. Les enregistrements des journaux sont également reportés sur le sous-système secondaire de stockage. De cette manière, une base de donnée RODAIN peut être presque entièrement restaurée suite, soit à un incident de la copie primaire, soit à un incident de la copie miroir. Notons qu'un noeud primaire peut, à la suite d'incident, devenir le noeud miroir, celui-ci devenant à son tour le noeud primaire. Pour davantage de détails sur le SGBDTR RODAIN, le lecteur pourra se référer à (Kiviniemi et al., 1997).

2.5.5 SGBDTR et systèmes d'exploitation

Un SGBDTR doit opérer dans un contexte où des services d'un système d'exploitation (SE) sont disponibles. Le fonctionnement correct du SGBD et le comportement dynamique (temporel) de ses algorithmes de contrôle de concurrence dépendent fortement des services offerts par le SE sur lequel il repose (Kim and Son, 1994; Lehr et al., 1995). Dans les SGBD traditionnels, certaines fonctions sont dupliquées car la plupart des SE-supports n'offrent pas exactement les services nécessaires à ces SGBD, comme la gestion des tampons par exemple. Ce problème est encore plus accru dans les SGBD temps réel où les fonctions sont plus complexes. Par conséquent, une condition nécessaire pour que les SGBD temps réel aient des chances de garantir des contraintes temporelles est de reposer sur un SE temps réel adéquat. Plusieurs types de SE temps réel existent (Ramamritham and Stankovic, 1994). Certains sont des SE dédiés, d'autres sont des extensions de SE multi-tâches existants. Parmi ces systèmes, on peut citer : RT-Mach (Tokuda et al., 1990) qui a servi de noyau au prototype de SGBD temps réel StarBase (Kim and Son, 1997), ARTS (Tokuda and Mercer, 1989) qui a servi de noyau aux prototypes de SGBD temps réel DeeDs (Andler et al., 1996) et RTDB (Son et al., 1993), et Chorus (Bricker et al., 1991), un SE réparti enrichi par des fonctions temps réel (Gien, 1995) qui a servi comme noyau du prototype de SGBD temps réel REACH (Ulusoy and Buchman, 1996).

D'autres SE temps réel peuvent également servir de support à des SGBD temps réel. Par exemple, la réalisation de SGBD temps réel réparti pourrait être facilitée par l'utilisation de plates-formes de type

[7] Real-time Object-oriented Database Architecture for Intelligent Networks

CORBA, étendues par des fonctions temps réel.

Certains auteurs (Graham, 1992) préconisent de travailler sur des architectures nouvelles pour concevoir un SGBD temps réel, de manière à pouvoir intégrer dans le noyau les fonctions de base (ordonnancement des transactions, gestion des entrées/sorties, gestion des tampons, etc.). Un exemple de ce type de SGBD est le prototype RTSORAC (Prichard et al., 1994) qui intègre les fonctions d'un noyau de SE temps réel à la norme POSIX.

2.5.6 Autres problèmes

En plus des travaux que nous avons cité, il existe beaucoup d'autres axes de recherche sur les $SGBDTR$. On peut citer la distribution et réplication des données (Ulusoy, 1994b), la localisation de la base de données entièrement en mémoire centrale ou non (Ulusoy and Buchman, 1998; Ulusoy and Buchman, 1996; Huang and Gruenwald, 1994), la spécification des contraintes temporelles pour les SGBDTR et leur propagation à travers les échéances des transactions et/ou des données (Xiong and Ramamritham, 1997), la maintenance des vues et les mises à jour (Kao et al., 2001), la gestion des disques, notamment l'ordonnancement des entrées/sorties (Kao and Cheng, 2001), les problèmes qui se posent dans des environnements mobiles (Xuan et al., 1997; Datta et al., 1997; Lam and Kuo,), etc.

2.6 Quelques prototypes de SGBDTR

2.6.1 Introduction

Au moment de l'apparition des premiers travaux de recherche sur les SGBDTR, notamment sur le contrôle de concurrence et l'ordonnancement des transactions temps réel, les différents protocoles proposés ont été testés et leurs performances évaluées par simulation. Les premiers simulateurs étaient très rudimentaires et ne prenaient pas en compte les différents paramètres liés aux SGBDTR, en particulier tous les types de contraintes temporelles ne sont pas pris en compte. Ensuite, les simulateurs sont devenus de plus en plus sophistiqués et prennent en compte de plus en plus de paramètres. Les résultats des simulations devenaient ainsi plus réalistes. Cependant, il n'en demeure pas moins qu'il s'agit de résultats de simulations, et donc assez loin des résultats réels.

Depuis, les travaux se sont intensifiés sur les SGBDTR et au cours du premier *workshop* organisé sur les SGBDTR en 1996 (Bestavros et al., 1996b), il a été préconisé de construire plutôt des prototypes de SGBDTR pour effectuer les tests des protocoles proposés et avoir ainsi des résultats plus proches de la réalité. De cette façon, les industriels pourraient être convaincus de la *valeur ajoutée* apportée par ces systèmes par rapport aux SGBD et aux STR traditionnels pour beaucoup d'applications. Depuis cette date, plusieurs prototypes de SGBDTR ont vu le jour (Andler et al., 1996; Arahna et al., 1996; Lortz et al., 2000; Prichard et al., 1994; Stankovic et al., 1997; Zhou et al., 1995; Kim and Son, 1997).

Comme il n'existe pas, à notre connaissance, de SGBD temps réel commercial qui ait les caractéristiques requises[8], nous allons présenter brièvement dans la suite les caractéristiques de certains prototypes.

2.6.2 Prototype DeeDS

DeeDS (Distributed Active Real-Time Database System) est un prototype de SGBDTR actif basé sur le déclenchement d'événements (*event-triggered*) qui utilise des ordonnancements dynamiques d'en-

[8] Beaucoup d'outils du commerce sont pr´esent´es comme des SGBDTR, mais ce sont en g´en´eral des syst`emes rapides et/ou interactifs.

sembles de transactions. Il a été développé à l'université de Skövde, en Suède (Andler et al., 1996). L'aspect actif du système est modélisé selon les règles ECA (Evénement-Condition-action). Dans l'état actuel d'avancement du prototype, il ne tient pas compte de la gestion de données multimédia, ni des intervalles de validité des données.

La prévisibilité est renforcée dans DeeDS en appliquant les contraintes suivantes sur le système et les applications : les bases de données sont entièrement répliquées et localisées en mémoire centrale, évitant ainsi les retards dus à l'accès au réseau et aux disques ; les traitements effectués par les applications sont dépendants uniquement des opérations sur un seul noeud, évitant ainsi les problèmes dus au *Commit* et aux interblocages au niveau global ; les mécanismes de réaction ont des comportements restreints afin de permettre des règles prévisibles d'exécution ; et enfin la gestion de l'ordonnancement et des événements (incluant la composition d'événements) sont traités par un processeur séparé. En outre, un langage de spécification de règles a été développé qui inclut la possibilité de spécifier des contraintes temporelles. Un ordonnancement préemptif dynamique est utilisé gérant de manière mixte les différents types de transactions (*hard, firm et soft*).

L'architecture du prototype DeeDS sépare les fonctions relatives à l'application des services critiques du système. Les fonctions consistent en un module de gestion de règles et un module de sauvegarde d'objets. Les services critiques incluent l'ordonnancement et le superviseur d'événements.

Les règles sont étendues avec la spécification de contraintes temporelles sur les actions déclenchées. Aux règles sont associés des événements spécifiques, pour réduire les déclenchements inutiles de règles. La recherche des règles à déclencher devient ainsi plus efficace. De même pour rendre prédictibles l'exécution des règles, il y a une restriction dans les modes de couplage et les déclenchements en cascade de règles sont limités.

Dans DeeDS, s'exécutent deux types de transactions : critiques et non critiques. Chaque transaction de type critique possède sa transaction *contingente* dont les besoins en ressources sont plus limités que l'originale et qui est exécutée à la place de l'originale durant les phases de surcharge du système. Les services critiques *hard* s'exécutent sur un processeur dédié pour gérer de manière efficace les événements critiques et améliorer la concurrence.

Le prototype DeeDS a été implémenté sur un noyau OSE Delta d'un système d'exploitation UNIX. Actuellement, dans le prototype DeeDS, est en train d'être implémenté le modèle de transactions imbriquées (Moss, 1985).

2.6.3 Prototype BeeHive

BeeHive (Stankovic et al., 1997) est une collection de services logiciels supportant une certaine vision des bases de données actuelles. Ces services s'entendent en termes d'environnements distribués aussi bien au niveau base de données, qu'au niveau gestion des ressources, au dessous du niveau base de données. Cela inclut la garantie point-à-point des diverses applications temps réel et non temps réel, fonctionnant sur le réseau Internet actuel ou sur la future génération.

Le système BeeHive possède plusieurs composants. Parmi celles-ci, on peut citer :

- un support pour le temps réel, comprenant la nouvelle notion d'échéances des données (en plus des échéances des transactions) ;
- la restauration parallèle et temps réel de la base, basée sur la sémantique des données et sur le mode opérationnel du système (ex. mode dégradé) ;

- l'utilisation d'informations dupliquées et d'un langage de spécification pour tenir compte de la tolérance adaptative aux fautes, des performances temps réel et de la sécurité ;
- l'idée d'encapsuler les règles de sécurité dans les objets, et la possibilité pour ces règles d'utiliser des profiles de types variés ;
- des objets composables tolérant aux fautes qui opèrent en synergie avec les propriétés des transactions dans la base et avec les mécanismes de journalisation/restauration ;
- une nouvelle architecture et un nouveau modèle d'interaction entre les calculs multimédia et transactionnels ;
- un modèle uniforme de tâches pour supporter simultanément le contrôle des tâches temps réel strictes et les traitements point-à-point multimédia ;
- de nouveaux ordonnancements basés sur la qualité de service, la gestion des ressources et les algorithmes de re-négociation.

Le prototype BeeHive a été construit autour des caractéristiques précédentes combinées pour former la conception d'une nouvelle base de données virtuelle. Beehive comporte trois composantes : Beehive 'natif', des sites portés sur beeHive et des interfaces avec des SGBD existants.

Dans BeeHive natif, il y a plusieurs niveaux. Au niveau application, l'utilisateur peut soumettre ses transactions, ses programmes ou ses accès audio/vidéo. Pour chaque donnée entrée, l'utilisateur peut spécifier des caractéristiques temps réel comme l'échéance, la qualité de service (QdS), la tolérance aux fautes, et le niveau de sécurité.

En termes de conception, les éléments suivants sont pris en compte :
- Le développement d'un haut niveau de spécification, et la manière dont les besoins interagissent.
- L'implémentation d'un support base de données orientée objet.
- la conception et l'implémentation d'une nouvelle sémantiques pour les objets.
- la conception et l'implémentation de médiateurs orientés objet.
- la conception et l'implémentation du gestionnaire de ressources et des sous-systèmes de tolérance aux fautes et de sécurité.

Dans la version finale du prototype, les auteurs comptent implémenter des aspects actifs sous forme de règles ECA.

2.6.4 Prototype RTSORAC

2.6.4.1 Caractéristiques

RTSORAC (Real-Time Semantic Objects Relationships and Constraints) (Prichard et al., 1994) est un prototype de SGBDTR implémenté au dessus d'un SGBD objet : Open OODB (Wells et al., 1992). Le modèle RTSORAC permet de spécifier les objets, les transactions et les associations entre les objets.

Un objet est caractérisé, entre autres, par ses attributs, ses méthodes, ses contraintes, et une fonction de compatibilité.

Les notions d'attributs et de méthodes sont celles utilisées dans le paradigme *objet*. Les contraintes permettent de spécifier les états corrects d'un objet. Par exemple, un objet *train*, pourrait avoir l'attribut HuilePr.temps (Pression d'huile) mis à jour toutes les 30 secondes. Pour maintenir la cohérence temporelle de cette donnée, on définit une règle de la forme :

Contrainte : HuilePr.temps $>$ Now - 30 secondes
Règle : Si Missed $\leq$ 2 **Alors**
HuilePr.temps = Now
Missed = Missed + 1
Signal HuilePr_Warning
Sinon Signal HuilePr_Alert

Cette règle spécifie que si une ou deux échéances d'opérations de lecture du capteur de pression sont manquées, alors incrémenter le nombre d'échéances de lectures manquées et Signaler un avertissement. S'il y a plus de deux lectures manquées, Signaler une alerte. Le compteur Missed est réinitialisé à 0 quand une lecture a eu lieu.

Matrice de compatibilité : elle est utilisée par le contrôleur de concurrence pour détecter les méthodes des objets qui sont incompatibles entre elles. Dans RTSORAC, on peut définir des imprécisions que l'on peut tolérer, par exemple entre une action de lecture et une action d'écriture. La fonction de compatibilité évalue une expression qui prend en compte les attributs des objets, les types d'actions, les contraintes, ... et renvoie un résultat booléen, qui autorise ou non deux méthodes à s'exécuter en concurrence.

Associations : ce sont des relations qui représentent l'agrégation de deux ou plusieurs objets. Dans l'exemple du train, on peut définir une relation *Gestion de l'énergie* qui associe l'objet *train* et l'objet *trace*. Ce dernier mémorise des informations relatives au train comme les limites de vitesse, la charge maximale, la puissance disponible. La relation Gestion de l'énergie exploite les information des objets *train* et *trace* pour déterminer par exemple les paramètres d'un algorithme de contrôle de la consommation de fuel ou de freinage.

Transactions : ce sont des structures composées, entre autres, d'un ensemble d'opérations (ordres dans un langage de manipulation et méthodes). Chaque opération possède dans ses arguments l'imprécision maximale tolérée de l'argument ainsi que son échéance.

Contraintes : on peut également spécifier pour une opération des contraintes de précédence ou d'exécution. Une transaction peut posséder (1) des pré-conditions, qui doivent être satisfaites avant l'exécution, (2) des post-conditions qui doivent être satisfaites après l'exécution, et (3) le résultat. Une pré-condition pourrait être le déclenchement d'un événement, comme dans les SGBD actifs ou la fin d'une autre transaction. Une post-condition pourrait, par exemple, être utilisée pour spécifier ce que doit faire l'opération Commit d'une transaction ayant des sous-transactions. Le résultat pourrait être une valeur lue ou calculée.

2.6.4.2 Architecture

Du point de vue architecture, les modifications apportées par l'implémentation du prototype RT-SORAC sur open DB sont notamment l'utilisation d'un système d'exploitation POSIX avec des extensions temps réel, et l'incorporation d'un sous-système de stockage persistant temps réel. Des politiques de gestion des transactions temps réel et d'objets temps réel ont également été implémentées.

Pour le contrôle de concurrence, des techniques de verrouillage sémantique d'objets sont ajoutées, qui permettent d'implémenter la cohérence logique, et/ou la cohérence temporelle, ou de spécifier un compromis entre les deux.

Dans la gestion des transactions de RTSORAC, les ordonnancements effectués au niveau du SGBDTR sont transformés en priorités du système d'exploitation temps réel POSIX.

2.6.4.3 Langage RTSQL

RTSQL (Prichard and Fortier, 1997) est une extension du langage SQL standard pour supporter les caractéristiques des bases de données temps réel. Il a été spécifié par un groupe de l'ANSI-SGBD et de l'US Navy en 1994. Il exprime les contraintes, le temps, et une structure de transaction rudimentaire. Il supporte la plupart des caractéristique du SGBDTR RTSORAC.

Les attributs de spécification du temps proposés dans RTSQL sont : DATE, TIME, TIMESTAMP, INTERVAL, CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP, liés éventuellement par les opérateurs +, -, <, >.

Par exemple, l'expression de la validité temporelle relative concernant les attributs *position* et *vitesse* d'un objet, peut être spécifiée de la manière suivante avec RTSQL :

```
CONSTRAINT speed_bearing_rvi
    CHECK TIMESTAMP(speed) BETWEEN
        TIMESTAMP (bearing) - INTERVAL '2' SECOND
      AND TIMESTAMP (bearing) + INTERVAL '2' SECOND
    AFTER CONTACT_MADE
```

La contrainte speed-bearing_rvi devient active après l 'instant CONTACT_MADE (AFTER ...)

La forme générale d'une spécification de contraintes d'exécution est la suivante :

```
< time_constraint_clause> ::=
        [ START BEFORE < expression date >]
        [ START AFTER <expression date> ]
        [ COMPLETE BEFORE <expression date> ]
        [ COMPLETE AFTER <expression date> ]
        [ PERIOD <intervalle> /* indique une exécution périodique*/
        [ START AT <expression de temps> ]
        [ UNTIL <expression booléenne> ]
```

Exemple :

```
 X : BEGIN
        SELECT price FROM stocks WHERE name='Acme'
        COMPLETE BEFORE CURRENT_TIMESTAMP + INTERVAL '30' SECOND;
        – autres instructions
 END X COMPLETE BEFORE CURRENT TIMESTAMP + INTERVAL '1' MINUTE;
```

Dans la clause SELECT, l'exécution doit se terminer avant 30 secondes. Les autres instructions du bloc doivent se terminer avant 1 minute, depuis un instant de référence.

Pour détecter une violation de contrainte de temps, des horloges sont utilisées.

exemple : START BEFORE CURRENT_TIMESTAMP + INTERVAL '1' MINUTE

Quand l'instruction est rencontrée, l'horloge est initialisée. L'exécution de l'instruction doit commencer dans la minute qui suit.

RTSQL : permet d'utiliser des directives, pour supporter la prévisibilité nécessaire aux SGBD temps réel. Par exemple, une température, pour être correcte, doit être recherchée depuis la base de données en quelque millisecondes. Pour cela une directive peut indiquer au système que cette variable doit toujours résider en MC pour éviter les accès disque (trop lents).

Structure d 'une transaction dans RTSQL

- Pré-condition : est la condition de déclenchement de la transaction
- Spécification : spécifie les structures de données, besoins en temps, limitation de ressources, criticité, préemptabilité,...
- Corps : est le code d'accès à la base et celui des actions
- Post-condition : elle définit la correction de la transaction (commit)
- Corps pour la restauration : définit ce qu'il convient de faire restaurer en cas d 'incident

En conclusion, on peut dire que RTSQL représente la première (et la seule à notre connaissance) tentative de standardisation d'un langage pour les bases de données temps réel. Il a été implémenté sur le prototype de SGBDTR RTSORAC. La tentative de standardisation n'a pas encore abouti, mais il y a de fortes chances que le langage RTSQL soit adopté dans un proche avenir.

2.7 Conclusion

Dans ce chapitre, nous avons passé en revue quelques caractéristiques des $SGBDTR$, puis nous avons cité quelques applications où l'utilisation de ces systèmes est particulièrement recommandée. Nous avons ensuite cité les principaux problèmes rencontrés lors de la conception des $SGBDTR$ ainsi que quelques solutions apportées. Les travaux concernant le contrôle de concurrence et ordonnancement des transactions dans ces systèmes sont juste évoqués. Ils seront décrits avec davantage de détails dans le chapitre 3 où l'on décrit également notre contribution dans ce domaine. Nous avons ensuite décrit quelques prototypes de recherche implantant la plupart des fonctionnalités de ces systèmes.

Nous pouvons dire que de nos jours, de nombreux travaux ont été effectués sur les $SGBDTR$ et la profusion de prototypes de recherche préfigure la conception et la réalisation de $SGBDTR$ commerciaux. Cependant, pour que ceux-ci voient le jour, une certaine standardisation est nécessaire. Elle va en effet apporter une base commune sur laquelle les concepteurs et développeurs de $SGBDTR$ pourront s'appuyer. On évitera ainsi la multitude de SGBD actuels dont les constructeurs vantent les qualités *temps réel* alors qu'en général, soit ils effectuent des traitements rapides, soit ils utilisent des données avec durée de validité, ou encore ils initialisent les échéances à respecter pour effectuer une certaine tâche. Dans ce dernier cas, la valeur de l'échéance est mise à jour dès qu'elle commence à se rapprocher, risquant ainsi de ne pas être respectée !

À notre connaissance, une seule équipe de recherche (Prichard et al., 1994) a commencé à travailler sur la standardisation en proposant le prototype RTSORAC que nous avons présenté, dans lequel ils ont implémenté un langage de type SQL augmenté avec des ordres relatifs à la spécification des contraintes temporelles (Prichard and Fortier, 1997).

D'autres travaux relatifs à la standardisation des composants des SGBDTR et leur langage sont nécessaires pour pouvoir passer à l'étape de développement de $SGBDTR$ commerciaux.

3 Contrôle de concurrence et ordonnancement des transactions temps réel

3.1 Problématique

Dans les bases de données temps réel, tous les ordonnancements sérialisables ne sont pas acceptables : ceux qui ne respectent pas les contraintes temporelles des transactions sont rejetés. Par conséquent, les techniques de contrôle de concurrence de transactions (désignées par CC dans la suite du document) développées dans les SGBD traditionnels ne sont pas directement applicables pour les SGBD temps réel. En effet, dans une base de données temps réel, il est nécessaire de maintenir, en plus de la cohérence logique des données, leur cohérence temporelle. D'autre part, comme nous l'avons déjà évoqué, les algorithmes conçus pour l'ordonnancement des tâches temps réel (Stankovic et al., 1995; Cottet et al., 2000) ne peuvent pas être appliqués tels quels aux transactions des bases de données pour intégrer les contraintes temporelles.

Beaucoup de travaux sur le CC des transactions dans les SGBD temps réel s'appuient sur les techniques traditionnelles de CC des transactions dans un SGBD d'une part, et sur les techniques d'ordonnancement des tâches dans les STR d'autre part. Ces travaux se basent aussi bien sur des politiques de CC optimistes que sur des politiques pessimistes.

A priori, on pourrait penser que les protocoles basés sur les politiques pessimistes ne doivent pas pouvoir être utilisés dans un contexte temps réel. Cependant, comme on le verra dans la suite, cela dépend de beaucoup de paramètres et on ne peut pas conclure de manière triviale à la supériorité d'un type de protocole sur un autre. C'est ainsi que parfois les performances des protocoles optimistes s'étaient avérées meilleures que celles des protocoles pessimistes, et d'autres fois, c'est l'inverse qui arrive.

Dans la section suivante, nous dressons un panorama général des protocoles utilisés dans les SGBDTR pour gérer les conflits d'accès potentiels aux mêmes données par plusieurs transactions.

3.2 Panorama des protocoles de CC des transactions temps réel

Dans le cas des SGBDTR, plusieurs protocoles ont été proposés pour gérer les conflits d'accès aux données par les transactions, tout en améliorant le respect des contraintes temporelles (des transactions et des données). Ces protocoles sont souvent des adaptations au contexte temps réel des protocoles existant dans les SGBD traditionnels. La principale adaptation est la prise en compte des connaissances sur les contraintes temporelles lors de la résolution des conflits. Cela passe par une collaboration étroite entre le contrôleur de concurrence et l'ordonnanceur pour tenir compte des priorités des transactions lors des prises de décision pour exécuter telle ou telle transaction.

Dans le domaine des SGBDTR, on distingue plusieurs types de protocoles de CC dont certains sont issus des protocoles utilisés dans les SGBD traditionnels comme les protocoles bloquants, les protocoles optimistes ou l'approche multi-versions, d'autres ont été proposés spécialement pour les SGBDTR, comme l'approche spéculative ou l'approche d'ajustement dynamique de l'ordre de sérialisation.

Dans la suite de cette section, nous allons passer en revue les protocoles les plus significatifs dans chacune des catégories précédentes. Nous décrirons ensuite les protocoles que nous avons développés pour atteindre le même objectif.

3.2.1 Protocoles bloquants

Des adaptations du protocoles 2PL, auquel des techniques d'assignation de priorités sont ajoutées, ont donné naissance à des protocoles de CC pour les SGBDTR. Nous en citerons quelques uns, qui correspondent à deux principales familles d'algorithmes : l'abandon par priorité et l'héritage de priorité.

3.2.1.1 Protocole 2PL-HP

2PL-HP (Two-Phase Locking-High Priority) est appelé également protocole d'abandon par priorité. Cet algorithme proposé dans (Haritsa et al., 1992) consiste à abandonner la transaction de plus basse priorité quand le phénomène d'inversion de priorité[1] (Liu and Leyland, 1973) survient. Ceci assure que les transactions de plus haute priorité ne soient pas retardées par des transactions de plus basses priorités. Si la priorité d'une transaction qui désire acquérir un verrou sur un objet est plus faible que celle de chacune des transactions détentrices du verrou sur l'objet considéré dans un mode incompatible, alors cette transaction doit attendre que l'objet soit libéré (comme dans l'algorithme 2PL traditionnel). En outre, une nouvelle transaction de lecture ne peut être ajoutée à l'ensemble des transactions de lecture qui détiennent un verrou de lecture sur un objet que si elle a une priorité plus élevée que toute transaction d'écriture en attente d'accès à cet objet.

Ce protocole assure que davantage de transactions de plus haute priorité se terminent dans les temps.

L'inconvénient est que des transactions peuvent être abandonnées à cause de l'exécution d'une transaction de plus haute priorité qui, plus tard, peut ne pas se terminer avant son échéance. Ce qui conduit à une perte de travail et donc à une baisse des performances du système.

3.2.1.2 Protocole Wait-Promote

Il est également appelé protocole d'héritage de priorité. L'algorithme d'héritage de priorité (Sha et al., 1990; Haritsa et al., 1992) permet d'assigner une nouvelle priorité à toute transaction accédant à un objet partagé, lui permettant de s'exécuter à un niveau de priorité plus élevé que celui des transactions qu'elle bloque. On parle alors d'héritage de priorité (ou de promotion de priorité). A chaque fois que le phénomène d'inversion de priorité survient, la transaction de plus faible priorité voit sa priorité promue et ramenée au niveau de celle de la transaction demandeuse du verrou, jusqu'à ce qu'elle se termine et libère le verrou. Les transactions détentrices de verrous pourraient se terminer plus tôt, réduisant ainsi les temps d'attente des transactions de plus hautes priorités.

Les inconvénients de ce protocole sont notamment : (1) le temps de blocage des transactions de haute priorité qui reste imprévisible, particulièrement lorsque surviennent des blocages en chaîne, c'est-à-dire une transaction de haute priorité qui se bloque plusieurs fois durant son exécution en attente des

[1] Une transaction de priorit´e ´elev´ee bloqu´ee par des transactions de plus faibles priorit´es.

transactions de plus basses priorités (dont les priorités sont promues à des priorités plus élevées), (2) les transactions de faible priorité ayant hérité de priorité plus élevée pour s'exécuter risquent de concurrencer les transactions à haute priorité pour l'accès aux autres ressources du système (CPU, organes d'Entrée/Sortie, etc.) et peuvent les conduire ainsi à manquer leurs échéances.

Une variante de ce protocole s'appelle l'héritage conditionnel de priorités (Huang and Stankovic, 1990). Il s'agit d'un protocole où une transaction de faible priorité n'hérite d'une haute priorité que dans le cas où elle est proche de sa terminaison. Autrement, elle est abandonnée. L'avantage est qu'il y a moins de travail perdu quand la transaction est loin de sa terminaison, et les délais d'attente sont réduits pour les transactions de haute priorité (qui ne sont plus bloquées par des transactions de plus faibles priorités auxquelles il reste beaucoup de temps pour se terminer).

Les simulations effectuées par Huang et al. (Huang and Gruenwald, 1996) ont montré que ce protocole présente de meilleures performances que celui de l'héritage de priorité simple ou que celui de l'abandon par priorité.

3.2.1.3 Protocole PCP (Priority Ceiling Protocol)

C'est un protocole qui utilise les priorités plafonds, comme définies dans l'ordonnancement PCP des tâches temps réel (Sha et al., 1987). Il s'agit d'une extension du protocole de l'héritage de priorité qui élimine les interblocages et limite la durée de l'inversion de priorité. Il est conçu à l'origine pour l'ordonnancement des tâches dans les STR (Sha et al., 1987), où chaque ressource se voit attribuer une priorité égale à la plus haute priorité des tâches susceptibles d'y accéder : sa priorité-plafond. Plusieurs adaptations de ce protocole aux SGBD temps réel ont été effectuées.

Sha et al. (Sha et al., 1991) ont conçu le protocole RW-PCP (Read/Write-PCP) pour le contexte temps réel strict et critique où ils exploitent la sémantique des opérations de lecture/écriture pour l'attribution de priorités-plafonds aux données. Les algorithmes fondés sur ce protocole garantissent qu'une transaction ne peut être bloquée que par une seule autre transaction de plus faible priorité. Cependant, ce protocole peut engendrer des blocages inutiles générant ainsi des retards dans l'exécution des transactions.

Une amélioration du RW-PCP a été proposée par Nazakato et al. (Nakazato, 1993) qui ont introduit le protocole CCP (Convex Ceiling Protocol). Ce protocole réduit les durées de blocage en déverrouillant les données de plus haute priorité-plafond avant la fin de la transaction qui y accède si celle-ci a fini de les utiliser. Malgré cette amélioration, le protocole CCP souffre toujours de la présence de blocages inutiles. Par ailleurs, ces deux protocoles présentent l'inconvénient qui consiste à ne donner qu'une borne statique de la durée de l'inversion de priorité, indépendante du comportement réel des transactions. La promotion d'une transaction de faible priorité vers une priorité plus haute est déterminée de manière statique (en se basant sur tous les accès potentiels des transactions aux données) et non sur le comportement dynamique du système.

Par la suite, Lam et al. ont proposé une nouvelle variante du protocole PCP qui améliore les deux précédentes, en évitant des blocages inutiles des transactions (Lam et al., 1997a).

3.2.2 Protocoles optimistes

Dans les systèmes où ces protocoles sont implantés, toutes les transactions sont servies dès qu'elles en font la demande. Nous rappelons que les écritures sont différées jusqu'à l'instant de la validation de chaque transaction. De plus, avant sa validation, chaque transaction doit passer un test de certification

afin de détecter les conflits éventuels avec d'autres transactions qui auraient effectué la validation depuis le début de son exécution (certification en arrière), ou avec des transactions actives (certification en avant). Une transaction passe donc par trois phases : lecture, validation (certification) et écriture.

Pendant le test de certification en avant, les objets écrits par la transaction courante T_c sont comparés à tous les objets lus par toute transaction active. S'il y a des objets communs entre T_c et ces transactions alors une erreur de sérialisation est détectée. Il est, dans ce cas, possible d'abandonner soit la transaction T_c, soit toutes les transactions avec lesquelles elle est en conflit.

Ce type de certification est plus adapté au contexte temps réel, car il laisse le choix quant aux transactions à abandonner, contrairement à la certification en arrière où la seule possibilité est d'abandonner la transaction courante.

Dans l'algorithme OCC de base (Kung and Robinson, 1981), les conflits entre transactions ne sont détectés qu'au moment du test de certification. Les ressources détenues par des transactions qui atteignent cette phase et qui devront être redémarrées, ont donc été accaparées pour rien.

Afin de minimiser les pertes de ressources et de redémarrer les transactions le plus tôt possible, Menasce et al. (Menasce and Nakanishi, 1982) ont proposé l'algorithme OCC-BC (Optimistic Concurrency Control-Broadcast Commit), une variante de la certification en avant de l'algorithme OCC. L'algorithme OCC-BC utilise la notification qui consiste à ce que la transaction qui effectue une validation avertisse de sa validation toutes les transactions avec lesquelles elle est en conflit. Ces transactions sont redémarrées immédiatement améliorant ainsi les performances du système, en termes de temps de réponse moyen des transactions.

Plusieurs protocoles temps réel dérivés du protocole OCC-BC ont été proposés. Dans la suite, nous en citons quelques uns et en donnons une brève description.

3.2.2.1 Protocole OCC-BC temps réel

Dans (Haritsa et al., 1992), Haritsa et al. ont proposé une variante de l'algorithme OCC-BC qui utilise la certification en avant, tenant compte des priorités des transactions. Considérons une transaction T_c en phase de validation. Si, avant sa validation, cette transaction entre en conflit avec d'autres transactions T_1, T_2, ..., T_n, deux cas peuvent se présenter :

```
Cas 1 : Si priorité (Tc) > priorité (Ti) i = 1,...,n (i ≠ c)
          Alors les transactions Ti sont abandonnées.
Cas 2 : Si il existe des transactions Tk (k ∈ [1,n])
          telles que priorité (Tk) > priorité (Tc),
          Alors Tc doit attendre la validation de ces
              transactions.
```

L'avantage vient du fait que la transaction T_c se met en attente et garde donc les ressources qu'elle avait acquises. Ce qui améliore les performances du système, puisque la transaction n'aura pas à redemander ces ressources. Un inconvénient potentiel est que les transactions abandonnées (1^{er} cas) sont redémarrées plus tard, et ont donc moins de chances de se terminer avant leurs échéances que si elles n'avaient pas été abandonnées. Il en résulte du travail inutile pour le système.

Le deuxième cas présente aussi un inconvénient qui provient du fait que la transaction en phase de validation (T_c) détient des ressources alors qu'elle est bloquée. La conséquence peut être une aug-

mentation du risque de conflits potentiels entre les transactions. Ces ressources sont en quelque sorte verrouillées pendant une période de temps, augmentant ainsi le risque de conflits entre transactions.

3.2.2.2 Protocole OPT-Wait

Il s'agit d'une variante du protocole OCC-BC temps réel. Une transaction qui atteint sa phase de validation et trouve des transactions de plus hautes priorités dans son ensemble de conflits est forcée à attendre. Cette période d'attente pourrait permettre aux transactions de priorités plus élevées de se terminer avant leurs échéances. Il peut arriver que la transaction courante soit redémarrée, suite à la validation (Commit) d'une au moins des transactions de plus haute priorité.

Cependant, le protocole OPT-Wait possède certaines caractéristiques qui peuvent s'avérer avoir un impact positif sur les performances d'un SGBDTR : (1) les transactions de plus haute priorité s'exécutent d'abord, leur donnant ainsi plus de chances de respecter leurs échéances, (2) le problème du gaspillage de ressources ne pourra pas survenir car la transaction qui attend n'est redémarrée par aucune autre transaction de plus haute priorité dont l'échéance n'a pas été respectée, (3) lorsque toutes les transactions de plus haute priorité manquent leurs échéances, la transaction courante pourrait valider.

3.2.2.3 Protocole Wait-50

Il améliore le protocole OPT-Wait. Il a été proposé par Haritsa et al. (Haritsa et al., 1992), qu'ils présentent comme une stratégie de compromis. En plus de *l'attente par priorité*, ils ont ajouté un mécanisme de *contrôle de l'attente*. Ainsi, en reprenant la notation du paragraphe précédent, la transaction T_c doit se mettre en attente jusqu'à ce que seulement moins de 50 % des transactions T_i (i=1,...,n) aient des priorités supérieures à la sienne. Une fois que ce stade est atteint, les transactions T_i restantes sont abandonnées sans tenir compte de leurs priorités, et la transaction T_c réalise sa validation.

Dans (Haritsa et al., 1992; Huang and Gruenwald, 1996), Haritsa et al. et Huang et al. ont comparé les techniques précédentes avec les protocoles bloquants et ont abouti à des conclusions contradictoires. En effet, pour Haritsa et al., la conclusion est que les protocoles optimistes présentent de meilleures performances que les protocoles bloquants, notamment dans des environnements où sont écartées les transactions qui ne peuvent pas respecter leurs échéances. Pour Huang et al., on arrive à la conclusion contraire mais dans des environnements surchargés. La raison de ces conclusions contradictoires est due à la différence de nature des systèmes et des hypothèses utilisées dans l'évaluation des protocoles.

3.2.2.4 Protocole OPT-Sacrifice

Le protocole OPT-Sacrifice modifie le protocole OCC-BC en y incorporant un mécanisme de sacrifice par priorité. On définit l'*ensemble de conflits* comme l'ensemble des transactions en exécution qui sont en conflit avec la transaction en cours de validation. Une transaction ayant atteint sa phase de validation, vérifie si des conflits existent avec les transactions en cours d'exécution. Si c'est le cas et qu'une au moins des transactions a une priorité plus élevée que la sienne, alors elle est abandonnée et redémarrée. La transaction en cours de validation se "sacrifie" dans l'espoir que les transactions de plus hautes priorités avec lesquelles elle est en conflit puissent se terminer avec succès.

3.2.2.5 Protocole CCA

L'algorithme CCA (Cost Conscious Approach) est proposé par Hong et al. dans (Hong et al., 1993), pour résoudre le problème de contrôle de concurrence et d'ordonnancement des transactions

temps réel. Cet algorithme améliore les performances des algorithmes EDF-HP[2] et EDF-CR [3] (Abbot and Garcia-Molina, 1988), deux des premiers algorithmes conçus pour l'ordonnancement des transactions temps réel.

Dans l'algorithme EDF-HP, seule l'information concernant l'échéance d'une transaction est prise en compte pour résoudre des conflits en donnant la priorité à la transaction dont l'échéance est imminente. Des performances meilleures sont obtenues avec l'algorithme EDF-CR car il dispose, comme information supplémentaire, d'une estimation du temps d'exécution des transactions. Il permet ainsi à des transactions non prioritaires de se terminer à condition que cela ne conduise pas les autres transactions en conflit, ayant une plus haute priorité, à rater leurs échéances.

Le principe d'EDF-CR est le suivant :

Soit T_d une transaction détentrice d'un verrou sur une donnée d et soit T_r une transaction de plus haute priorité que T_d désirant accéder à d dans un mode incompatible.

- L'algorithme fait une estimation du temps restant à T_d pour se terminer.
  S'il estime que T_d peut terminer son exécution sans remettre en cause le respect de l'échéance de T_r,
- Alors il l'autorise à poursuivre son exécution évitant ainsi un abandon (peut-être) inutile,
- Sinon il abandonne T_d.

L'algorithme CCA prend en compte non seulement les aspects statiques de l'exécution des transactions (échéances, ...), mais aussi des aspects dynamiques (temps effectif d'exécution, coût des redémarrages, ...) en recalculant en cours d'exécution la priorité des transactions. CCA est fondé sur une préanalyse des transactions temps réel qui permet de connaître précisément la structure de celles-ci ainsi que les données potentiellement accédées en fonction des chemins d'exécution. La préanalyse consiste à effectuer une analyse syntaxique des transactions afin de construire leur arbre de décision. Des exemples qui illustrent le principe de cette préanalyse peuvent être trouvés dans (Duvallet et al., 1999b).

Cependant, le facteur *charge du système* n'est pas pris en compte dans l'algorithme CCA de base. Une extension prenant en compte ce facteur a été proposée par Chakravarthy et al. dans (Chakravarty et al., 1998). Ils ont proposé l'algorithme CCA-ALF (CCA-Average Load Factor) qui permet de prendre en considération la charge du système au moment de l'exécution des transactions et donc de recalculer dynamiquement les priorités des transactions en fonction des ressources système disponibles.

Notons que l'estimation dynamique des coûts permet d'assigner les priorités aux transactions de façon plus équitable que d'autres algorithmes car elle est plus proche de la réalité des exécutions. De plus, CCA tient compte des coûts des annulations et des redémarrages des transactions abandonnées. Des simulations ont été effectuées dans ce sens dans (Chakravarty et al., 1998) qui montrent que CCA est plus performant que EDF-HP et EDF-CR pour les transactions à échéances non strictes. Pour les transactions à échéances strictes non critiques, CCA présente de meilleures performances que EDF-HP.

Ces performances sont notamment perceptibles dans le cas des bases de données résidant sur disque. La non prise en compte du coût de la préanalyse des transactions peut s'avérer négative pour l'évaluation des performances ; néanmoins puisqu'il s'agit d'une préanalyse qui peut être effectuée lors de la compilation des transactions, on peut considérer qu'elle n'intervient pas dans l'exécution des transactions et ne remet pas en cause les apports de l'aspect dynamique de CCA.

[2] Earliest Deadline First-High Priority.
[3] Conditional Restart.

Cependant, comme le soulignent les auteurs, les simulations effectuées considèrent que les verrous sur les données sont exclusifs et que toutes les transactions ont le même niveau de criticité. Donc, l'utilisation de verrous partagés et des niveaux multiples de criticité des transactions pourrait affecter les résultats obtenus.

3.2.3 Protocoles spéculatifs SCC (Speculative Concurrency Control)

SCC est une nouvelle classe de protocoles de contrôle de concurrence conçue spécialement pour les SGBD temps réel (Bestavros, 1992; Braoudakis, 1995). Les algorithmes de cette classe sont fondés sur la redondance des ressources. L'objectif est de trouver le plus tôt possible des ordonnancements sérialisables et de les mettre en oeuvre, augmentant ainsi les chances pour les transactions du système de respecter leurs échéances.

Les algorithmes SCC présentent les avantages des protocoles pessimistes (ils détectent les conflits éventuels le plus tôt possible) et les avantages des protocoles optimistes (ils permettent à des transactions concurrentes de s'exécuter, leur évitant ainsi des attentes inutiles qui risqueraient de compromettre leurs chances de respecter leurs échéances). L'inconvénient est qu'ils nécessitent la disponibilité de ressources abondantes. Cette hypothèse, souvent inacceptable dans les systèmes conventionnels, devient très plausible pour les systèmes temps réel, où souvent sont mis en jeu des problèmes de sécurité, des vies humaines, etc.

On a vu que dans un algorithme optimiste de type OCC classique, l'exécution d'une transaction comporte trois phases : lecture, validation, écriture. Le sort d'une transaction n'est déterminé qu'à la phase de validation, aussi appelée certification. En effet, c'est seulement à ce moment-là qu'elle est redémarrée (si elle n'a pas satisfait à son test de certification), ou bien que sa validation est effectuée (phase de modification de la base, rendant ainsi visibles les modifications qu'elle aura effectuées).

Dans un contexte temps réel et, étant donné que les conflits ne sont détectés qu'à la phase de certification avec les protocoles OCC, à partir de quel instant peut-on dire qu'il est trop tard pour redémarrer une transaction ?

Dans les protocoles pessimistes, ce type de problème ne se pose pas car ils détectent les conflits dès qu'ils surviennent. Il se pose par contre un autre problème : beaucoup de transactions risquent de rater leurs échéances à cause des durées non bornées des attentes. L'algorithme OCC-BC améliore l'algorithme OCC de base en utilisant la notification (diffusion) (cf. paragraphe 3.2.2.1).

Le premier protocole conçu suivant l'approche SCC est SCC-OB (Speculative Concurrency Control-Order Based). Il est basé sur l'ordre de sérialisation des opérations des transactions (lecture et écriture). Il franchit une étape de plus dans l'utilisation des connaissances sur les conflits entre transactions. En effet, au lieu d'attendre une éventuelle incohérence de la base due à l'exécution concurrente des transactions, ce protocole *spécule* sur des mesures correctives à prendre dès le début du conflit, en se servant des ressources redondantes. Selon les auteurs de SCC-OB, en prenant des mesures le plus tôt possible, on augmente les chances des transactions à respecter leurs contraintes de temps. Le protocole explore les ordonnancements sérialisables potentiels le plus tôt possible pour augmenter ainsi la possibilité d'adopter celui qui se déroule sans qu'aucune transaction ne rate son échéance.

Brièvement, un exemple de fonctionnement du protocole SCC-OB est le suivant : Soient deux transactions T_1 et T_2 accédant, respectivement, en écriture et en lecture à une donnée x. Deux scénarios sont possibles selon le temps mis par la transaction T_2 à atteindre son étape de certification. À l'instant où la

transaction T_2 fait une requête pour lire x, toutes les informations nécessaires pour détecter la présence de conflit entre T_1 et T_2 sont disponibles (puisque l'on sait déjà que T_1 a fait une requête d'écriture de x). Au lieu de bloquer T_2 comme dans les protocoles pessimistes ou d'ignorer le conflit jusqu'à la certification de T_2 comme dans les protocoles optimistes, les auteurs suggèrent d'utiliser les ressources dupliquées en créant une copie de la transaction de lecture (ici T_2). La transaction T_2 d'origine continue à s'exécuter sans interruption alors que la copie T_2' est redémarrée pour s'exécuter sur un processeur différent. Ainsi, deux versions d'une même transaction s'exécutent en parallèle, sachant évidemment que l'une d'elles seulement va effectuer sa validation et l'autre va être abandonnée.

En conclusion, on peut dire que le protocole SCC-OB augmente les chances que les transactions se terminent avant leurs échéances. Cependant, il est très gourmand en ressources, car il est fondé sur la duplication des transactions qui s'exécuteraient sur des processeurs différents. Il est donc destiné à des architectures spécialisées de systèmes temps réel. Des variantes de ce protocole ont été développées pour réduire le nombre de duplications de ressources : SCC-CB, SCC-DC, SCC-k (Braoudakis, 1995).

Finalement, on peut considérer le protocole SCC comme un précurseur pour les algorithmes d'ordonnancement des transactions dans les SGBD temps réel. Les applications temps réel étant souvent des applications critiques, l'inconvénient majeur de ces algorithmes, représenté par la nécessité de dupliquer les ressources, est relativement atténué.

3.2.4 Protocole multi-versions (MCC)

Dans ce protocole (Bernstein and Goodman, 1983), les écritures ne sont pas considérées comme des écrasements de valeurs (comme c'est le cas habituellement), mais comme la création de nouvelles versions d'objets. Les estampilles sont utilisées pour renvoyer la version adéquate de l'objet suite à une requête de lecture. Il n'y a donc pas de surcoût de contrôle de concurrence, puisque les différentes versions peuvent être nécessaires à l'algorithme de reprise (Bernstein et al., 1987).

Dans (Kim and Srivastava, 1991), les auteurs ont adapté le protocole MCC (Multiversion Concurrency Control) au contexte temps réel pour réduire le taux de transactions abandonnées. Cependant, cette technique ne peut être appliquée que dans les systèmes où les écritures des données correspondent à la création de nouvelles versions. Si tel est le cas, les transactions qui écrivent les valeurs issues des capteurs par exemple ne rentrent pas en conflit avec celles qui lisent ces valeurs. L'exploitation d'anciennes versions constitue l'obstacle majeur pour utiliser cette technique dans beaucoup d'environnements temps réel où la nécessité d'utiliser des informations fraîches[4] est un critère important.

3.2.5 Protocole hybride des intervalles d'estampilles

Ce protocole proposé dans (Son et al., 1992) combine les techniques optimistes et l'ordre des estampilles. Il améliore les algorithmes optimistes où se posent des problèmes de gestion efficace des ressources, dus à l'abandon des transactions durant leur phase de validation. De plus, cet abandon est souvent inutile car une transaction passe un test de certification, avant sa phase de validation, pour vérifier si elle fait partie d'un ordre d'exécution sérialisable. Si ce n'est pas le cas, elle est abandonnée. Or, ce test se base sur les ensembles de lecture et d'écriture de la transaction, ensembles qui sont déterminés a priori et non sur l'ordre réel d'exécution. Cela conduit parfois à conclure de façon erronée qu'une exécution est non sérialisable.

Le problème d'utilisation inefficace des ressources est résolu partiellement en utilisant la certification *en avant* qui consiste à détecter au plus tôt les conflits (le test de certification est effectué durant la

[4] Durant leurs intervalles de validit´e.

phase de lecture (Haritsa et al., 1990)). Ainsi, si un conflit est détecté lors de la phase de lecture d'une transaction, celle-ci ou les transactions avec lesquelles elle est en conflit peuvent être abandonnées, procurant à l'algorithme assez de flexibilité pour pouvoir être utilisé avec des mécanismes d'affectation de priorités.

En outre, le protocole hybride des intervalles d'estampilles utilise l'allocation dynamique d'estampilles (Bayer et al., 1982), qui consiste à construire à la demande l'ordre de sérialisation à chaque fois que des conflits réels surviennent dans le système. Seul un ordre partiel nécessaire pour les transactions est construit (au lieu d'un ordre total construit par le mécanisme d'allocation statique des estampilles). Cette allocation dynamique des estampilles est rendue plus efficace par l'utilisation de la notion d'intervalle d'estampilles (Boksenbaum et al., 1987). À chaque transaction est affecté un intervalle d'estampilles (correspondant initialement à tout l'espace des estampilles). Ensuite, à chaque lecture ou écriture d'un objet par la transaction, son intervalle d'estampilles est ajusté pour préserver l'ordre de sérialisation induit par la validation d'autres transactions.

Basant leur jugement sur une analyse qualitative, les auteurs affirment que les algorithmes hybrides comme celui présenté, offrent des performances meilleures que ceux fondés sur un protocole unique. Cependant, comme des études quantitatives n'ont pas été effectuées, il n'est pas possible de déterminer le coût effectif de ces algorithmes.

3.3 Notre contribution

Depuis que nous avons commencé à nous intéresser au domaine des SGBDTR, nous avons axé nos travaux sur la gestion des transactions temps réel, notamment sur la conception et l'évaluation de protocoles de contrôle de concurrence et d'ordonnancement efficaces. Nous avons alors abordé plusieurs points de vue, dont certains consistent à améliorer les performances des protocoles déjà proposés, c'est-à-dire de faire en sorte que le ratio de succès[5] soit amélioré, et d'autres ont consisté à proposer de nouveaux protocoles. Dans la suite, nous allons discuter de ces travaux.

3.3.1 Protocole basé sur l'imprécision

Nous avons proposé, dans (Sadeg and Bouzefrane, 1999; Bouzefrane and Sadeg, 2000b), une classification des applications temps réel en nous fondant sur les travaux de (Pu, 1991). Les critères de classification sont le degré d'importance donnée à l'échéance d'une transaction, et/ou le degré d'importance affectée à la précision/complétude du résultat. Des protocoles de contrôle de CC&O des transactions ont ensuite été proposés, en tenant compte des différents critères.

3.3.1.1 Notions de ϵ-donnée et de Δ-échéance

Notre travail a initialement été inspiré par les travaux de Pu et al. (Pu and Leff, 1992; Pu, 1993) concernant les epsilon-transactions, qui sont des transactions dont le critère de correction est *l'epsilon-sérialisabilité*. Cette notion permet aux transactions d'importer une quantité bornée d'incohérence (transactions d'interrogation), et/ou d'exporter une quantité bornée d'incohérence (transactions de mise à jour).

À partir des résultats obtenus, nous avions émis l'idée de considérer des transactions pouvant manipuler des données et résultats imprécis (ϵ-$donnee$) et pouvant avoir deux types d'échéances : échéances initiales et échéances étendues (Δ-échéance).

[5] NombreDeTransactionsAyantRespect´eLesEch´eances / NombreTotalDeTransactions.

Nous sommes partis de la constatation que dans beaucoup d'applications, il est préférable que les résultats des transactions, notamment d'interrogation, soient complets, exacts et obtenus avant une échéance donnée. Cependant, si pour certaines raisons, ceci est impossible, il est toléré que ces transactions puissent dépasser leur échéance d'une quantité de temps borné Δ (au delà de laquelle la transaction est abandonnée), et/ou de se contenter de résultats partiels/imprécis.

Des protocoles de CC&O qui se basent sur cette idée ont été proposés (Bouzefrane and Sadeg, 2000b). Des simulations ont montré que les protocoles proposés améliorent les performances des SGBDTR, par rapport aux protocoles temps réel déjà proposés. Dans la suite de cette section, nous donnerons les grandes lignes de ces protocoles de CC&O basés sur ces notions, ainsi que quelques éléments sur ces deux notions et leur utilisation.

3.3.1.2 Contexte

L'objectif pour les SGBDTR de devoir respecter à la fois les contraintes logiques et les contraintes temporelles de la base de données reste difficile à atteindre. Plusieurs travaux ont été proposés qui apportent de nouveaux concepts pour relaxer certaines propriétés $ACID$, caractéristiques des transactions, notamment la sérialisabilité. En effet, dans certaines applications temps réel, la stricte sérialisabilité des transactions n'est pas souhaitable si son non-respect pourrait améliorer le respect des échéances. Les travaux décrits ici concernent ces applications temps réel où certaines imprécisions contrôlées sur les résultats et/ou quelques dépassements bornés d'échéances des transactions sont tolérés.

3.3.1.3 Imprécision des données/résultats et dépassements d'échéances

Les transactions dans un SGBD doivent posséder les propriétés $ACID$. Mais ces propriétés sont jugées trop strictes dans un contexte temps réel (Ramamritham, 1993). Des recherches ont été entreprises et des notions sont proposées qui tiennent compte de la sémantique des données et qui apportent une certaine flexibilité dans le respect de ces propriétés. Nous avons proposé deux notions, ϵ-donnée et Δ-échéance, qui, non seulement permettent de relaxer la sérialisabilité des transactions, mais permettent également de relaxer et de contrôler les échéances des transactions.

Ces notions de ϵ-donnée et Δ-échéance permettent de modéliser, par exemple, les applications qui établissent un compromis entre la ponctualité des résultats et leur précision, moyennant le contrôle, et de l'imprécision des données/résultats, et des dépassements d'échéances.
Une application potentielle de ces notions est le domaine du multimédia sur lequel beaucoup de travaux ont été effectués (Wang et al., 1997; Bertino et al., 2003; Wu et al., 2001). Nous présenterons dans le paragraphe 3.3.2 l'adaptation de ces concepts dans une application multimédia. Une autre application de ces concepts aux SGBD mobiles est présentée dans le paragraphe 3.3.3.

Selon les valeurs des paramètres ϵ et Δ, nous avons établi la classification suivante des applications temps réel :

1. Applications ϵ-Δ : ce sont des applications qui peuvent tolérer l'obtention de résultats imprécis/incomplets, qui arriveraient légèrement en retard, e.g. applications multimédia.
2. Applications ϵ-0 : ce sont des applications qui peuvent tolérer l'obtention de résultats imprécis/incomplets, mais qui doivent arriver avant échéance (dans les temps), e.g. applications de prise de décision.
3. Applications 0-Δ : ce sont des applications dont les résultats doivent être précis/complets, même s'ils sont obtenus légèrement en retard, e.g. applications bancaires, réservations de places d'avion ou de train, etc.

4. Applications 0-0 : ce sont des applications dont les résultats doivent être non seulement précis et complets mais aussi obtenus dans les temps, e.g. temps réel strict, critique.

Nous présentons dans la suite des protocoles de contrôle de concurrence et ordonnancement des transactions temps réel utilisant ces notions. Auparavant, nous donnons les notations utilisées :

$T^{\epsilon-\Delta}$: est une transaction ayant une Δ-$echeance$ et manipulant une ϵ-$donnee$. T peut être une transaction d'interrogation ($Q_{\epsilon,\Delta}$), ou la partie lecture ($R^{\epsilon,\Delta}$) ou écriture ($W^{\epsilon,\Delta}$) d'une transaction de mise à jour.

W$^{\epsilon-\Delta}$: est la partie *écriture* d'une transaction de mise à jour U$^{\epsilon-\Delta}$.

R$^{\epsilon-\Delta}$: est la partie *lecture* d'une transaction de mise à jour U$^{\epsilon-\Delta}$.

Q$^{\epsilon-\Delta}$: est une transaction d'interrogation qui autorise la récupération de résultats exploitables, mais incomplets (ou imprécis).

v_to_w : une valeur à écrire par la transaction W$^{\epsilon-\Delta}$, en conflit avec les transactions Q$^{\epsilon-\Delta}$.

v_to_r : une valeur à lire par la transaction Q$^{\epsilon-\Delta}$, en conflit avec les transactions W$^{\epsilon-\Delta}$.

ϵ(d) : quantité d'imprécision tolérée par la donnée d, ayant pour valeur exacte v, c-à-d qu'une valeur de v dans l'intervalle [v-ϵ, v+ϵ] est acceptable.

ϵ_w(d) : est la valeur absolue de la différence entre la valeur courante de d et sa valeur après modification par W$^{\epsilon-\Delta}$.

$\Delta(T_i)$: dénote la quantité de temps avec laquelle la transaction T_i est autorisée à dépasser son échéance.

d (T_i) : est l'échéance initiale de la transaction T_i.

commit_inst(T_i) : dénote l'instant de Commit de la transaction T_i.

3.3.1.4 Contrôle de concurrence et ordonnancement

De nouveaux verrous sont utilisés dans le contrôle de concurrence des ϵ-Δ-transactions, et la nouvelle matrice de compatibilité des verrous est élaborée. Elle est illustrée par la figure 3.1 où :

- les colonnes correspondent aux transactions détentrices de verrous et les lignes aux transactions demandeuses des verrous.

- les cases X correspondent à un cas d'incompatibilité de verrous
- les cases OK correspondent à un cas de compatibilité de verrous
- la case V_1 correspond à une compatibilité conditionnelle : cas où une transaction d'écriture détient un verrou sur un objet, et une transaction d'interrogation sollicite ce verrou. Cette dernière peut acquérir le verrou à condition que l'imprécision qu'elle importe soit inférieure à ϵ de l'objet.

– la case V_2 correspond à une compatibilité conditionnelle : cas où une transaction d'interrogation détient un verrou sur un objet, et une transaction d'écriture sollicite ce verrou. Cette dernière peut acquérir le verrou à condition que l'imprécision qu'elle engendre en accédant à l'objet soit inférieure à ϵ de l'objet.

3.3.1.5 Description du protocole

Le protocole proposé se base sur un gestionnaire de transactions qui contrôle l'exécution des transactions selon la figure 3.1. Contrairement aux gestionnaires de transactions traditionnels, celui-ci permet à deux transactions d'accéder simultanément à une même donnée avec des accès, a priori, incompatibles. Le gestionnaire intègre pour cela un mécanisme qui contrôle la quantité d'incohérence introduite par ces accès, ainsi que le dépassement éventuel de l'échéance des transactions. Le contrôleur de concurrence utilise le protocole $\epsilon - \Delta$-2PL-HP, une version étendue du protocole 2PL-HP (Abbot and Garcia-Molina, 1988). Le gestionnaire de transaction coopère avec l'ordonnanceur du système d'exploitation pour éliminer les transactions inutiles, i.e. ayant manqué leurs échéances. Intuitivement, on peut penser que ce protocole offre de meilleures performances que les protocoles déjà proposés dans le même but puisqu'il contient deux cas supplémentaires (V_1 et V_2 dans la figure 3.1.) qui permettent à des transactions en conflit de s'exécuter en concurrence. D'autre part, nous travaillons dans un environnement mono-processeur avec une base de données en mémoire centrale. Les transactions sont périodiques et sont ordonnancées selon l'algorithme EDF. Le principe du protocole ϵ-Δ-2PL-HP est le suivant.

Quand un conflit d'accès du type V_1 ou V_2 sur une donnée survient, les transactions en conflit T_i et T_j sont autorisées à poursuivre leur exécution à condition que la quantité d'incohérence qu'elles introduisent dans la base soit bornée par le paramètre ϵ de la donnée d'une part, et que les échéances des deux transactions ne soient pas dépassées, ou dépassées seulement d'une durée inférieure au minimum des Δ (c-à-d $\min(\Delta_i, \Delta_j)$) de chaque transaction.

Dans la suite, nous explicitons le cas où un conflit de type V_1 de la figure 3.1 se produit. La description du protocole relatif au cas V_2 peut être trouvé dans (Bouzefrane and Sadeg, 2000b).

	$Q^{\epsilon,\Delta}$	$R^{\epsilon,\Delta}$	$W^{\epsilon,\Delta}$
$Q^{\epsilon,\Delta}$	OK	OK	V_1
$R^{\epsilon,\Delta}$	OK	X	X
$W^{\epsilon,\Delta}$	V_2	X	X

FIG. 3.1 – Matrice de compatibilité des verrous

Nous décrivons les différentes possibilités induites par V_1, selon que la transaction tolère ou non l'utilisation de données imprécises ($\epsilon > 0$ ou $\epsilon = 0$) et/ou que son échéance initiale peut ou non être étendue ($\Delta > 0$ ou $\Delta = 0$). Ces possibilités sont résumées dans l'algorithme général suivant, où le gestionnaire de transactions exécute les étapes suivantes :

Debut
-- Quand un conflit V_1 survient : une donnée d verrouillée
-- en écriture par une transaction $W^{\epsilon-\Delta}$ et une transaction
-- d'interrogation $Q^{\epsilon-\Delta}$ souhaite verrouiller en lecture d.
Si ϵ(d) $>$ 0 et v_to_w(d) $\in [v\_to\_r(d) - \epsilon(d), v\_to\_r(d) + \epsilon(d)]$
Alors $Q^{\epsilon-\Delta}$ poursuit son exécution en concurrence avec $W^{\epsilon-\Delta}$
Sinon Si d($W^{\epsilon-\Delta}$) + Δ($W^{\epsilon-\Delta}$) < d($Q^{\epsilon-\Delta}$)) + Δ($Q^{\epsilon-\Delta}$)
    Alors $Q^{\epsilon-\Delta}$ est bloquée
    Sinon $W^{\epsilon-\Delta}$ est abandonnée
    Finsi
Finsi
Appel EDF-scheduler(transactions)
Fin

Quand $W^{\epsilon-\Delta}$ effectue son Commit ou est abandonnée, le gestionnaire de transactions exécute l'algorithme suivant :

Debut
Si $W^{\epsilon-\Delta}$ Commit
Alors $\forall$ $Q_i^{\epsilon-\Delta}$ une transaction en attente,
    elle est réveillée (mise dans la file des transactions *prêtes*)
    Appel EDF-scheduler (transactions)
Finsi
Si $W^{\epsilon-\Delta}$ manque son échéance
    (d($W^{\epsilon-\Delta}$) + Δ($W^{\epsilon-\Delta}$) avec Δ($W^{\epsilon-\Delta}$)> 0)
Alors Le gestionnaire de transactions :
    Abandonne $W^{\epsilon-\Delta}$
    Réveille les transactions $Q^{\epsilon-\Delta}$ en attente;
    Appel EDF-scheduler (transactions)
Finsi
Fin

Nous pouvons constater d'après les algorithmes qui précédent que dans le cas V_1, notre protocole : (1) ne bloque pas les transactions d'interrogation si l'imprécision qu'elles importent est bornée, (2) bloque toutes les transactions si la valeur écrite par la transaction détenant le verrou est telle qu'elle engendre une erreur supérieure à l'imprécision tolérée. Une fois que la transaction d'écriture en cours a effectué son Commit ou est abandonnée, les transactions d'interrogation en attente sont réveillées pour s'exécuter. Elles ont ainsi davantage de chances de se terminer avant leurs échéances, car elles ont des échéances étendues.

Pour illustrer les conditions V_1 et V_2, nous prenons un exemple :

Soit d=15 une donnée élémentaire dans la base de données. ϵ(d)=0.3 (c'est-à-dire 30% de la valeur réelle de d). L'exemple de la figure 3.2. illustre deux situations : un conflit de type V_2 qui se produit quand une transaction d'interrogation T_{Q1} verrouille en lecture d, et que pendant son exécution, trois transactions d'écriture T_{W1}, T_{W2} et T_{W3} avec des priorités plus élevées préemptent T_{Q1} et demandent à verrouiller en écriture d (valeurs écrites : 16, 18 et 20 respectivement).

Un conflit du type V_1 se produit quand la transaction de mise à jour T_{W3} verrouille d en écriture, et pendant son exécution deux transactions d'interrogation T_{Q2} et T_{Q3} avec des priorités plus élevées préemptent T_{W3} et demandent à verrouiller en lecture d.

Les échéances initiales des transactions sont fixées respectivement à : d(T_{Q1}) = 6, d(T_{W1}) = 6, d(T_{W2}) = 7, d(T_{W3}) = 11, d(T_{Q2}) = d(T_{Q3}) = 12, et soit Δ égal à 1 pour toutes les transactions, c'est-à-dire, chaque transaction est autorisée à poursuivre son exécution au delà de son échéance initiale d'une unité de temps (en réalité, pour chaque transaction, il vaudrait mieux initialiser Δ à un pourcentage de sa durée d'exécution).

Une variable Sum_Eps qui accumule les fluctuations ϵ-W_i est initialisée à 0 pour la donnée élémentaire d. En appliquant les protocole que nous proposons, nous obtenons :

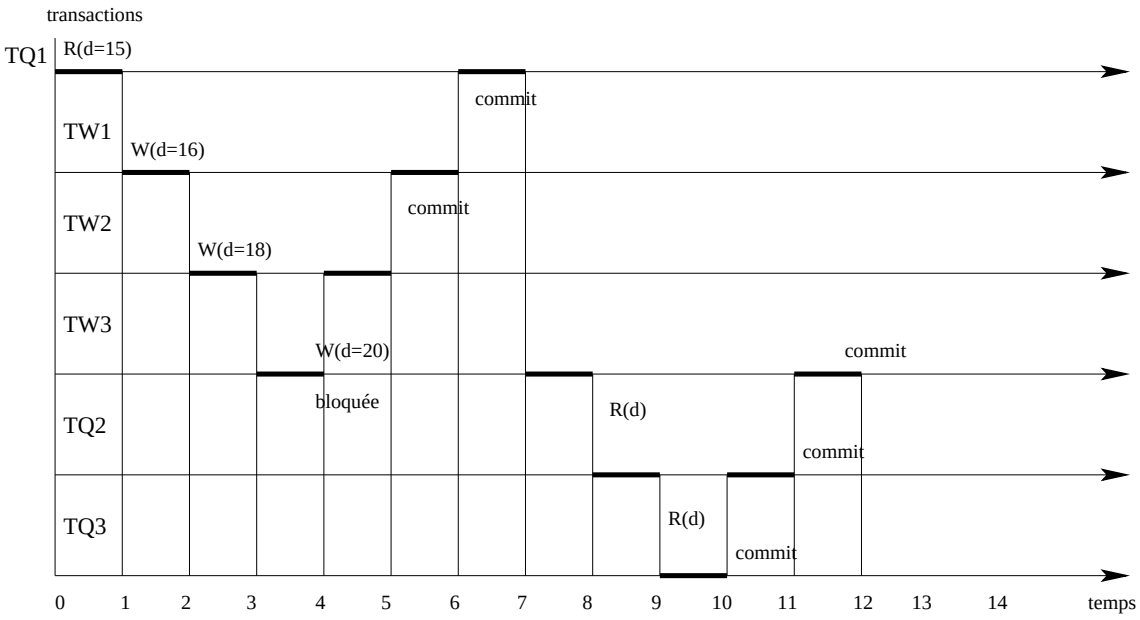

FIG. 3.2 – Illustration des conflits V_1 et V_2

– à l'instant 0, une transaction d'interrogation T_{Q1} commence à s'exécuter. Pendant son exécution, la transaction d'interrogation T_{Q1} demande à verrouiller en lecture d. Elle peut le faire car aucune transaction n'accède à d. Par conséquent, T_{Q1} lit la valeur 15,

– à l'instant 1, une transaction T_{W1} de mise à jour avec une priorité plus élevée que celle de T_{Q1} préempte T_{Q1} et commence à s'exécuter. Pendant ce temps, T_{W1} demande à verrouiller en écriture d. Un conflit de type V_2 se produit. Alors le GT calcule la fluctuation ϵ-T_{W1}(d) apportée par T_{W1} (elle est égale à (16-15)/15=0.06 = 6%) et il contrôle si Sum_Eps + 6% = 0.06 est $<$ ϵ(d). Comme $0.06 < 0.3$, on permet à T_{W1} de verrouiller en écriture d et de reprendre son exécution,

– à l'instant 2, une autre transaction de mise à jour T_{W2}, avec une priorité plus élevée que celle de T_{W1} préempte T_{W1}. Pendant son exécution, T_{W2} demande à verrouiller en écriture d. Le principe est de cumuler les fluctuations apportées par chaque transaction en écriture (en mise à jour) tant que la transaction de lecture n'a pas encore effectué sa validation (ou été interrompue). T_{W2} est autorisée à verrouiller en écriture d car : Sum_Eps + ϵ-T_{W2}(d) = 0.06+20% $<$ 0.30 (où 20% est la valeur de ϵ-T_{W2}(d) = (18-15)/15),

– à l'instant 3, une autre transaction de mise à jour T_{W3} avec une priorité plus élevée que celle de T_{W2} préempte T_{W2} et commence à s'exécuter,

– à l'instant 4, T_{W3} demande à verrouiller en écriture la même donnée élémentaire d. Le GT refuse sa demande car sum_eps + ϵ-T_{W3}(d) = 0.26 + 33% excède 0.30 (où 33% est la valeur de ϵ-T_{W3}(d) = (20-15)/15). T_{W3} est alors bloquée et T_{W2} reprend son exécution puisque c'est la transaction de plus haute priorité parmi celles qui sont préemptées,

– à l'instant 5, T_{W2} effectue sa validation avant son échéance initiale et la nouvelle valeur de d devient 18. T_{W1} reprend alors son exécution,

– à l'instant 6, T_{W1} atteint son échéance initiale. À ce moment, T_{Q1} (de même échéance initiale que T_{W1}) pourrait être abandonnée. Mais notre ordonnanceur EDF étendu reconsidère T_{Q1} avec sa Δ-$echeance$, qui est égale à 6+1=7. T_{Q1} reprend alors son exécution et effectue sa validation à l'instant 7,

– à l'instant 7, dès que T_{Q1} est validée, le GT libère T_{W3} qui obtient le verrou en écriture sur d (puisque aucune transaction n'accède à d),

– à l'instant 8, une transaction T_{Q2} de priorité plus élevée que celle de T_{W3} préempte T_{W3} et commence à s'exécuter. Pendant ce temps, elle demande à verrouiller en lecture d. Le GT teste la condition requise par V_1. Si elle est vraie alors il permet à T_{Q2} de verrouiller en lecture d. Ici, l'évaluation de la condition donne le résultat suivant :

$$\frac{\|v\_to\_w\| - \|v\_to\_r\|}{\|v\_to\_r\|} = \frac{\|16 - 20\|}{20} = 20\% < \epsilon(d) = 30\%$$

Donc, la transaction T_{Q2} est autorisée à verrouiller en lecture d,

– à l'instant 9, une transaction T_{Q3} de priorité plus élevée que T_{Q2} préempte T_{Q2}. Pendant ce temps, T_{Q3} demande à verrouiller en lecture d. Elle peut le faire puisque :

$$\frac{\|v\_to\_w\| - \|v\_to\_r\|}{\|v\_to\_r\|} = \frac{\|16 - 20\|}{20} = 20\% < \epsilon(d) = 30\%$$

Notons que, quand la condition induite par V_1 est évaluée à VRAI, toutes les transactions d'interrogation T_{Qi} qui arrivent avant la validation de la mise à jour de T_W sont autorisées à s'exécuter en même temps que T_W (car ϵ-T_W(d) n'est pas modifié),

– à l'instant 10, T_{Q3} est validée avant son échéance initiale et T_{Q2} reprend son exécution,

– à l'instant 11, T_{Q2} est validée et T_{W3} est activée. T_{W3} pourrait être interrompue puisqu'elle a atteint son échéance initiale avant la fin de son exécution. Mais comme notre protocole d'ordonnancement reconsidère T_{W3} avec son échéance étendue, T_{W3} reprend son exécution et est validée à sa Δ-$echeance$ (instant 12).

Nous avons démontré dans (Bouzefrane and Sadeg, 2000b) que ce protocole permet à davantage de transactions temps réel de respecter leurs échéances, à condition que l'application concernée autorise des imprécisions bornées de résultats et des dépassements limités d'échéances.

3.3.1.6 Conclusion

Nous avons proposé une classification des applications temps réel qui tient compte, d'une part de la tolérance ou non des applications à manipuler des données ou résultats incomplets/imprécis, et d'autre part de la possibilité que des résultats puissent arriver ou ne doivent pas arriver en retard.

Les notions de ϵ-donnée et Δ-échéance ont été introduites pour faciliter la mise en oeuvre de cette classification. Enfin, des protocoles de contrôle de concurrence et ordonnancement des transactions temps réel ont été proposés et la démonstration de leurs correction a été donnée.

Nous avons appliqué ces notions dans le domaine du multimédia et une brève description de ces travaux est donnée dans le paragraphe 3.3.2. Les détails peuvent être trouvés dans (Sadeg and Bouzefrane, 2000a). L'application de ces notions à un environnement mobile et sans fil pour pallier les déconnexions fréquentes dans les SGBDTR a également été effectuée. Notre contribution est décrite dans le paragraphe 3.3.3. Les détails peuvent être trouvés dans (Sadeg et al., 2001b).

3.3.2 Application au domaine du multimédia

3.3.2.1 Flux multimédia

Les applications multimédia (MM) telles que la visio-conférence, la vidéo-à-la-demande imposent des contraintes à tous les composants du système (réseau, applicatifs, interfaces, ...). Les besoins des applications MM vont des débits élevés (grandes quantités de données) aux contraintes temporelles strictes (dues à la nature continue des flux audio et vidéo). Parmi ces contraintes temporelles, certaines résultent des conditions de synchronisation : synchronisation intra-flux : le temps d'attente impose une borne supérieure sur le retard toléré. Les données qui excèdent cette limite deviennent inutiles et sont ignorées. La synchronisation inter-flux traite des liens temporels qui peuvent exister entre les différents flux (lip-synchronisation)

3.3.2.2 Exemple d'application multimédia

Par exemple, dans une application de visio-conférence, l'information est créée en temps réel depuis une caméra, et devrait être consommée en temps réel, e.g. par un moniteur. Les données produites sont valables pendant une période, et une fois cette durée écoulée, les données deviennent inutiles.

Cependant, plusieurs de ces applications peuvent tolérer certaines imprécisions bornées. En effet, on peut parler de ϵ-$donnee$ lorsqu'un transaction acquiert une image incomplète (perte de pixels due à la transmission à travers le réseau par exemple). De plus, si à la réception du son et de l'image un certain décalage se produit, dû aux dépassements d'échéances de certaines transactions de transport de ces flux, et si on peut contrôler ce décalage, c'est-à-dire les Δ-$echeance$, la scène sera encore exploitable. Dans ces applications, les notions d'ϵ-$donnee$ et de Δ-$echeance$ pourraient donc être très utiles, puisqu'elles évitent d'abandonner certaines transactions et améliorent ainsi les performances du système.

Les SGBD sur lesquels reposent ces applications doivent être conçus différemment des SGBD traditionnels, qui ont pour objectif l'amélioration de l'exécution concurrente des transactions, tout en assurant la cohérence logique de la base de données. Pour cela, les transactions doivent posséder les propriétés ACID. Dans les SGBD multimédia temps réel, ces propriétés, notamment la propriété d'isolation (qui implique la sérialisabilité), pourraient être relaxées afin de tenir compte des données imprécises et/ou de permettre à des transactions de s'exécuter au delà de leurs échéances.

Nous avons introduit les notions d'ϵ-$donnee$ et de Δ-$echeance$, et nous nous basons sur la matrice de compatibilité de la figure 3.1. pour contrôler l'exécution concurrente des transactions. Cela nous conduit à utiliser les algorithmes d'ordonnancement présentés dans le paragraphe 3.3.1 pour effectuer des traitements sur des données imprécises et pour gérer les dépassements éventuels des échéances des transactions.

3.3.2.3 ϵ-$donnee$ et Δ-$echeance$ dans une application multimédia

Une application multimédia de visio-conférence peut consister par exemple en un travail sur un document par plusieurs utilisateurs (conférenciers) qui agissent en temps réel sur les éléments du document. Le document pouvant être composé de texte, d'images fixes ou animées, de graphiques, etc. Les différents participants agissent sur le document commun simultanément chacun de son côté sachant que les modifications effectuées pas les uns doivent être vues par les autres pour que chaque utilisateur puisse voir un contenu cohérent du document.

Cette situation pourrait être gérée par un SGBD temps réel où sont implantés les concepts d'ϵ-$donnee$ et de Δ-$echeance$. Les éléments à prendre en compte pourraient être : la présentation des images à la cadence (ou période) de 25 images par seconde, période requise pour que l'utilisateur perçoive une image continue nette. Donc une image est traitée en 40 millisecondes. Si le système prend un peu plus de temps pour traiter une image, 50 millisecondes par exemple (échéance étendue des transactions de traitement de l'image,) cela ne devrait pas nuire à la qualité de l'image, si le dépassement d'échéance est borné par une quantité Δ (ici on a donc $\Delta = \|40 - 50\|/40 = 25\%$ de 40 millisecondes). Ce qui reviendrait à recevoir 20 images par secondes, ce qui représente la limite minimale pour recevoir une image relativement correcte. Ceci illustre la notion de Δ-$echeance$. Une image étant un ensemble d'octets, la transmission d'une image avec une perte de quelques octets (10% par exemple) ne devrait pas nuire à la qualité de l'image. Ceci illustre la notion d'ϵ-$donnee$.

3.3.3 Application dans un environnement mobile

3.3.3.1 Contexte

Depuis quelques années, des avancées rapides ont été réalisés dans la technologie des communications mobiles et sans fils. Un réseau sans fils peut être vu comme une collection de noeuds mobiles qui forment un réseau temporaire. Par exemple, dans une application de commerce électronique, des acteurs du système disposent de média (ordinateurs portables, ...), appelés par la suite `MH`[6], connectés à des sites fixes via un réseau sans fils. Dans un tel système, de grandes quantités de données sont manipulées et les transactions soumises par les média mobiles doivent recevoir leurs résultats avant des échéances (fixées), pour éviter des conséquences économiques néfastes.

Prenons l'exemple d'une application de marché : elle est composée d'une base de données principale localisée sur un site fixe, où sont stockées toutes les caractéristiques des produits, et de plusieurs mobiles (clients) disposant chacun d'une partie de la base de données, concernat les caractéristiques des produits dont ils disposent localement. Il devra être possible pour un client (acheteur/vendeur) d'accéder à la base principale pour exécuter des transactions. Chaque client est responsable de la mise à jour *dans les temps* de sa base de donnée (locale et globale).

Un des problèmes posés dans ces systèmes est la déconnexion fréquente qui peut survenir entre les `MH` et le site fixe. Afin d'assurer la ponctualité et la continuité des traitements des utilisateurs, il est nécessaire de gérer le mieux possible ces déconnexions.

Dans ce travail, nous avons exploité les notions de ϵ et Δ définies précédemment pour définir un protocole de contrôle de concurrence et ordonnancement qui assure la ponctualité des résultats et minimise l'effet des déconnexions pour les transactions d'interrogation.

[6] Mobile Host.

3.3.3.2 Modèle de SGBD mobile

Nous considérons un modèle de SGBD mobile comme celui décrit dans (Badrinath and Sudame, 1996) et présenté dans la figure 3.3. Chaque MH couvre une zone géographique et peut se déplacer dans d'autres zones. Cependant, à un instant donné, il ne peut communiquer qu'avec une seule *station de base* (un site fixe).

Nous supposons, sans perte de généralité, que nous disposons d'une seule station à laquelle sont connectés plusieurs MH par un réseau sans fils. Nous considérons une base de données principale localisée sur la station de base et des copies partielles de cette base localisées sur les MH.

Nous supposons également que les applications qui tournent sur ce système peuvent tolérer l'obtention de résultats incomplets (imprécis) obtenus dans les temps, et que les transactions peuvent dépasser leurs échéances initiales d'une quantité de temps bornée, fixée a priori.

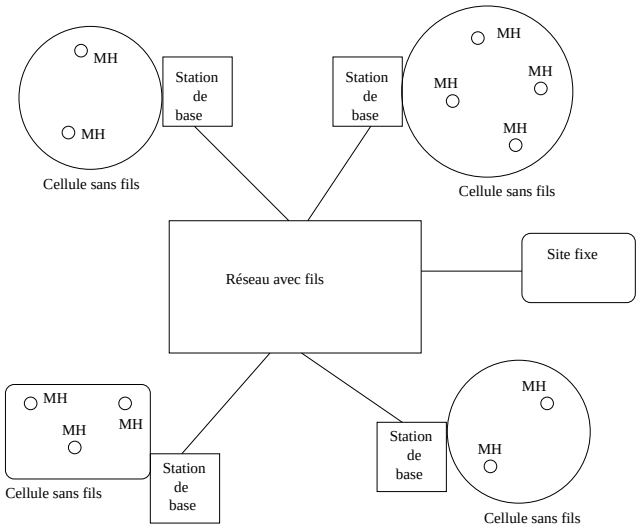

FIG. 3.3 – Illustration du modèle

3.3.3.3 Brève description du protocole

Dans le but de circonscrire ou d'atténuer l'effet des déconnexions entre les MH et le site fixe, nous proposons le protocole DT[7], pour gérer les transactions d'interrogation de type *Soft* ayant certaines caractéristiques, i.e. chaque transaction a une échéance initiale et une échéance étendue (Sadeg and Bouzefrane, 2000a). Selon les applications, il peut être préférable d'obtenir dans les temps des résultats entachés d'imprécision/d'incomplétude, plutôt que des résultats précis/complets en retard.

L'idée de base du protocole est de déterminer au niveau de la station de base qui reçoit une transaction d'interrogation s'il est préférable d'envoyer au MH demandeur un résultat incomplet (un sous ensemble des tuples d'une table, par exemple), ou de patienter un certain temps pour envoyer le résultat complet, basé sur une estimation des instants de déconnexion et de connexion du réseau sans fils.

Le protocole DT proposé comporte deux parties : une qui s'exécute sur la station de base, une autre qui s'exécute sur le MH concerné. Les détails des algorithmes peuvent être trouvés dans (Bouze-

[7] Disconnexion Tolerance

frane et al., 2001).

Notre objectif est que des résultats soient transmis dans les temps. Pour cela, nous déterminons un paramètre, appelé *temps de réponse limite*, qui permet au MH d'envoyer ou non sa requête à la station de base. De même la station de base détermine s'il est opportun d'envoyer le résultat (même incomplet) immédiatement, ou s'il est préférable d'attendre.

Nous avons effectué des simulations dans (Bouzefrane et al., 2001) qui ont montré que même si les déconnexions sont fréquentes, le temps de réponse dépend seulement de la durée des déconnexions, qui est donc un problème majeur dans les environnements sans fils. Le protocole DT présenté permet de déterminer, moyennant certaines hypothèses, ce temps de réponse limite, qui est ensuite utilisé pour assurer que davantage de transactions respectent leurs échéances.

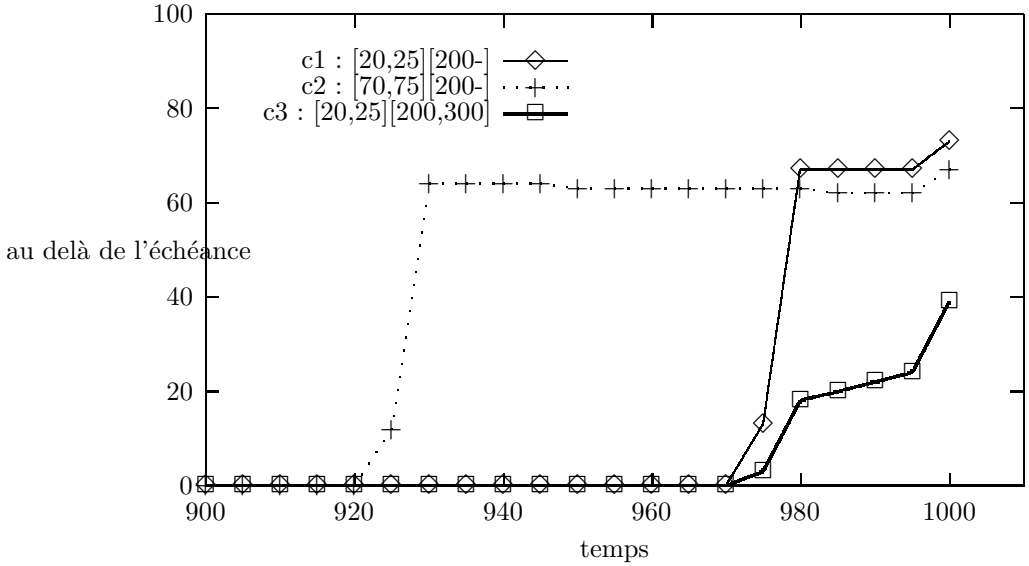

FIG. 3.4 – Échéance étendue et temps de réponse limite (1)

A titre d'exemple, nous illustrons dans les figures 3.4 et 3.5 les instants où les transactions sont autorisées à transmettre leurs requêtes ; les intervalles de valeurs représentant respectivement les durées des déconnexions et des connexions.

On note par exemple sur les courbes c_1 et c_2 que plus de 60% des transactions poursuivent leur exécution jusqu'à leurs échéances étendues (sans déconnexions). Les courbes c_3 et c_4 (Figure 3.5) indiquent que, quand les durées des connexions sont inférieures aux durées des déconnexions, alors seulement 30% des transactions peuvent respecter leurs échéances, après le paramètre *temps de réponse limite*. Dans ce cas, la notion de ϵ-$donnee$ prend toute son importance, car plus de 60% des résultats incomplets obtenus avant le *temps de réponse limite* ne peuvent plus être améliorés.

3.3.4 Protocole spéculatif amélioré

3.3.4.1 Problématique

Après une étude détaillée du protocole SCC (cf. paragraphe 3.2.3), nous avons jugé que son fonctionnement est intéressant pour le contexte temps réel. Cependant, le protocole gagnerait à être amélioré sur certains points. Nous avons donc cherché les points où ce protocole paraît moins performant et nous

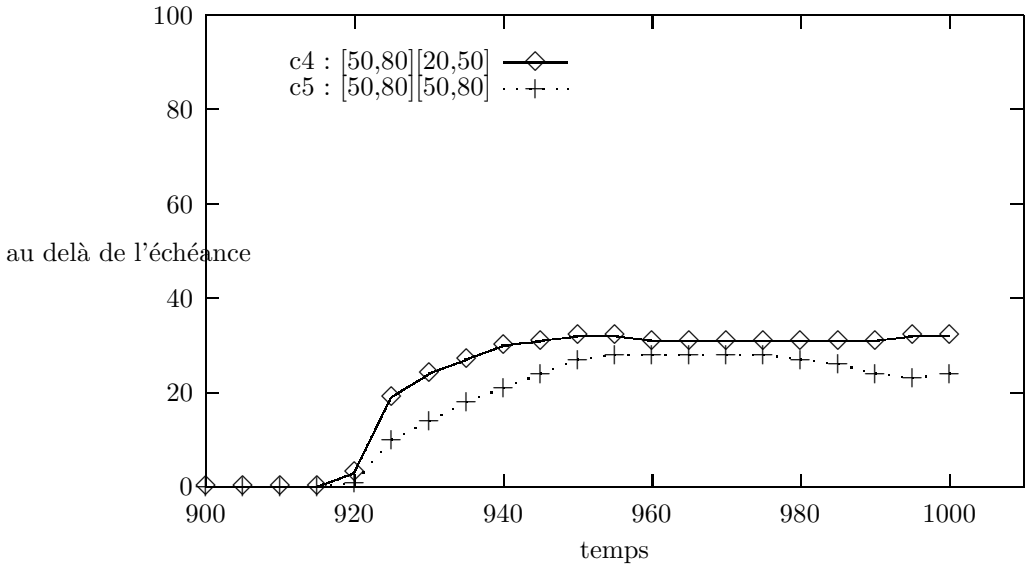

FIG. 3.5 – Échéance étendue et temps de réponse limite (2)

avons travaillé dessus. Le résultat est la proposition du protocole ESCC qui pallie les problèmes rencontrés dans SCC de base.

Des extensions au protocole SCC existent, notamment celles proposées par les auteurs (Bestavros et al., 1993). Mais ces extensions concernent seulement les moyens de diminuer le nombre de transactions *fantômes* à créer. Dans notre travail, nous proposons une nouvelle extension du protocole SCC qui permet d'améliorer les performances de SCC en agissant sur le type de conflit. Nous agissons, en particulier, sur deux éléments : (1) Nous nous sommes intéressés à la résolution des conflits de type W-W (write-write) qui sont résolus dans SCC de base par la méthode TWR (Thomas Write Rule), et (2) Nous proposons une amélioration pour la prise en compte des échéances des transactions pour résoudre les autres types de conflits : R-W (read-write) et W-R (write-read).

L'intérêt principal du protocole SCC est qu'il combine les avantages des méthodes pessimistes et optimistes tout en évitant leurs principaux inconvénients. Les conflits sont en effet détectés dès leur apparition comme pour les méthodes pessimistes, mais les transactions continuent à s'exécuter en concurrence comme dans les méthodes optimistes (grâce à la création de transactions fantômes).

3.3.4.2 Commentaires

Dans SCC, lors de la résolution des conflits, nous pouvons distinguer deux cas : (1) les conflits de type R-W et W-R qui sont résolus de manière similaire, et (2) les conflits de type W-W qui sont résolus d'une autre manière.

Dans un premier temps, considérons les conflits de type W-W. Ils sont résolus par la méthode TWR qui utilise les estampilles des transactions pour éventuellement ignorer certaines opérations d'écriture. Dans un contexte temps réel, l'hypothèse sous-jacente à la méthode TWR est difficilement applicable. En effet, si les estampilles des transactions représentent leurs dates d'arrivée dans le système, une transaction plus ancienne avec un temps d'exécution très long et loin de son échéance peut être en conflit avec une transaction plus jeune mais dont l'échéance est plus proche. Dans ce cas, l'application de TWR va ignorer l'opération d'écriture de la première transaction. Le problème n'est pas résolu de manière correcte puisque la 2^{eme} transaction pourrait avoir plus de chances de se terminer (valider) avant la 1^{ere}

et doit alors réécrire la donnée. Ainsi, la résolution des conflits W-W en utilisant les estampilles des transactions ne semble pas adaptée aux SGBDTR.

Considérons maintenant les autres types de conflits (R-W et W-R). Avec le protocole SCC, on peut se poser la question suivante : peut-on déterminer les cas où la transaction principale (d'origine) est abandonnée et que la transaction fantôme est exécutée ? Comme nous l'avons montré dans (Haubert et al., 2003a), certaines transactions ne peuvent pas respecter leurs échéances parce que le temps restant pour qu'elles s'exécutent pourrait ne pas être suffisant. Si les échéances des transactions sont de type *soft*, seule la qualité des résultats va diminuer (Bestavros and Braoudakis, 1995; Duvallet et al., 1999b). Mais si les transactions sont de type *firm* (voire *hard*), la transaction ne fournit aucun résultat ; ce qui pourrait engendrer des conséquences non souhaitées.

Dans ESCC, nous avons choisi de bloquer la transaction principale plutôt que la transaction fantôme. Les transactions en conflit pourraient ainsi toutes les deux valider et avoir plus de chances de se terminer avant échéance. En résumé, nous pouvons dire que le protocole SCC de base ne tient pas compte des priorités des transactions (échéance et/ou criticité) pour la résolution des conflits. L'extension proposée sera discutée et on montre que davantage de transactions respectent leurs échéances en utilisant : (1) une nouvelle technique de résolution des conflits W-W et (2) une nouvelle approche pour appréhender les échéances des transactions pour la résolution des conflits R-W et W-R.

3.3.4.3 Conflits W-W

La résolution des conflits basée sur les estampilles des transactions fournit des résultats discutables dans les SGBDTR. Pour les conflits W-W, nous proposons d'utiliser la même notion que pour les autres types de conflits : la duplication des transactions. De cette façon, une transaction fantôme est créée dès l'apparition d'un conflit. Le nombre de transactions fantômes par transaction peut être limité de la même manière que pour le protocole SCC-2S (Bestavros et al., 1993).

Examinons la possibilité d'appliquer la duplication des transactions aux conflits W-W. Lorsqu'un conflit de type W-W apparaît, on duplique la transaction dont l'échéance est la plus proche car c'est la transaction qui a le moins de chances de se terminer si on la redémarrait. Deux cas se présentent alors :

- soit le point de conflit se situe avant le début de la transaction fantôme et la transaction fantôme doit être redémarrée, et se bloque au nouveau point de conflit,
- soit le point de conflit est l'instant courant : il suffit de bloquer la transaction fantôme à l'instant courant.

  Ainsi, la duplication des transactions peut s'appliquer à tous les types de conflits.

3.3.4.4 Autres types de conflits

On montre dans (Haubert et al., 2004b) que le protocole SCC est limité pour résoudre les problèmes de conflits R-W et W-R entre des transactions concurrentes. Existe-t-il une méthode qui permette aux transactions en conflit R-W et W-R de respecter leurs échéances ? Pour cela, nous avons émis une nouvelle hypothèse : une transaction peut lire les résultats d'une autre transaction non encore validée. Nous relaxons ainsi la propriété d'isolation. Ainsi, si deux transactions accèdent à une même donnée en écriture et en lecture respectivement, alors dès qu'un conflit apparaît entre ces transactions, il y a création d'une transaction fantôme pour la transaction qui rentre en conflit. Le protocole ESCC étend le protocole SCC de la façon suivante :

1. Si la transaction de lecture a de grandes chances de se terminer avant l'autre transaction (de mise

à jour), alors l'écriture n'a pas d'impact sur le déroulement de l'exécution. De manière optimiste, on peut donc laisser les transactions se dérouler comme dans le protocole SCC de base.

2. Si c'est l'inverse, alors la transaction d'écriture aura plus de chances de se terminer en premier, et les modifications qu'elle aura apportées devront être considérées par la transaction de lecture. On choisit ici de bloquer la transaction principale et d'exécuter la transaction fantôme. Autrement dit, la transaction en lecture s'exécute avec l'*image après* de la donnée. Les deux transactions ont ainsi plus de chances de se terminer et de valider avant leurs échéances.

En combinant les échéances et la duplication des transactions dès l'apparition d'un conflit, le protocole ESCC permet donc à davantage de transactions de respecter leurs échéances.

3.3.4.5 Discussion sur la méthode

Comme nous l'avons signalé dans la section précédente, le protocole ESCC proposé ne respecte pas la propriété d'isolation des transactions. Dans un contexte temps réel, cette propriété peut être relaxée (Ramamritham, 1993) dans le sens où l'isolation peut différer l'exécution de certaines transactions et les empêcher de respecter leurs échéances. Si on relaxe la propriété d'isolation, alors les transactions peuvent accéder à des données non encore validées et peuvent ainsi fournir des résultats même partiels avant échéance. Autrement dit, dans un contexte temps réel, il peut être plus important d'obtenir des résultats partiels avant échéance plutôt que des résultats complets après échéance (Ramamritham, 1993).

Le fait de relaxer l'isolation des transactions oblige à faire davantage attention à la validité des données. Lorsqu'une transaction accède à des données déjà utilisées par une autre transaction, la validation de la 2^{eme} transaction est alors dépendante de celle de 1^{ere}. Si la 1^{ere} transaction se termine correctement et valide, alors la 2^{eme} peut continuer normalement son exécution. Mais, dans le cas où la 1^{ere} doit être abandonnée, la 2^{eme} doit également être abandonnée puisqu'elle a manipulé des données devenues invalides. Des heuristiques sur les conditions d'accès aux données non encore validées permettent notamment d'éviter l'abandon en cascade. Un autre exemple d'heuristique porte sur l'échéance de la transaction qui accède à la donnée : si cette transaction est proche de son échéance et risque donc de ne pas la respecter, alors la 2^{eme} transaction ne sera pas autorisée à accéder à la donnée.

Un autre point négatif du protocole ESCC est le nombre de transactions qu'implique la duplication. En effet, avec N transactions et en se limitant à une transaction fantôme par transaction principale, 2N transactions peuvent coexister. Mais, comme le montre la discussion précédente, la plupart du temps une transaction et sa transaction fantôme ne s'exécutent pas en même temps. La transaction fantôme est souvent bloquée pendant que la transaction principale s'exécute sauf dans le cas où la transaction fantôme est redémarrée. On peut donc penser que la surcharge entraînée par cette méthode est négligeable par rapport aux performances obtenues en terme de respect des échéances des transactions. Des simulations ont montré que c'est effectivement le cas (Bestavros et al., 1993).

3.3.4.6 Conclusion

Les améliorations que nous avons proposées avec le protocole ESCC sont fondées sur une approche qui prend en compte les échéances des transactions dans la résolution des conflits et une nouvelle résolution des conflits de type W-W. Le protocole SCC amélioré, appelé ESCC, propose d'utiliser à la fois la duplication des transactions et la prise en compte des échéances pour résoudre tous les types de conflits. Des simulations ont été effectuées pour évaluer les performances du protocole ESCC par rapport au protocole SCC. Elles ont montré que l'on obtient de meilleures performances[8] avec ESCC qu'avec SCC de base (Haubert et al., 2004a). Cependant, elles ont montré également les limites du protocole

[8] NombreDeTransactionsRespectantLesEch´enaces/NombreTotalDeTransactions.

ESCC dans la résolution de conflits W-W, même s'il améliore la méthode TWR.

Nos travaux peuvent être appliqués également aux SGBD distribués temps réel. La répartition des données implique de gérer à la fois le contrôle de concurrence localement sur chaque site, mais aussi la validation globale des transactions distribuées.

Ainsi, le protocole ESCC comme le protocole SCC peuvent être utilisés localement sur chaque site pour le contrôle de concurrence, mais il faut tenir compte de la validation globale des transactions. Certains protocoles de validation ne sont pas adaptés au contexte temps réel, soit parce qu'ils sont bloquants, soit à cause du grand nombre de messages échangés entre le coordonnateur et les participants (Bernstein et al., 1987; Haritsa et al., 2000). La validation des transactions distribuées temps réel reste un domaine de recherche ouvert. Nous y reviendrons dans le chapitre 4.

3.3.5 Encadrements, collaborations et diffusion des résultats

Les travaux décrits précédemment sur les notions de ε et Δ (paragraphe 3.3.1) sont le résultat d'une collaboration avec ma collègue S. Bouzefrane. Ces travaux ont donné lieu à plusieurs publications dans des conférences internationales (Sadeg and Bouzefrane, 1999; Bouzefrane and Sadeg, 2000a; Sadeg and Bouzefrane, 2000b) et nationales (Sadeg and Bouzefrane, 2000a), et dans un journal (Bouzefrane and Sadeg, 2000b).

Les travaux sur l'utilisation de ces notions dans les applications multimédia (paragraphe 3.3.2) ont été effectués en collaboration avec mes collègues L. Amanton et S. Bouzefrane. Ils ont donné lieu à plusieurs publications dans des conférences internationales (Sadeg and Bouzefrane, 2000a; Bouzefrane and Sadeg, 2000a; Bouzefrane et al., 2001; Sadeg et al., 2001c).

Quant aux travaux sur l'application des notions de ε et Δ dans les environnements mobiles (paragraphe 3.3.3), ils ont donné lieu à des publications dans les *proceedings* de conférences (Sadeg et al., 2001c; Amanton et al., 2001; Bouzefrane et al., 2001) et à un article dans un numéro spécial de revue (Sadeg et al., 2001b).

Ces travaux ont été effectués dans notre équipe de recherche SGBD temps réel du LIH, constituée alors de trois maîtres de conférences : L. Amanton, S. Bouzefrane et moi-même.

Les travaux sur l'amélioration du protocole SCC (paragraphe 3.3.4) sont effectués dans le cadre de la thèse de doctorat de Jérôme Haubert que je co-encadre et ils rentrent dans la réalisation du projet ACI-Jeunes Chercheurs #1055 que notre équipe de recherche a obtenu en 2002.

Ce travail a donné lieu a des publications dans une revue (Haubert et al., 2004b) et dans des conférences internationales (Haubert et al., 2004a; Haubert et al., 2003b; Haubert et al., 2003a).

4 Validation (Commit) des transactions distribuées temps réel

On trouve les bases de données distribuées dans des structures décentralisées d'applications de SGBDTR qui sont distribuées de manière inhérentes (ex. systèmes boursiers, marchés électroniques, systèmes de contrôle/commande). Les systèmes de bases de données distribués offrent des possibilités aux transactions de partager des donnés, où chaque transaction pourra accéder à des données sur des bases de données localisées sur des sites distants. Quand on ordonnance les transactions distribuées temps réel selon des critères temporels, il faudrait également assurer la cohérence globale de la base, ainsi que la cohérence locale sur chaque site.

Le modèle le plus commun pour les transactions distribuées est un système composé d'un site maître (le coordonnateur) où la transaction globale est soumise, puis subdivisée en sous-transactions, et de plusieurs sites participants auxquels sont soumises ces sous-transactions.

Une littérature abondante existe sur les résultats de travaux de recherche dédiés aux problèmes des SGBDTR distribués. Ces problèmes concernent le contrôle de concurrence (Lam et al., 1997a; Lam and Yan, 1998; Lam et al., 1999; Lam et al., 2000; Braoudakis, 1995; Ulusoy, 1993b; Ulusoy and Buchman, 1996), l'assignement d'échéances (Chen and Gruenwald, 1996), la réplication (Ulusoy, 1994b), l'exécution des transactions imbriquées (Chen and Gruenwald, 1996; Ulusoy, 1998c). Mais il n'en existe pas beaucoup sur la validation (Commit) des transactions temps réel distribuées (Gupta et al., 1996; Haritsa et al., 2000; Haritsa and Ramamritham, 2000a).

Parmi les problèmes posés par les SGBDTR distribués, le principal est la gestion des transactions et en particulier la validation (Commit) des transactions distribuées temps réel. Nous allons dans la suite présenter notre contribution sur cet aspect, après avoir rappelé brièvement quelques résultats sur le Commit des transactions distribuées non temps réel et résumé un travail de recherche sur un protocole de Commit des transactions distribuées temps réel.

4.1 Rappels sur les protocoles de Commit distribué

L'un des principaux problèmes posés par les architectures distribuées est la validation des transactions au niveau global. Le problème est autrement plus compliqué que dans le cas centralisé. Nous allons rappeler la notion de validation atomique (*Atomic Commitment*) (Bernstein et al., 1987).

4.1.1 Validation atomique

Soit une transaction T qui s'exécute sur les sites S_1, S_2, ..., S_n. Le gestionnaire de transaction (GT) sur le site maître S_1 supervise l'exécution. Avant qu'il n'envoie l'opération Commit aux autres sites, il doit s'assurer que l'ordonnanceur (ORD) et le gestionnaire de données (GD) sur chaque site sont prêts.

Les conditions sont :

(1) L'ORD sur un site peut accepter d'exécuter Commit(T) aussi longtemps que T satisfait à la condition de recouvrabilité sur ce site : chaque donnée lue par T sur ce site est écrite par une transaction ayant validée (c'est-à-dire ayant effectué son Commit).

(2) Le GD sur un site peut accepter d'exécuter Commit(T) aussi longtemps qu'il satisfait à la règle *Redo* (toutes les valeurs écrites par T sur ce site sont stockées sur support stable). Si T ne soumet que des lectures à un site, il n'est pas nécessaire qu'elle demande le consentement du GD de ce site.

(3) Le GT de S_1 peut initier Commit(T) aux ORD et GD de S_1, S_2, ..., S_n, seulement après avoir reçu leur consentement Il existe un processus sur chaque site où la transaction est exécutée. Ces processus réalisent la validation atomique.

Chaque site contient un fichier *log* de transactions distribuées (DT log) sur support stable où des informations sont enregistrées. Les DT log doivent survivre aux pannes.

Un protocole de Validation Atomique (ACP) est un algorithme pour le coordonnateur et les participants tel que : ou bien le coordonnateur et tous les participants valident la transaction, ou bien tous l'abandonnent. Plus précisément, chaque processus doit effectuer exactement un vote OUI ou un vote NON et atteindre exactement une décision : Commit ou Abort. Des conditions pour qu'un protocole de validation soit atomique et d'autres considérations relatives à la validation des transactions distribuées peuvent être trouvées dans (Bernstein et al., 1987).

4.1.2 Protocoles 2PC, PA, et PC

De nombreux protocoles ont été proposés pour la validation (Commit) des transactions dans les SGBD distribués (Abdallah et al., 1998; Abdallah and Pucheral, 1998; Guerraoui and Schiper, 1995; Haritsa et al., 1997). Ces travaux incluent le protocole classique *2PC*[1] et ses variantes : PC (*Presumed Commit*) et PA (*Presumed Abort*) , et le 3PC[2]. Tous ces protocoles nécessitent l'échange de plusieurs messages, au cours de plusieurs phases, entre les différents sites participants et le site coordonnateur. Le processus de Commit tient donc une place non négligeable dans le calcul du temps d'exécution d'une transaction distribuée (Abbot and Garcia-Molina, 1992). C'est pour cela que le choix d'un protocole de Commit est une décision importante dans la gestion des transactions temps réel distribuées.

Le protocole 2PC s'exécute en deux phases. Durant la première phase, le maître atteint une première décision (Valider ou Abandonner) basée sur les décisions locales des sites participants. Durant la deuxième phase, le maître transmet cette décision aux sites participants. Durant ce processus, le maître suppose que chaque site participant est capable de défaire les actions de mise à jour qu'il a effectuées en cas de décision d'abandon de la transaction, i.e. chaque site (maître ou participant) dispose d'un journal des mises à jour effectuées localement sur support stable.
Le protocole 2PC nécessite l'échange de plusieurs messages entre le coordonnateur et les participants et l'écriture (parfois forcée) des journaux (log) sur supports stables.

La variante, appelée PA, du protocole 2PC, tente de réduire cet inconvénient en adoptant la règle du "en cas de doute, abandonner". Ce protocole fonctionne de manière similaire au protocole 2PC pour les transactions qui valident (Commit), mais réduit le nombre de messages échangés et la taille des journaux écrits pour les transactions abandonnées.

[1] Commit `a 2 phases
[2] Commit `a 3 phases

L'autre variante, appelée PC, est issue du constat que, en général, il y a davantage de transactions qui seront validées que celles qui seront abandonnées. Dans cette variante, l'inconvénient est réduit pour les transactions qui valident. Il obéit à la règle du "en cas de doute, valider".

Le protocole 2PC et ses variantes ne sont pas adaptés au contexte temps réel car ils présentent deux inconvénients majeurs : la durée de l'échange de plusieurs messages peut être très longue et surtout ils sont sujets aux blocages. Le protocole 3PC a été proposé pour éviter les blocages, mais le nombre de messages échangés entre le coordonnateur et les participants est encore plus grand que dans le 2PC puisqu'il comporte une troisième phase.

À cause des inconvénients qu'ils recèlent, les protocoles présentés ne sont pas appropriés pour les SGBDTR distribués. De plus, ils n'intègrent pas de mécanismes qui puissent gérer les transactions distribuées selon leurs échéances. On peut trouver deux grandes lacunes dans ces protocoles relativement au respect des échéances : (1) en rendant inaccessibles les données *préparées*, ils augmentent les temps de blocage. Ce qui a pour effet l'abandon de plusieurs transactions dû aux manquements d'échéances, (2) ils ne tiennent pas compte des priorités des transactions. Si des ordonnancements basés sur les priorités sont proposés pour en tenir compte, cela conduira souvent au phénomène d'*inversion de priorité*.

Dans le paragraphe suivant, nous décrivons le protocole PROMPT[3], un protocole proposé dans (Haritsa et al., 2000) pour la validation des transactions distribuées temps réel. PROMPT permet d'atténuer les deux inconvénients précédents.

4.2 Brève présentation du protocole PROMPT

Le protocole PROMPT est proposé par Haritsa et al. (Haritsa et al., 2000). Il est appliqué à des transactions distribuées temps réel de type *firm*, où le Commit est défini comme suit :

On dit qu'une transaction de type *firm* a effectué son *Commit* si le site maître a atteint la décision finale (c-à-d qu'il a forcé l'écriture sur son journal local de l'enregistrement Commit), avant l'expiration de son échéance sur ce site.

La principale caractéristique du protocole PROMPT est qu'il permet à des transactions d'accéder à des données *préparées* par d'autres transactions. En d'autres termes, des transactions en *phase d'incertitude* peuvent prêter les données qu'elles ont mises à jour à d'autres transactions. Selon la décision reçue (COMMIT ou ABORT) et selon qui la reçoit en premier (le prêteur ou l'emprunteur), le mécanisme utilisé peut conduire à l'une des situations suivantes :

1. Le prêteur reçoit la décision avant que l'emprunteur ne termine son exécution : si la décision est COMMIT, alors l'emprunteur poursuit normalement son exécution. En effet, il aura utilisé des données mises à jour par une transaction qui a finalement validé. Si par contre la décision globale était ABORT, alors le prêteur sera abandonné normalement. De plus, l'emprunteur sera également abandonné car il aura utilisé des données invalides.
2. L'emprunteur reçoit la décision avant que le prêteur ne reçoive la décision : l'emprunteur est dans ce cas obligé d'attendre et n'est pas autorisé à envoyer un message DONE (travail effectué) à son coordonnateur. Cette attente va durer jusqu'à ce que le prêteur reçoive la décision ou que son échéance expire. Dans le premier cas, si la décision reçue par le prêteur est COMMIT, le prêteur

[3]Permits Reading Of Modifi ed Prepared-data for Timeliness

peut envoyer le message DONE à son coordonnateur. Si par contre la décision est ABORT, alors l'emprunteur doit également être abandonné car il aura utilisé des données invalides.

3. L'emprunteur est abandonné pendant son exécution et avant que le prêteur ne reçoive la décision : l'abandon de l'emprunteur (pour cause d'échéance manquée, problème local ou réception d'un message ABORT de son coordonnateur), a pour cause d'annuler ses éventuelles mises à jour et d'annuler le prêt tout simplement.

En résumé, le protocole PROMPT permet aux transactions d'accéder à des données *non committées*[4], dans l'espoir (vision optimiste) que la transaction ayant mis à jour ses données finira par recevoir la décision COMMIT de son coordonnateur. Cela permet de pallier les problèmes d'inaccessibilité des données et d'inversion de priorité cités précédemment.

Des caractéristiques améliorant les aspects temps réel du protocole PROMPT sont ajoutées. Elles sont décrites ci-dessous :

1. Abandon actif : dans un SGBDTR distribué une transaction qui n'est pas encore dans sa phase de Commit pourra être abandonnée à cause d'un conflit avec une transaction de plus haute priorité. Dans PROMPT, la transaction ainsi abandonnée avertit immédiatement son coordonnateur, qui avertira ainsi plus tôt les autres sites participants. La transaction est abandonnée plutôt et donc redémarrée plutôt. Ce qui lui donne davantage de temps pour sa nouvelle exécution et donc davantage de chances de respecter son échéance.
2. Abandon silencieux : si une transaction est abandonnée au niveau d'un site participant avant que son coordonnateur n'ait lancé le processus de décision, alors elle n'a pas besoin d'avertir son coordonnateur, évitant ainsi un round d'échange de messages.
3. Prêt sain : une transaction en phase d'incertitude n'est autorisée à prêter ses données que si elle n'est pas proche de son échéance. Ceci évite que des données soient prêtées par une transaction qui risque fort de manquer son échéance, conduisant ainsi à l'abandon des transactions emprunteuses. Ce critère est mis en oeuvre par l'assignation d'un facteur, appelé FS (facteur de santé). Une transaction n'est alors autorisée à prêter ses données que si son facteur FS dépasse un certain seuil fixé.

Notons qu'une transaction emprunteuse ne peut pas être en même temps prêteuse, puisque seules les transactions en phase d'incertitude (*préparées*) peuvent prêter leurs données.

Dans (Haritsa et al., 2000), les auteurs ont montré que le protocole PROMPT apporte des améliorations significatives par rapport aux protocoles standards. Ils ont montré également qu'en positionnant le *seuil du facteur* de santé à la *bonne* valeur, les mauvais comportements du protocole PROMPT relativement au prêt des données peuvent être éliminés.

4.3 Notre contribution

Notre contribution à la gestion des transaction temps réel inclut l'aspect distribué, notamment le processus de validation (Commit). Dans ce cadre, nous avons d'abord basé notre travail sur l'amélioration de protocoles de Commit des transactions distribuées temps réel. Puis nous avons proposé de nouveaux protocoles pour des transactions distribuées temps réel ayant certaines caractéristiques.

Dans (Sadeg and Haubert, 2002), nous avons proposé l'amélioration du protocole de Wang et Mitra (Wong and Mitra, 1996). L'amélioration a résulté en de meilleures performances du protocole qui

[4] mises `a jour par des transactions n'ayant pas encore effectu´e leur op´eration COMMIT

permet ainsi à davantage de transactions de valider avant leurs échéances. Dans leur article, Wong et Mitra ont proposé d'utiliser les états des transactions temps réel distribuées pour effectuer la validation de celles-ci, en se basant sur un automate d'états/transations.

Le protocole que nous avons proposé optimisait l'architecture sous-jacente en supprimant les états inutiles en fonction de certaines hypothèses et en en modifiant d'autres. Le travail a été réalisé en collaboration avec le doctorant J. Haubert et a donné lieu à une communication dans la conférence maghrébine MCSEAI (Sadeg and Haubert, 2002).

Ensuite nous avons proposé d'autres protocoles de Commit dont certains sont basés sur des transactions pondérées, d'autres sur la duplication des transactions et d'autres encore sur les protocoles d'ordre causal par phases. Nous en donnons une description dans les paragraphes suivants.

4.3.1 Protocole WEP (Weighted Early Protocol)

4.3.1.1 Introduction

Nous considérons dans ce travail des transactions que l'on peut scinder en plusieurs sous-transactions d'inégale importance. Le facteur *importance* associé à chaque transaction est représenté par un poids. Un seuil fixé permet de distinguer les sous-transactions obligatoires des sous-transactions optionnelles.

Ensuite, le site coordonnateur est autorisé à lancer le processus de Commit d'une transaction même si des sous-transactions optionnelles n'ont pas encore effectué leur vote. La condition est que toutes les sous-transactions obligatoires aient voté OUI.

L'idée de considérer des unités d'exécution obligatoires et optionnelles a déjà été proposée dans le contexte d'un système temps réel, où une tâche est divisée en partie obligatoire qui doit respecter son échéance, et une partie optionnelle qui peut être omise dans le cas où le système n'a pas la possibilité de lui allouer toutes les ressources dont elle a besoin (Liu et al., 1991), en cas de surcharge par exemple.

Nous avons généralisé cette idée en faisant en sorte que chaque sous-transaction se voit assigner un poids (nombre réel positif) qui détermine son importance dans le système, en plus de son échéance (qui est la même que l'échéance globale de la transaction). Ensuite, un seuil est fixé au départ, par l'administrateur de la base par exemple, et servira pendant la résolution des conflits d'accès aux données et pendant l'ordonnancement, à déterminer les sous-transactions obligatoires et les sous-transactions optionnelles : les sous-transactions dont le poids est supérieur ou égal à la valeur du seuil sont déclarées obligatoires, les autres sont optionnelles.

4.3.1.2 Description du protocole WEP

Contrairement aux autres protocoles de Commit distribué (2PC et ses variantes, ou 3PC) (Bernstein et al., 1987), le protocole 1-PC proposé par Abdallah et al. (Abdallah et al., 1998; Abdallah and Pucheral, 1998), minimise le nombre de messages échangés entre le coordonnateur et les participants. Dans (Haritsa and Ramamritham, 2000a), Haritsa et al. ont proposé le protocole PEP (Priority Early Prepare) basé sur le protocole de Commit à une phase EP, proposé pour le processus de Commit des transactions distribuées temps réel. Dans le protocole PEP, le participant qui termine l'exécution de sa sous-transaction prend l'initiative d'envoyer un message DONE au coordonnateur, et entre dans sa phase de préparation (d'incertitude).

Le protocole WEP (Weighted Early Prepare) que nous proposons est une adaptation du protocole PEP aux transactions pondérées. Quand le coordonnateur reçoit une transaction, il la subdivise en sous-

transactions qui sont soumises aux sites participants adéquats (où sont localisées les données à accéder).

Dans le protocole PEP, toutes les sous-transactions d'une même transaction ont la même importance. Cependant, dans beaucoup d'applications temps réel, on arrive souvent à distinguer des parties obligatoires et des parties optionnelles (Aydin et al., 2001).

Quand le coordonnateur reçoit des messages de vote OUI depuis les participants, si la transaction est proche de son échéance, alors le coordonnateur teste si toutes les sous-transactions obligatoires ont envoyé leur vote OUI, auquel cas il lance le processus de Commit sans attendre les votes des autres sous-transactions.

4.3.1.3 Simulations et résultats

Le protocole WEP est simulé sous le système d'exploitation Linux, sous forme de deux classes C++ : une qui gère les opérations des transactions (initialisation, distribution, validation, ...), et une autre utilisée pour définir des fonctions de gestion des files de transactions (enfiler, défiler, FileVide, ...). Le module principal :

– initialise le coordonnateur et les participants, et
– lance l'exécution : gestionnaire de temps, initialisation des files, gestionnaires de transactions

Périodiquement, le gestionnaire de temps teste si les échéances sont respectées et quel type de sous-transaction (obligatoire/optionnelle) à abandonner. Si nécessaire, il informe le coordonnateur de la (ou des) sous-transaction(s) concernée(s) pour qu'il envoie le message $ABORT$ aux participants.

La prise en compte des poids des sous-transactions est effectuée en modifiant le paramètre *poids* dans les différents *constructeurs*. Ce paramètre est également ajouté dans le fichier d'entrée des données. Nous avons effectué les simulations avec 3 valeurs du seuil, où chaque valeur est utilisée pour déterminer si une sous-transaction est obligatoire ou optionnelle. Un seuil de 0 correspond au protocole PEP de base.

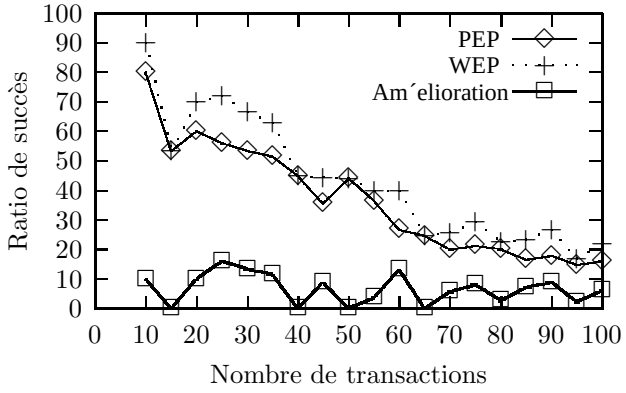

FIG. 4.1 – Performances des protocoles PEP vs WEP

La figure 4.1 montre que le protocole WEP présente des performances meilleures que le protocole PEP de base : environ 15% de transactions en plus respectent leurs échéances.

Dans la figure 4.2, on constate l'influence de la valeur du seuil : plus sa valeur est élevée, moins de transactions sont obligatoires. Donc davantage de transactions respectent leurs échéances, puisqu'elles

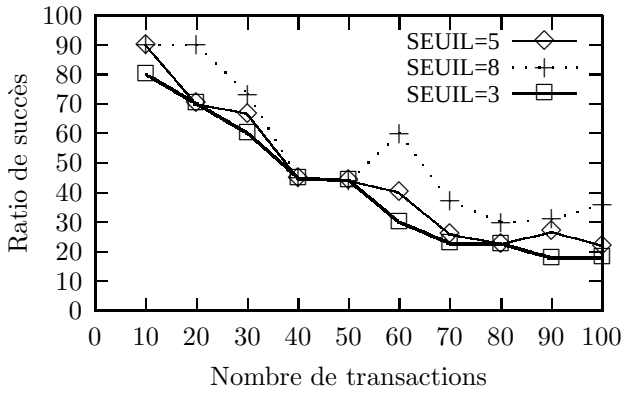

FIG. 4.2 – Influence de la valeur du seuil

n'exigent la terminaison que d'un très faible nombre sous-transactions obligatoires pour effectuer leur commit.

Comme suite à notre travail, nous comptons effectuer des études et d'autres simulations pour déterminer la valeur du seuil qui soit un compromis entre le ratio de succès des transactions (de manière à ce qu'un maximum de transactions respectent leurs échéances) et l'acceptabilité du résultat fourni, c-à-d pour qu'il soit toujours exploitable.

4.3.2 Protocole basé sur les transactions (m, k)-$firm$

Un travail basé sur une une idée similaire à la précédente et utilisant les contraintes (m, k) - $firm$ (Hamdaoui and Ramanathan, 1995) a également été proposé.

Dans ce travail, nous avons considéré une transaction temps réel distribuée, composée de k sous-transactions dont m sont obligatoires, les autres sont optionnelles (Haubert et al., 2004c). Nous avons alors proposé un algorithme de CC et de Commit des transactions distribuées possédant cette caractéristique. Les sous-transactions obligatoires doivent se terminer avec succès avant leurs échéances. Ensuite, le protocole tente d'améliorer la qualité de service offerte par la transaction en exécutant le plus grand nombre possible de sous-transactions optionnelles.

4.3.2.1 Illustration

Nous considérons un SGBD distribué où il y a un site coordonnateur et trois sites participants.

La figure 4.3 montre un exemple d'exécution de deux transactions (m, k)-$firm$ sur ce système.

T_1 composée des deux sous-transactions $ST_{1,1}$ and $ST_{1,2}$ est (2,2)-firm, i.e. elle correspond à une transaction de type *firm* "classique". T_2 est composée de trois sous-transactions ($ST_{2,1}$, $ST_{2,2}$ et $ST_{2,3}$) ; $ST_{2,1}$ et $ST_{2,3}$ sont obligatoires. Donc, T_2 est une transaction (2,3)-firm. Le coordonnateur reçoit les deux transactions et les subdivisent en sous-transactions qu'il envoie aux sites participants adéquats. Le premier site exécute les sous-transactions $ST_{1,1}$ et $ST_{2,1}$, le second site exécute les sous-transactions $ST_{1,2}$ et $ST_{2,2}$, et le dernier site exécute la sous-transaction $ST_{2,3}$.

Des conflits d'accès aux données peuvent survenir au niveau de chaque site, par exemple entre T_1 et T_2 sur les sites S1 et S2.

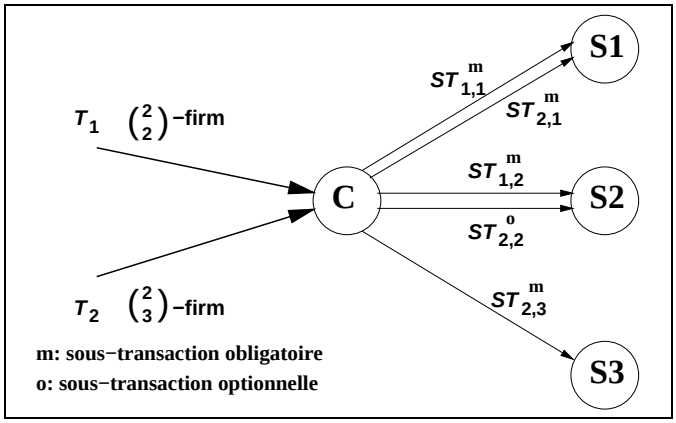

FIG. 4.3 – Exécution de deux transactions (m,k)-firm

Dans le présent travail, nous avons distingué les différents cas de conflits : entre sous-transactions obligatoires et/ou sous-transactions optionnelles.

L'objectif est de garantir le respect des échéances des parties obligatoires des transactions et de maximiser, pour chaque transaction, le nombre des ses sous-transactions optionnelles qui s'exécutent avant échéance.

4.3.2.2 Simulations et résultats

– Modèle de simulation

Nous utilisons un modèle de SGBDTR distribué avec un site coordonnateur et 10 sites participants.

Pour distinguer les sous-transactions obligatoires des sous-transactions optionnelles, nous n'effectuons pas d'analyse hors ligne mais nous utilisons une formule (Haubert et al., 2004d) qui permet de distribuer régulièrement les sous-transactions en obligatoires et optionnelles en fonction de leur numéro (qui correspond au numéro de site dans lequel chaque sous-transaction s'exécute).

Nous avons simulé 10 participants dans lesquels sont localisées 1000 données de la base. Le nombre de transactions oscille entre 100 et 1000 et chaque transaction est composée de 1 à 10 opérations sur la base. La probabilité qu'une opération soit une écriture (resp. une lecture) est de 0.5. Nous supposons que les durées d'une opération d'écriture et de lecture sont de 5 et 2 unités, respectivement. L'arrivée des transactions suit une loi uniforme et les transactions sont de type *firm*.

Les paramètres de simulation sont résumés dans la table 4.1.

– Résultats

Dans la figure 4.5, les résultats des simulations ont montré que l'utilisation du protocole (m,k)-firm exhibe des performances meilleures que l'utilisation des protocoles spéculatifs (comme SCC-2S (Bestavros and Braoudakis, 1996)) et 2PL-HP (Abbot and Garcia-Molina, 1992). L'amélioration n'est pas significative quand il y a relativement peu de transactions. Elle devient importante (environ 25% par rapport au protocole SCC-2S et 45% par rapport au protocole 2PL-HP), pour un nombre de transactions plus élevé (à partir de 150-200 transactions). Dans le protocole (m,k)-

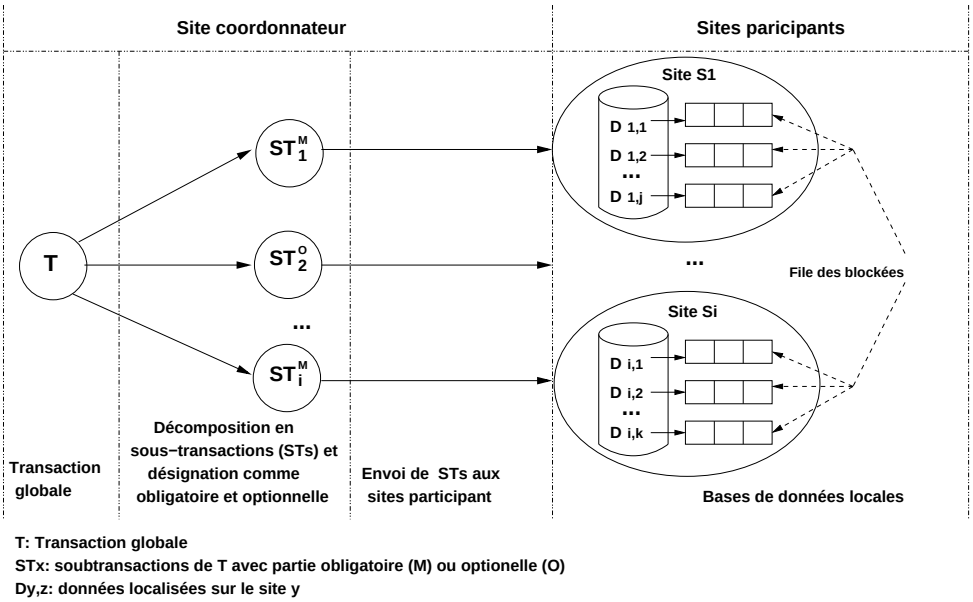

FIG. 4.4 – Modèle de simulation

Paramètere	Désignation	Valeur
TailleBD	Taille de la base de donn´ees	1000
NbSites	Nombre de sites	10
TailleSite	Taille d'une base locale	TailleBD /NbSites
NbTr	Nombre de transactions	100-1000
TailleTr	Nombre d'op´erations par sous-transaction	1-10
TauxAr	Taux d'arriv´ee	Uniforme
ProbaEcr	Probabilit´e d'´ecriture	0.5
TpsLect	Temps d'ex´ecution d'une op´eration de lecture	2 unit´es de temps
TpsEcr	Temps d'ex´ecution d'une op´eration d'´ecriture	5 unit´es de temps
Ech	Criticit´e des ´ech´eances des transactions	firm

TAB. 4.1 – Paramètres utilisés dans la simulation

firm que nous avons proposé, les priorités des transactions sont déterminées en fonction de plusieurs paramètres : l'échéance, le poids et les paramètres m et k, alors que dans le 2PL-HP et le SCC-2S, seules les échéances sont prises en compte.

Dans l'expérience de la figure 4.6, nous avons mis en évidence les meilleures performances du modèle (m,k)-firm en fonction de la charge du système, par rapport au modèle traditionnel. Nous avons fixé le ratio m/k à 0.5. La comparaison a été effectuée entre le protocole (m,k)-firm général et ceux que l'on a appelé sur la figure référence-0 et référence-k, qui correspondent respectivement aux protocoles (0,k)-firm et (k,k) firm. L'amélioration est d'environ 20% par rapport au protocole référence-k. Ceci est du au fait que dans le protocole référence-k, toutes les sous-transactions sont obligatoires. Les meilleures performances sont obtenues avec le protocole référence-0, où toutes les transactions sont optionnelles.

L'expérience de la figure 4.7 évalue la QdS des transactions en termes de nombre de sous-transactions optionnelles ayant terminé avant échéance, i.e. elle montre l'influence du ratio m/k sur les performances. Par exemple, quand m/k=1, toutes les sous-transactions sont obligatoires et la situation devient équivalente au cas de transactions de type *firm* "traditionnelles".

Les simulations de la figure montrent que quand le ratio augmente, i.e. on a des (k,k)-transactions,

les performances deviennent moins bonnes, vu que le nombre de sous-transactions critiques augmente.

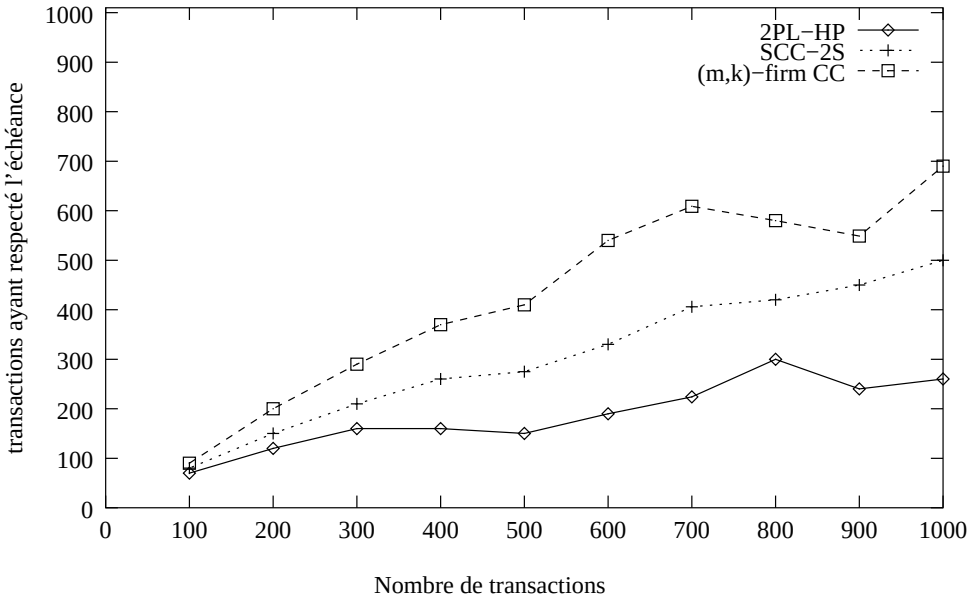

FIG. 4.5 – Comparaison des performances : protocoles (m,k)-firm, SCC-2S, 2PL-HP

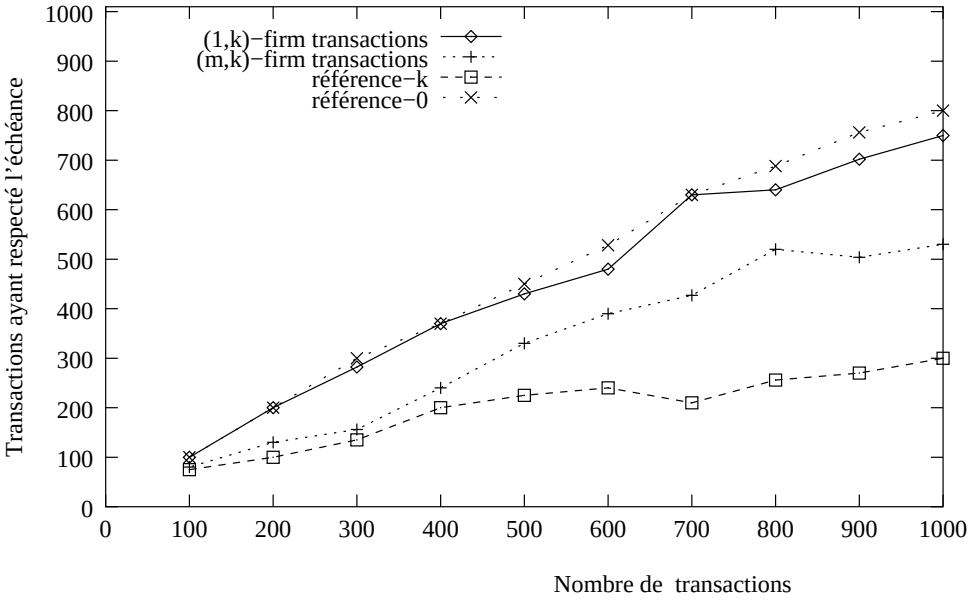

FIG. 4.6 – Résultats de simulation : Performances du modèle de transactions (m,k)-firm

En résumé, nous pouvons dire que le modèle de transactions (m,k)-firm permet une meilleure flexibilité dans la gestion des transactions distribuées temps réel, pour ajuster les performances aux besoins de l'application, i.e. augmenter ou diminuer la valeur de m.

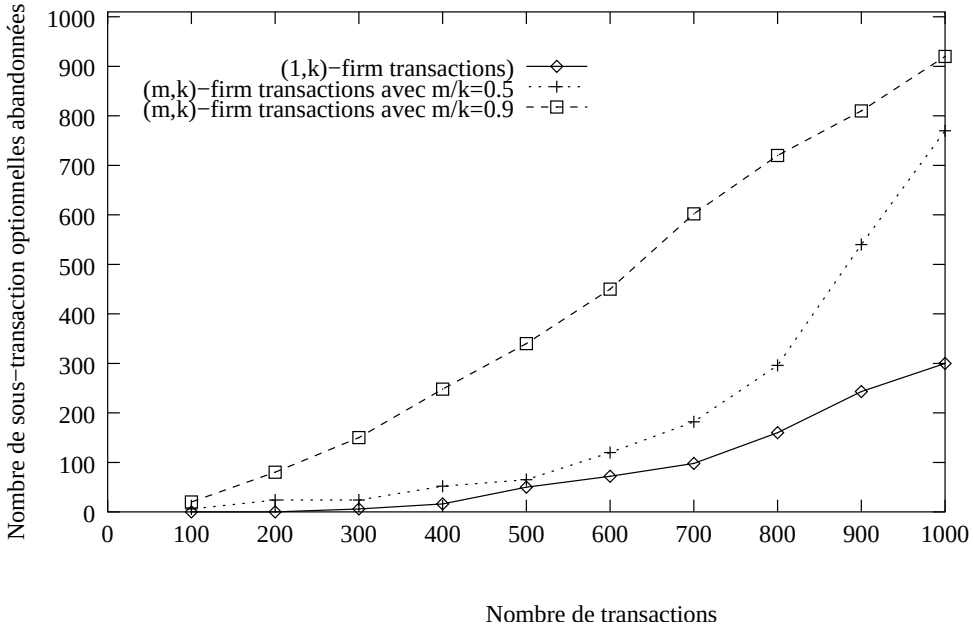

FIG. 4.7 – Évolution du taux de succès quand le ratio m/k augmente

4.3.3 Protocole D_ANTICIP

Dans nos travaux sur le commit des transactions distribuées temps réel, nous utilisons le modèle classique de SGBD distribué avec un site coordonnateur et des sites participants. Les transactions sont soumises au site coordonnateur, qui les analyse et envoie les sous-transactions aux sites participants appropriés, en fonction de la localisation des données sur lesquelles travaille chaque sous-transaction. Nous considérons que les transactions sont de type strict non critique (*firm*) et qu'une sous-transaction possède la même échéance que sa transaction-mère[5].

4.3.3.1 Principe du protocole

Le protocole $D\_ANTICIP$ (Descending-ANTICIP) que nous proposons est basé sur le principe de relaxation de la propriété d'isolation comme dans (Gupta et al., 1996), mais également sur celui de la duplication des sous-transactions comme le protocole de CC proposé pour le contexte centralisé par Bestavros et Braoudakis (Bestavros and Braoudakis, 1995).

L'idée de base du protocole est que, dès qu'un conflit survient entre une sous-transaction W_A qui arrive sur un site avec une autre (sous-)transaction déjà active W_E, alors W_A est dupliquée de sorte qu'une copie s'exécute avec l'*image avant* des données et l'autre copie est bloquée en attendant que l'*image après* soit disponible. Ces données après mise à jour par W_E sont "prêtées" à la copie de W_A même si la sous-transaction W_E est dans son état d'incertitude. Chaque sous-transaction qui arrive sur un site suit le précédent schéma, augmentant ainsi le nombre de transaction en exécution sur le site.

L'exécution des sous-transactions peut être représentée par une arborescence (voir Figure 4.8). La figure 4.9 illustre le cas où le parent d'une sous-transaction a effectué son Commit. Il s'agit d'une sous-transaction ayant prêté ses données non encore validées à d'autres sous-transactions actives ou à leurs copies. Dans ce cas, toutes les copies de sous-transactions ayant utilisé les données "avant mise à

[5]La transaction dont elle est issue.

jour" (l'image avant) sont abandonnées. L'ordre d'abandon est ici descendant contrairement à une autre variante du protocole que nous avions proposée et où l'abandon des sous-transactions inutiles s'effectue dans l'ordre ascendant (Sadeg et al., 2001d).

Toujours dans (Sadeg et al., 2002b), nous avons effectué des simulations de l'algorithme qui ont montré que les performances obtenues sont meilleures avec le protocole $D\_ANTICIP$ par rapport à l'utilisation des protocoles tels que le 2PC.

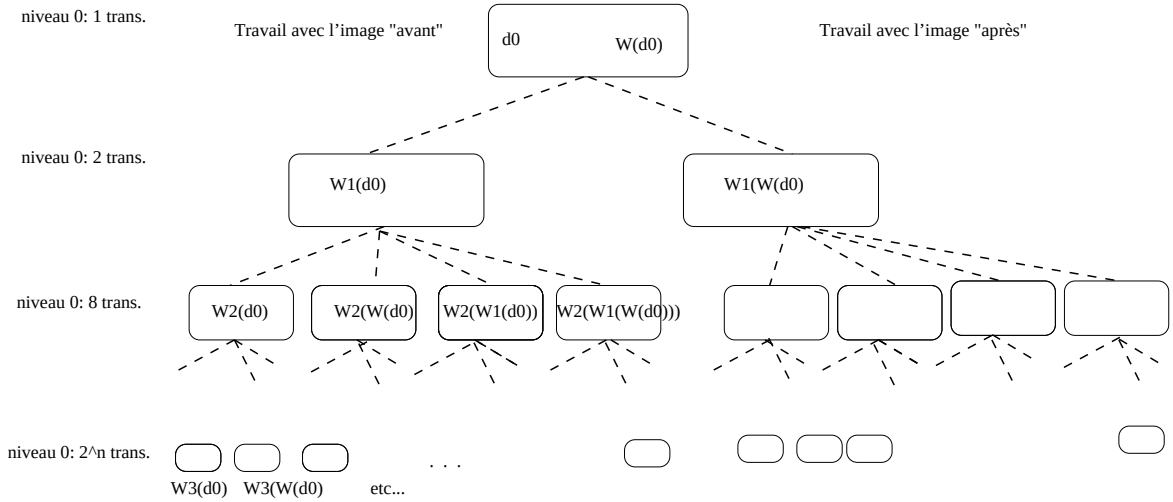

FIG. 4.8 – Illustration du modèle de copies de sous-transactions

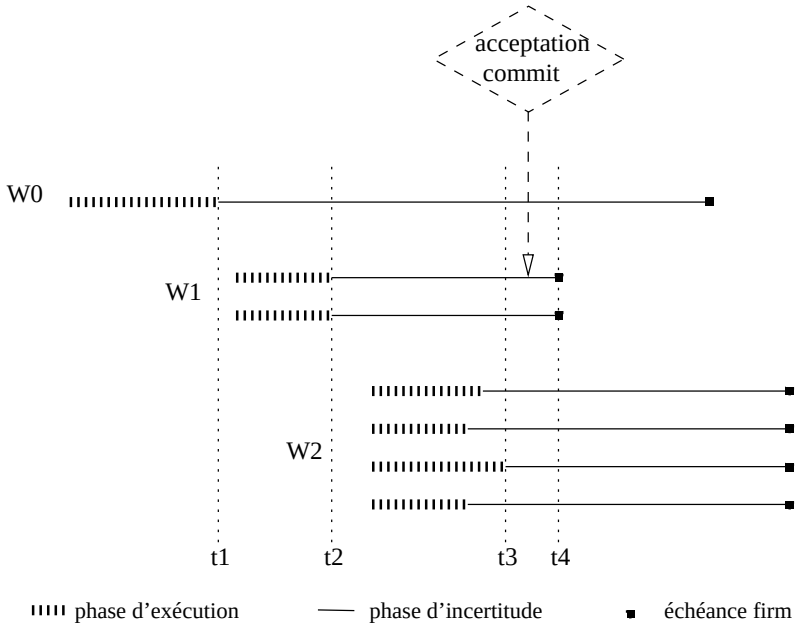

FIG. 4.9 – Schéma des sous-transactions quand la transaction *prêteuse* a validé

4.3.3.2 Conclusion

Le protocole proposé permet à davantage de transactions distribuées temps réel d'effectuer leur Commit avant l'expiration de leurs échéances. Comme le protocole est basé sur la duplication des sous-transactions, le problème est le risque de surcharge du système. Mais, ce problème est atténué grâce à une politique efficace d'abandon de sous-transactions devenues inutiles.

Les perspectives de nos travaux sont l'utilisation des notions de ϵ-données et Δ-échéance pour le commit des transactions distribuées, d'une part, et pour la minimisation de la durée pendant laquelle une sous-transaction demeure dans un état d'incertitude, d'autre part.

4.3.4 Protocole RT_DIS_COM

4.3.4.1 Introduction

Dans un système distribué, des algorithmes maintenant une synchronisation virtuelle entre les sites ont été déjà utilisés avec succès (Birman et al., 1991; Raynal and Schiper, 1995).
Un protocole d'ordre causal assure que si deux messages sont causallement reliés et ont la même destination, alors ils sont délivrés à l'application dans le même ordre. Cette propriété nous est très utile dans le cas de l'exécution de transactions distribuées temps réel ayant, dès leur soumision, des relations de précédence.

Dans un SGBD distribué, une transaction est subdivisée en sous-transactions qui vont s'exécuter sur des serveurs distincts. Dans le travail que nous allons présenter, nous utilisons un algorithme d'ordre causal (Amanton and Naïmi, 1996; Baldoni et al., 1996b; Mostefaoui and Theel, 1996; Birman et al., 1991) pour sérialiser l'exécution des transactions. La méthode consiste à construire des "coupes" consistantes entre les transactions en conflit.

Le résultat est un protocole, appelé RT_DIS_COM (Real-Time DIStributed COMmit), qui permet de gérer les transactions distribuées de type *soft*, subdivisées en sous-transactions (ou actions atomiques) envoyées sur les sites où sont localisées les données auxquelles elles accèdent.

4.3.4.2 Ordre causal par phases

La majorité des programmes peuvent être subdivisés en plusieurs transactions. Dans un contexte distribué, le problème le plus crucial est l'ordre des communications entre les différentes composantes du système, qui peut être total ou partiel pour satisfaire les contraintes des applications.

Pour construire un ordre, on peut : 1) ou bien ordonner les sous-transactions des transactions, 2) ou bien couper l'exécution des transactions en plusieurs phases consistantes, donc en plusieurs groupes de transactions ou de sous-transactions. La deuxième voie utilise des protocoles comme la *Re-Synchronisation*.
Basé sur ces travaux, un nouvel ordre a été introduit : l'ordre causal par phases (Amanton and Naimi, 1999).

Dans les travaux que nous présentons, un protocole de Commit des transactions distribuées a été proposé. Les sous-transactions soumises aux sites participants s'exécutent dans des groupes temporels, appelés phases. Ce protocole exploite des résultats obtenus par Amanton et Naimi (Amanton and Naïmi, 1998; Amanton and Naimi, 1999) sur l'ordre causal par phases.

4.3.4.3 Principe du protocole

Si dans un SGBD distribué, au sein d'un site, des sous-transactions ne sont pas en conflit, alors elles peuvent être exécutées dans la même phase sur les serveurs. Le site coordonnateur gère ce groupe de sous-transactions de la manière suivante :

- quand la transaction courante ne génère pas de conflit avec aucune autre transaction d'un groupe, elle est ajoutée à ce groupe.
- quand la transaction courante est susceptible de créer un conflit, le coordonnateur commence une nouvelle phase : toutes les transactions du groupe précédent doivent être terminées avant le début de la transaction courante, et des suivantes. Le groupe est ensuite effacé et réinitialisé avec la nouvelle transaction dans une nouvelle phase.
- quand une transaction d'un groupe a été validée (Commit) ou est abandonnée (Abort), elle est retirée du groupe.

Le protocole d'ordre causal par phases garantit que les sous-transactions de deux groupes successifs vont être exécutées sur les serveurs distribués selon leur ordre causal. Ce qui permet d'éviter les conflits et de préserver la cohérence des données.

La description détaillée du protocole ainsi que la démonstration de sa correction sont donnés dans (Sadeg et al., 2001a; Amanton et al., 2002b). Il a été montré (Amanton and Sadeg, 2003) que ce protocole suit les règles d'un protocole de validation atomique (Bernstein et al., 1987), d'une part, et qu'il possède un comportement équivalent au protocole PROMPT (Haritsa et al., 2000) dans beaucoup de situations, tout en s'avérant meilleur dans certains cas (Amanton and Sadeg, 2003), i.e. davantage de transactions respectent leur échéances.

4.3.4.4 Conclusion

La suite que nous comptons donner ce travail consiste à se baser sur un protocole d'ordre causal par phases qui utiliserait à la fois la technique de prêt des données comme dans PROMPT (Haritsa et al., 2000), ainsi que la duplication de sous-transactions comme dans les protocoles de contrôle de concurrence spéculatifs (Bestavros, 1992).

4.3.5 Encadrements, collaborations et diffusion des résultats

Les travaux sur le protocole WEP ainsi que ceux sur les transactions (m,k)-$firm$ rentrent dans le cadre de la thèse de Doctorat de Jérôme Haubert, entamée en 2001. Ils rentrent également dans le projet ACI-Jeunes Cherheurs #1055. Les participants à ces travaux sont le doctorant et deux maîtres de conférences L. Amanton et moi-même.

Les travaux du paragraphe 4.3.1 sont publiés dans la conférence internationale IEEE ISSPIT'03 (Haubert et al., 2003c) et quelques résultats relatifs aux travaux présentés dans le paragraphe 4.3.2 sont publiés dans les *WIP*[6]*-proceedings* de la conférence internationale Euromicro'04 (Haubert et al., 2004c). Des résultats plus complets sont publiés dans la conférence internationale RTCSA'04 (Haubert et al., 2004d).

Les travaux présentés dans le paragraphe 4.3.3 ont été effectués alors qu'il n'existait pas encore de DEA à l'université du Havre. Les participants aux travaux sont L. Amanton, S. Bouzefrane et moi-même. Les travaux ont donné lieu à publication dans la conférence internationale ICEIS'02 (Sadeg et al., 2002b).

[6]Work In Progess.

Quant aux travaux sur le protocole RT_DIS_COM (paragraphe 4.3.4), ils ont été initiés dans l'équipe SGBDTR et sont le résultat d'une collaboration entre L. Amanton, S. Bouzefrane et moi-même. Ils ont donné lieu à une publication dans une conférence internationale (Sadeg et al., 2001a). Ils se sont poursuivis entre L. Amanton et moi-même et ont donné lieu à un article dans une conférence internationale (Amanton and Sadeg, 2003).

5 Transactions imbriquées temps réel

5.1 Introduction

Les bases de données sont déployées dans de plus en plus de domaines d'applications pour stocker et manipuler l'information qui agit sur les limites des fonctionnalités et des performances des techniques de gestion des transactions traditionnelles. Parmi ces applications, on peut citer les applications de génie logiciel (CAD/CASE[1]), de gestion des réseaux et de multimédia, et d'un grand nombre de nouvelles applications temps réel.

L'industrie est également en train d'utiliser de manière intensive les transactions (avec des besoins de plus en plus sophistiqués). De grands progrès ont été effectués pour l'amélioration des performances des transactions traditionnelles. Dans le même temps, des avancées significatives sont effectuées dans l'étude des modèles de transactions traditionnels, ainsi que des traditionnels critères de correction[2].

Ces techniques peuvent être classifiées en trois catégories (Ramamritham and Chrysanthis, 1997) :

- Des extensions spécifiques apportées aux protocoles de contrôle de concurrence, de recouvrement et de validation.
- L'ajout de paramètres d'entrée dans la programmation des transactions qui permettent au système d'exécuter de manière plus *intelligente* les transactions tout en garantissant la sérialisabilité.
- L'exploitation de la sémantique des données et des opérations définies sur ces données, de la structure de la base, des propriétés comportementales des activités de la base et des critères de correction dont ont besoin les applications.

Nous consacrons cette section à nos travaux concernant les modèles étendus de transactions, et particulièrement l'utilisation du modèle de transactions imbriquées (*Nested Transactions Model*) dans les SGBDTR. Nous présentons les résultats auxquels nous avons abouti, notamment de nouveaux protocoles de contrôle de concurrence et une nouvelle technique d'utilisation des verrous, en utilisant la relaxation de la propriété d'isolation des sous-transactions d'une transaction. L'objectif étant d'améliorer les performances des SGBDTR en termes de nombre de transactions qui respectent leurs échéances. Nous commençons par décrire brièvement les modèle de transactions imbriquées.

5.2 Modèles de transactions imbriquées

Les modèles de transactions "plates" (*Flat Transactions*) ne sont pas toujours bien adaptés aux SGBDTR. Certaines applications temps réel ont en effet besoin de modèles de transactions où l'on peut relaxer certaines propriétés ACID des transactions, notamment l'atomicité et l'isolation. Pour aboutir à une décomposition des transactions qui puisse satisfaire cet objectif, le modèle de transactions im-

[1] Computer-Aided Design/Computer-Aided Software Engineering.
[2] La s´erialisabilit´e et les propri´et´es ACID.

briquées paraît convenir. Le premier modèle de transactions imbriquées a été introduit par Moss (Moss, 1985).

Dans ce modèle, une transaction est considérée comme une hiérarchie de sous-transactions, et chaque sous-transaction peut contenir, ou bien d'autres sous-transactions, ou bien des opérations élémentaires sur la base (lire ou écrire). En général une transaction imbriquée est une collection de sous-transactions composant une unité atomique d'exécution. Mais dans un contexte temps réel, comme nous le verrons plus loin, la terminaison d'une transaction peut ne pas nécessiter la terminaison de toutes ses sous-transactions. C'est cette caractéristique que nous exploiterons pour les SGBDTR.

Le modèle de transactions imbriquées peut être utilisé de manière efficace pour les transactions ayant une longue durée de vie, i.e., CAD/CASE, mais également pour des transactions de plus courte durée de vie, comme les transactions temps réel. Dans ce dernier cas, une transaction est décomposée en une hiérarchie de sous-transactions d'inégales importances. L'objectif est de pouvoir autoriser, dans certains cas, une transaction à valider sans avoir besoin de valider toutes ses composantes.

Dans les modèles transactionnels classiques, une transaction se compose d'un ensemble d'opérations atomiques (*lire* et *écrire*) partiellement ordonnées. Les transactions plates sont celles qui ont un simple point de début et un simple point de terminaison. Les transactions imbriquées, dérivées des transactions plates, permettent à une transaction d'invoquer aussi bien des transactions atomiques que des opérations atomiques.

Une transaction imbriquée peut contenir un nombre quelconque de sous-transactions et chaque sous-transaction, à son tour, peut être constituée d'un nombre quelconque de sous-transactions ou d'opérations *lire* et *écrire*. La transaction entière forme donc une arborescence de sous-transactions, appelée *arbre de transaction* (voir Figure 5.1).

Nous utilisons la terminologie définie dans (Moss, 1985). La transaction racine, qui n'est imbriquée dans aucune autre, est appelée transaction globale **(TG)**. La TG organise seulement le contrôle et détermine a quel moment elle doit invoquer des sous-transactions. Les transactions qui n'ont pas de sous-transactions sont appelées des transactions feuilles **(TF)**. Les Transactions qui ont des sous-transactions sont appelées transactions parents **(TP)** et leurs sous-transactions sont appelées enfants **(TE)**. Les *supérieurs* d'une sous-transaction donnée incluent toutes les transactions dans le chemin de la sous-transaction à la racine mais n'incluent pas elle-même. *Les inférieurs* d'une transaction sont ses transactions dont chacune fait partie d'une hiérarchie de sous-transactions recouverte par la transaction, mais pas elle-même. *Les ancêtres* (resp. *les descendants*) d'une transaction sont les supérieurs (resp. les inférieurs) de la transaction, et la transaction elle-même (Voir Figure 5.1)

- T1 est la racine ou la TG,
- T2 et T3 sont les filles de T1,
- T2 et T3 sont des frères (sibling),
- T4 et T5 sont les filles de T2,
- T8, T9, T5, T6 et T7 sont des TF,
- Les ancêtres de T8 sont T1, T2, T4 et T8,
- Les supérieurs de T8 sont T1, T2 et T4,
- Les descendants de T2 sont T8, T9, T4, T5 et T2,
- Les inférieurs de T2 sont T8, T9, T4 et T5.

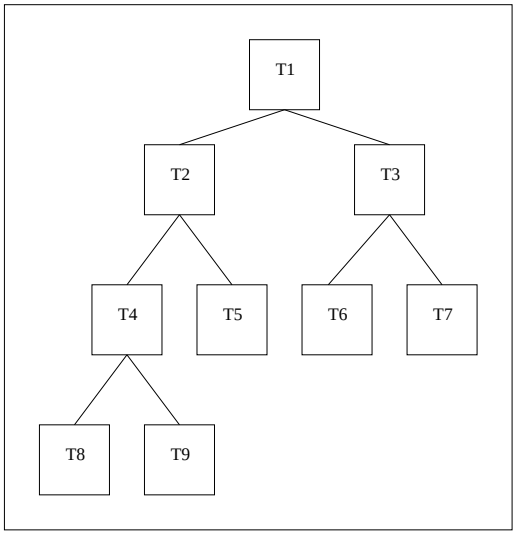

FIG. 5.1 – Exemple d'arbre de transaction imbriquée

.

Les principaux modèles de transactions imbriquées sont (Moss, 1985) (1) *transactions imbriquées fermées (closed nested transaction)* et (2) *transactions imbriquées ouvertes (open nested transaction)* (Madria et al., 1998).

Dans le modèle de transactions fermées, l'effet d'une sous-transaction ne peut être vu à l'extérieur de son parent, et sa validation est conditionnée par celle de son parent. Tandis que dans le modèle des transactions ouvertes (Madria et al., 1998), les sous-transactions peuvent s'exécuter et valider de façon indépendante.

Ce modèle fournit une exécution non-stricte en tenant compte de la sémantique (propriétés de commutativité) des opérations à chaque niveau d'abstraction de donnée. Une sous-transaction peut libérer les verrous avant la validation de la transaction globale. Les verrous sont libérés tôt seulement si la sémantique des opérations est connue. Dans beaucoup d'applications, la sémantique des opérations ne peut être connue et ainsi il est difficile de fournir une exécution non stricte. Dans nos travaux, nous avons considéré le modèle de transactions fermées.

Parmi les autres modèles de transactions imbriquées, il y a les modèles qui permettent à un parent de manipuler directement les données et ceux qui, au plus, permettent l'exécution concurrente avec les enfants (Harder and Rothermel, 1993). Dans les modèles de transactions imbriquées, les propriétés ACID sont respectées par la TG (Resende and Abbadi, 1994), tandis que seulement un sous ensemble de ces dernières sont respectées par les sous-transactions. La transaction n'est autorisée à valider qu'à la terminaison de toutes ses sous-transactions. Cependant, si une sous-transaction a été abandonnée, son parent n'est pas obligé d'abandonner. La TG peut choisir alors d'exécuter l'une des actions suivantes :

1. Ignorer la sous-transaction qui a été abandonnée,
2. Essayer de ré-exécuter la sous-transaction,
3. Lancer une autre sous-transaction (contingente ou de compensation),
4. Abandonner.

La validation d'une sous-transaction dépend du résultat (Validation ou Abandon) de son supérieur, même si elle est validée et que son supérieur abandonne. De plus, les mises à jour ne deviennent permanentes que si la TG valide. On distingue quatre règles de base pour les modèles de transactions imbriquées :

- **Terminaison** : une sous-transaction T_j, fille d'une (sous)-transaction T_i, démarre après T_i et se termine avant elle.
- **Validation** : la validation est relative, une mise à jour ne devient permanente que si la TG valide. Les résultats de la validation des sous-transactions sont accessibles à son parent.
- **Abandon** : l'abandon d'une sous-transaction entraîne l'abandon de ses sous-transactions descendantes, mais pas nécessairement l'abandon de ses ancêtres.
- **Visibilité** : toutes les mises à jour faites par la sous-transaction deviennent visibles à son parent après sa validation, en revanche, ses frères (sibling) ne verront pas ses résultats.

Dans les paragraphes suivants, nous décrivons le protocole de contrôle de concurrence que nous avons conçu pour les transactions imbriquées temps réel.

5.3 Notre contribution : protocole de contrôle de concurrence

5.3.1 Description du protocole

Dans le modèle de transactions imbriquées que nous considérons, nous supposons que les opérations élémentaires sont des lectures et des écritures qui peuvent être *essentielles* ou *non-essentielles*. Une opération essentielle doit se terminer avant l'échéance de la transaction, tandis qu'une opération non essentielle peut être omise si l'échéance de la transaction est imminente. Le protocole que nous proposons est donc basé sur le verrouillage différencié et offre 4 modes de synchronisation, correspondant à 4 modes de verrouillage :

- (**ER**-mode) : lecture **E**ssentielle,
- (**EW**-mode) : écriture **E**ssentielle,
- (**NER**-mode) : lecture **N**on-**E**ssentielle,
- (**NEW**-mode) : écriture **N**on-**E**ssentielle.

Les principales règles qui régissent le protocole sont les suivants, où le terme *transaction* désigne une branche quelconque d'une transaction imbriquée :

Règle 1 : Si T est une transaction essentielle, alors :

a) T peut acquérir un verrou en mode-ER sur une donnée d si :
   - aucune autre transaction ne détient un verrou en mode EW sur d, et
   - toutes les transactions, ayant verrouillé d en mode EW sont ses ancêtres.

b) T peut acquérir un verrou en mode EW si :
   - aucune autre transaction ne détient un verrou EW ou ER sur d, et
   - toutes les transactions détenant le verrou en mode EW ou ER sont ses ancêtres.

Règle 2 : Si T est une transaction non-essentielle, alors :

a) T peut acquérir un verrou en mode NER si :

- aucune autre transaction ne détient de verrou en mode EW ou NEW sur d, et
- toutes les transactions détenant le verrou en mode NEW sont ses ancêtres.

b) T peut acquérir un verrou en mode NEW si :

- aucune autre transaction ne détient de verrou en mode EW, NEW, ER ou NER, et
- toutes les transactions qui détiennent un verrou en mode NEW ou NER sont ses ancêtres.

Règle 3 : Quand T valide (commit), son parent hérite de ses verrous (acquis ou *retenus*). Ensuite, le parent de T "retient" le verrou dans le même mode que celui où T a été verrouillée précédemment.

Règle 4 : Quand T abandonne (abort), elle libère ses verrous (acquis ou *retenus*). Si une de ses transactions *supérieures* détient ou retient un de ses verrous, alors elle continue à les garder.

Le mode ER permet à plusieurs transactions de partager les données. De plus, une transaction essentielle et non essentielle pourraient acquérir un verrou en même temps. Si une transaction non-essentielle détient un verrou NEW sur d, tandis qu'une transaction essentielle demande à verrouiller d en mode ER, alors le conflit est résolu en faveur de la transaction essentielle. La transaction non-essentielle est alors abandonnée et redémarrée.

Si une transaction essentielle demande à verrouiller d en mode EW, alors tout conflit sur d avec des transactions non-essentielles est résolu en faveur de la transaction essentielle.

Des verrous *retenus* de type EW, ER, NEW ou NER indiquent que des transactions en dehors de la hiérarchie de la transaction qui retient le verrou, ne peuvent pas acquérir le verrou. Mais ses descendants le peuvent potentiellement, i.e. si une transaction T retient un verrou EW, alors aucun de ses descendants ne peut acquérir de verrou d'aucun type sur la même donnée. La table 5.1 montre la matrice de compatibilité des verrous entre les détenteurs (sur les colonnes) et les demandeurs (sur les lignes). Dans cette table, le terme *ancêtre* veut dire que sur la colonne correspondante, toutes les (sous-)transactions qui retiennent le verrou sont ses ancêtres, et le terme *non ancêtre* veut dire que sur la colonne correspondante, toutes les (sous-)transactions qui retiennent le verrou ne sont pas ses ancêtres. Le symbôle 'O' signifie que les verrous sont compatibles, 'N' qu'ils ne sont pas compatibles, '-' indique que la résolution du conflit se fait en faveur de la transaction essentielle.

Verrou demand´e \ Verrou d´etenu	Ancêtre ER	Ancêtre EW	Ancêtre NER	Ancêtre NEW	Non Anc. ER	Non Anc. EW	Non Anc. NER	Non Anc. NEW
ER	O	O	-	-	O	N	O	O
EW	O	O	-	-	N	N	O	O
NER	-	-	O	O	N	N	O	N
NEW	-	-	O	O	N	N	N	N

TAB. 5.1 – Matrice de compatibilité des verrous (ER, EW, NER et NEW)

5.3.2 Commentaires

On note que le protocole proposé présente quelques restrictions, comme par exemple le fait qu'il n'autorise pas la sérialisation entre les transactions essentielles et non essentielles. Cependant, malgré cet inconvénient, les résultats des simulations que nous avons effectuées ont montré que notre approche est meilleure comparée aux protocoles de base des transactions imbriquées et au protocole des transactions plates. En effet, elle améliore sensiblement le ratio de succès des transactions (Abdouli et al., 2004a).

Cette approche permet également d'adapter les propriétés $N\text{-}ACID$ aux transactions temps réel. Par exemple, l'atomicité est respectée seulement par la transaction TLT (de plus haut niveau), puisqu'il peut arriver qu'une transaction puisse effectuer sa validation (commit) sans que certaines de ses sous-transactions non essentielles soient terminées. Il peut arriver également qu'une sous-transaction ne respecte pas la propriété de N-durabilité. En effet, les effets d'une sous-transaction ne deviennent permanents que lorsque la transaction de plus haut niveau de l'arbre auquel elle appartient aura été validée.

5.3.3 Conclusion et travaux futurs

Nos futurs travaux sur les transactions temps réel imbriquées seront axés principalement sur la validation partielle des transactions, i.e. avec au moins toutes les sous-transactions essentielles qui valident. Nous sommes également en train d'explorer la manière de nous inspirer du protocole PROMPT de validation des transactions distribuées temps réel (Haritsa et al., 2000), où une sous-transaction en phase d'incertitude peut "prêter" ses données à d'autres (sous)-transactions.

Dans un contexte optimiste, i.e. en supposant que la sous-transaction prêteuse va finir par valider (commit), nous pensons que cette voie qui consiste à utiliser le prêt des données d'une part, et les transactions imbriquées avec partie essentielle et partie non essentielle d'autre part, devrait améliorer les performances des SGBDTR.

Un autre travail que nous envisageons est de nous baser sur le langage ACTA (Chrysanthis and Ramamritham, 1992) afin d'effectuer une étude plus formelle des protocoles de contrôle de concurrence des transactions imbriquées temps réel que nous développons, en particulier pour démontrer certaines propriétés.

Enfin, nous envisageons d'implémenter ce modèle dans un simulateur de SGBDTR que nous sommes en train de développer dans notre équipe de recherche (http ://www-lih.univ-lehavre.fr/ sadeg/recherche/recherche.html).

5.3.4 Encadrements, collaborations et diffusion des résultats

Ces travaux rentrent dans le cadre de la thèse du doctorant Majed Abdouli que je co-encadre et qui a commencé en Décembre 2003.

Les résultats de ces travaux sont publiés dans des conférences internationales (Abdouli et al., 2003b; Abdouli et al., 2004b) et nationales (Abdouli et al., 2003a).

En plus du doctorant, les autres participants à ces travaux sont L. Amanton, le Professeur A. Berred et moi-même de l'université du Havre, et le Professeur A. Alimi de l'université de Sfax (Tunisie).

6 Temps réel et systèmes multi-agents : vers des transactions *Anytime*

Nos travaux concernent la conception d'algorithmes Anytime pour l'interrogation de bases de données distribuées. Ces interrogations doivent pouvoir renvoyer des résultats, même partiels, dans les temps, et ces résultats doivent pouvoir être améliorés en fonction du temps imparti.

6.1 Généralités sur les algorithmes *Anytime*

Dans le but d'introduire les aspects temps réel dans les systèmes multi-agents, nous avons opté pour l'utilisation d'algorithmes *Anytime*, qui sont apparus vers la fin des années 80 dans les articles de (Dean and Boddy, 1988; Horvitz, 1990). La définition de base d'un algorithme Anytime peut être la suivante : *c'est un algorithme conçu de telle manière que la qualité du résultat qu'il fournit s'améliore au fur et à mesure qu'il dispose de temps d'exécution*. C'est à dire plus long est le temps d'exécution octroyé à l'algorithme, meilleurs seront les résultats qu'il fournit.

On distingue généralement deux catégories :

- Les algorithmes Anytime par contrat : ils nécessitent d'avoir une connaissance préalable du temps alloué pour construire un résultat cohérent.
- Les algorithmes Anytime interruptibles : ils constituent l'idéal des algorithmes Anytime. Ce sont des algorithmes acceptant d'être interrompus *à tout moment* de leur exécution et fournir un résultat exploitable. Nos travaux concernent cette deuxième catégorie d'algorithmes, que nous désignerons simplement dans la suite par le terme *Anytime*.

Les algorithmes Anytime possèdent certaines propriétés, qui sont :

- La qualité du résultat dépend du temps alloué, mais également des ressources disponibles en entrée.
- Des mécanismes doivent permettre de mesurer la qualité du résultat.
- Ces algorithmes doivent être "prévisibles", i.e. disposer d'informations statistiques sur la qualité de leurs résultats en fonction du temps et des ressources.
- Ils doivent être interruptibles, i.e. fournir un résultat exploitable à tout moment si on les interrompt.
- Ils doivent être monotones, i.e. la qualité du résultat qu'ils fournissent augmente ou reste stable au cours du temps.

Des travaux sur l'application des algorithmes Anytime existent et concernent notamment le contrôle de robots mobiles (Zilberstein and Russell, 1993). Zilberstein et Russell ont appliqué dans (Zilberstein and Russell, 1996) les algorithmes Anytime pour la composition de systèmes temps réel. Dans (Grass

and Zilberstein, 1995), Grass et Zilberstein ont donné des méthodes pour programmer avec les algorithmes Anytime. Dans (Mouaddib, 1997), Mouaddib a proposé l'utilisation d'algorithmes Anytime pour la négociation entre agents autonomes dans un environnement contraint par le temps.

6.2 Notre contribution

6.2.1 Les algorithmes *Anytime* pour l'aide à la décision

Les travaux de recherche abordés dans cette partie sont effectués dans le cadre d'une thèse cofinancée par la région Haute-Normandie, (Contrat de plan État-Région). Ils ont pour thème central : le temps réel appliqué à des systèmes multi-agents. Les problèmes adressés sont relatifs aux Systèmes Multi-Agents temps réel qui permettent de traiter des problèmes faisant intervenir des situations complexes et difficiles à traiter avec des modèles classiques et centralisés. Bien que le domaine des systèmes multi-agents soit traité de façon importante par la communauté scientifique, très peu de travaux concernant l'extension de ce domaine et faisant le lien avec le domaine du temps réel ont été effectués. Dans nos travaux, nous avons élaboré un algorithme *Anytime* et nous avons testé ses performances sur un prototype que nous avons développé. Des détails sur les résultats de recherches sont présentés dans (Duvallet et al., 1999d), (Duvallet, 2001) et (Duvallet and Sadeg, 2004).

Les algorithmes Anytime et les techniques qui leur sont associées s'avèrent particulièrement intéressants pour les applications devant fournir des réponses et des informations à des décideurs humains. En effet, il arrive souvent qu'une réponse partielle obtenue dans les temps (avant que la décision ne soit prise) soit préférable à une réponse plus complète mais qui arriverait trop tard, et donc inutilisable pour le décideur.

Le thème sur lequel nous avons travaillé concerne les interrogations et traitements effectués en temps réel sur des Systèmes d'Information distants, i.e. Système de Gestion de Bases de Données. L'extraction d'informations depuis des systèmes d'information distribués doit être effectuée en temps réel d'où la notion de SGBD temps réel que l'on aborde principalement du point de vue interrogation des données, mais qui pourrait être étendue de façon plus large aux opérations habituellement effectuées dans les SGBD (Duvallet et al., 1999b).

Nous avons conçu un prototype de système multi-agents *Anytime* relatif à une application de gestion de marché électronique que nous allons décrire dans la suite. Nous décrirons également une application de gestion de logistique portuaire que notre système pourrait gérer.

– Application 1 : gestion de plate-forme portuaire

  Sur une plate-forme portuaire, il faut pouvoir décharger en un minimum de temps un navire, mais aussi placer l'ensemble des conteneurs à des endroits libres de la plate-forme. Il est donc important de disposer d'un système informatique donnant les emplacements libres dans un délai minimal. Il arrive également que deux ou plusieurs navires doivent être déchargés en même temps et que plusieurs terminaux donnent accès au système informatique chargé de gérer les emplacements libres. Si deux personnes sur deux terminaux différents cherchent à placer en même temps leurs conteneurs et si le système leur alloue le même emplacement libre, alors survient un conflit d'accès aux ressources et il est nécessaire d'allouer un nouvel emplacement pour l'un des deux conteneurs, d'où une perte de temps. Il devrait donc être tenu compte dans le système informatique de la synchronisation des demandes d'emplacements libres. Nous avons illustré notre propos sur les systèmes temps réel par un exemple simple, mais les problèmes qui

se posent dans la réalité de tels systèmes informatiques sont bien souvent beaucoup plus complexes et font intervenir des techniques d'au moins trois domaines : la logistique, les bases de données et le temps réel.

– Application 2 : gestion de marché électronique

Le commerce électronique fait partie des domaines d'applications où les concepts et méthodes issus des nouvelles technologies sont et seront de plus en plus utilisés. Un système d'aide à la gestion de marché électronique doit permettre d'effectuer des transactions d'achat et de vente et d'opérer une surveillance de l'état du marché en temps réel pour des groupes d'utilisateurs géographiquement distants.

Le système de gestion de marché effectue l'ensemble de ses opérations en utilisant des Systèmes d'Information (SI) partagés et/ou privés. Certains de ces SI sont mis à jour régulièrement afin de rendre compte de l'état du marché en temps réel à chaque utilisateur, alors que d'autres systèmes sont sujets à des modifications très peu fréquentes. Ils peuvent aussi être dédiés à un utilisateur ou à un groupe d'utilisateurs. Lorsqu'un utilisateur interroge le système, il doit pouvoir interagir avec celui-ci selon plusieurs niveaux d'urgence des demandes effectuées, mais aussi avec différents niveaux de précision dans la qualité du résultat escompté.

L'interrogation des SI se fait au moyen d'un script (Duvallet and Sadeg, 2004) selon un schéma préétabli au moyen d'ontologies qui permettent d'établir la connaissance sur le domaine. Une ontologie particulière est définie afin de représenter et catégoriser le domaine dans lequel opérera le système. Nous traitons plus particulièrement une application dans le domaine de la vente de produits alimentaires. Cette partie du système peut être vue comme un module et pourra être interchangée avec d'autres modules selon le domaine dans lequel on souhaite opérer, par exemple, entrer dans la composition d'un système de logistique (portuaire, ...).

6.2.2 Interrogation en temps réel des systèmes d'information

Il est nécessaire de contrôler les temps d'exploration des systèmes d'information lorsqu'il s'agit d'aider le décideur dans sa prise de décision. Nous utilisons pour cela une approche qui consiste à concevoir un modèle logique du système d'information facile à implémenter sur machine. Ce modèle permet de regrouper à la fois les avantages des SMA (Système Multi-Agents), des techniques *Anytime* et des systèmes distribués. De cette manière, nous pouvons envisager une exploration à différents niveaux de profondeur ou de précision des systèmes d'information situés sur différents sites, en fonction du temps alloué pour cette exploration. L'approche *Anytime* nous semble la plus adéquate pour ce type d'applications (Grass and Zilberstein, 1995).

Dans un contexte d'applications distribuées, les interrogations doivent être construites de façon à pouvoir extraire de l'information depuis des bases de données distribuées et de fusionner ces informations extraites qui sont souvent hétérogènes et parfois contradictoires.

6.2.3 Un modèle de système multi-agents temps réel : ANYMAS

Un SMA est composé par nature de multiples entités coopérantes, appelées agents. Les agents d'un SMA interagissent et coopèrent pour résoudre des problèmes complexes. Le schéma d'interaction pour la résolution d'un problème particulier est imprévisible : on ne peut pas connaître à l'avance le nombre d'agents qui vont intervenir, quels vont être leurs types, le nombre de fois qu'ils vont communiquer et s'échanger des messages. Par conséquent, il apparaît très difficile de prévoir les temps d'exécution nécessaires à la résolution d'un problème. Donc, l'approche basée sur les ordonnancements temps réel

"classiques" (Cottet et al., 2000; Duvallet et al., 1999b) où chaque tâche/transaction est tenue de respecter son échéance, s'avère inexploitable. Nos travaux présentent une nouvelle approche pour prendre en compte l'aspect temps réel dans un SMA : le modèle ANYMAS, un modèle de système multi-agents basé sur l'approche *Anytime*.

Dans le modèle ANYMAS, un ATN (Augmented Transition Network) (Woods, 1970) est utilisé pour stabiliser les agents dans différents états. Cette notion d'ATN est héritée de l'architecture d'agents utilisée dans la plate-forme DIMA (Guessoum, 1997), mais aussi dans d'autres architectures d'agents. Nous considérons deux modes d'exécution : le mode d'apprentissage et le mode de fonctionnement normal. Durant le mode d'apprentissage, la vitesse de l'agent est constamment recalculée afin d'obtenir une valeur moyenne de la vitesse d'exécution. En mode réel d'exécution, cette valeur est utilisée afin d'estimer le temps nécessaire pour exécuter la prochaine tâche. Ces différentes valeurs moyennes mesurées lors du passage d'un état à l'autre de l'ATN sont alors discrétisées en une unité de temps que l'on appelle ΔT, à la fin du mode d'apprentissage.

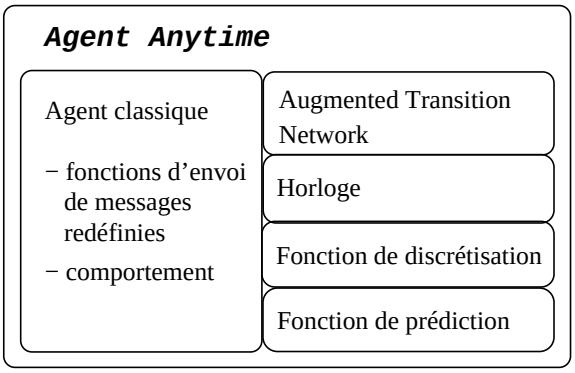

FIG. 6.1 – Agent *Anytime*

Pour le fonctionnement d'un agent *Anytime* (voir figure 6.1), les composants suivants sont utilisés :

- un ATN, qui permet de contrôler le comportement d'un agent durant sa phase active,
- une horloge temps réel, qui mesure le temps écoulé entre chaque état franchi de l'ATN,
- un module de réification[1] des agents de coordination temporelle (Duvallet and Sadeg, 2004),
- une fonction de discrétisation du temps écoulé entre deux états de l'ATN,
- une fonction de prédiction du temps nécessaire au franchissement d'un état dans l'ATN à partir de l'état courant. Elle est appelée "fonction de prédiction temporelle".

Nous présentons brièvement l'algorithme de la fonction de discrétisation du temps. Pour les détails et la description des autres composants, se référer à (Duvallet and Sadeg, 2004).

L'objectif de cette fonction est de trouver une unité de temps qui puisse servir à mesurer le temps écoulé pour franchir deux états de l'ATN et qui soit plus grande que l'unité de temps de base (qui est en général la milliseconde). Cette unité de temps est ensuite utilisée dans la fonction de prédiction temporelle afin d'effectuer des estimations sur les temps nécessaires au franchissement des états.

La discrétisation s'effectue à partir des informations temporelles stockées sur les transitions de l'ATN. Pour expliquer le fonctionnement du système, nous allons nous fonder sur un ATN simplifié (linéaire). Cette discrétisation va se faire de la façon suivante :

[1] constitution par regroupement

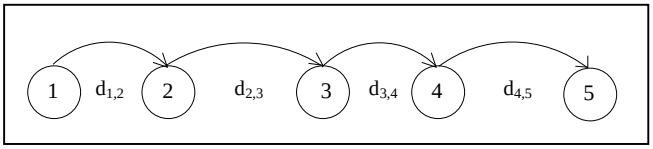

FIG. 6.2 – Exemple d'ATN linéaire

Étant donné un ATN avec n états (par exemple l'ATN schématisé dans la figure 6.2, où n est égal à 5), $d_{i,i+1}$ ($1 \leq i < n-1$) est la transition entre deux états i et i+1 ; Soit une variable ΔT représentant une sous-division du temps et $n_{i,i+1}$ le nombre de ΔT sous-divisions entre deux états i and $i+1$.

Alors, on obtient l'équation suivante :

$d_{i,i+1}$ = ΔT * $n_{i,i+1}$, où i=1,2,3,... $\Rightarrow$ ΔT = $\frac{d_{i,i+1}}{n_{i,i+1}}$

Pour l'exemple de la figure 6.2, nous obtenons :

$\Delta T = \frac{d_{1,2}}{n_{1,2}} = \frac{d_{2,3}}{n_{2,3}} = \frac{d_{3,4}}{n_{3,4}} = \frac{d_{4,5}}{n_{4,5}}$

Le nombre de sous-divisions pour le premier intervalle est initialement fixé. Puis un algorithme (voir figure 6.3) nous permet de déterminer (1) la valeur de la variable ΔT (unité de temps sur l'ATN) et (2) le nombre de divisions entre deux états de l'ATN. La variable ε permet de fixer la précision du calcul effectué pour déterminer la valeur de ΔT.

En entrée : $d_{i,i+1}$ le temps écoulé entre deux états de l'ATN
N le nombre d'états dans l'ATN
En sortie : ΔT la nouvelle unité de temps calculée
$n_{i,i+1}$ le nombre de ΔT entre les états i et i+1
L'algorithme :
$n_{1,2} \leftarrow 1$
TANT QUE ((| $d_{i,i+1}$ - ΔT * $n_{i,i+1}$ | > ε)
ET ((i+1) $\neq$ Dernier Etat))
i $\leftarrow$ 1
$n_{1,2} \leftarrow n_{1,2}$ + 1
ΔT = arrondi ($d_{i,i+1}$ / $n_{i,i+1}$)
TANT QUE (|$d_{i,i+1}$ - ΔT * $n_{i,i+1}$| < ε
ET (i+1) $\neq$ Dernier Etat))
i $\leftarrow$ i + 1
$d_{i,i+1}$ = $d_{i,i+1}$ + $d_{i-1,i}$ - ΔT * $n_{i-1,i}$
$n_{i,i+1}$ = arrondi ($d_{i,i+1}$ / ΔT)
FIN TANT QUE
FIN TANT QUE

FIG. 6.3 – Algorithme de discrétisation du temps sur un ATN linéaire

Après application de l'algorithme, nous obtenons l'ATN représenté sur la figure 6.4.

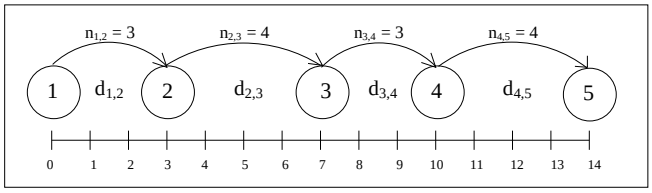

FIG. 6.4 – Exemple de discrétisation du temps sur un ATN linéaire

6.2.4 Réalisation d'un prototype

Ce système est réalisé en langage Java. Pour la conception des systèmes d'information (SI), nous avons utilisé les services offerts par le SGBD Oracle. Les SI sont matérialisés sous forme de bases de données relationnelles Oracle. Pour la partie multi-agents, nous avons utilisé la plate-forme de développement MadKit (Gutknecht and Ferber, 2000), étendue par des algorithmes *Anytime* (modèle ANYMAS) et par des composants relatifs à la gestion de l'aspect distribué (Duvallet, 2001).

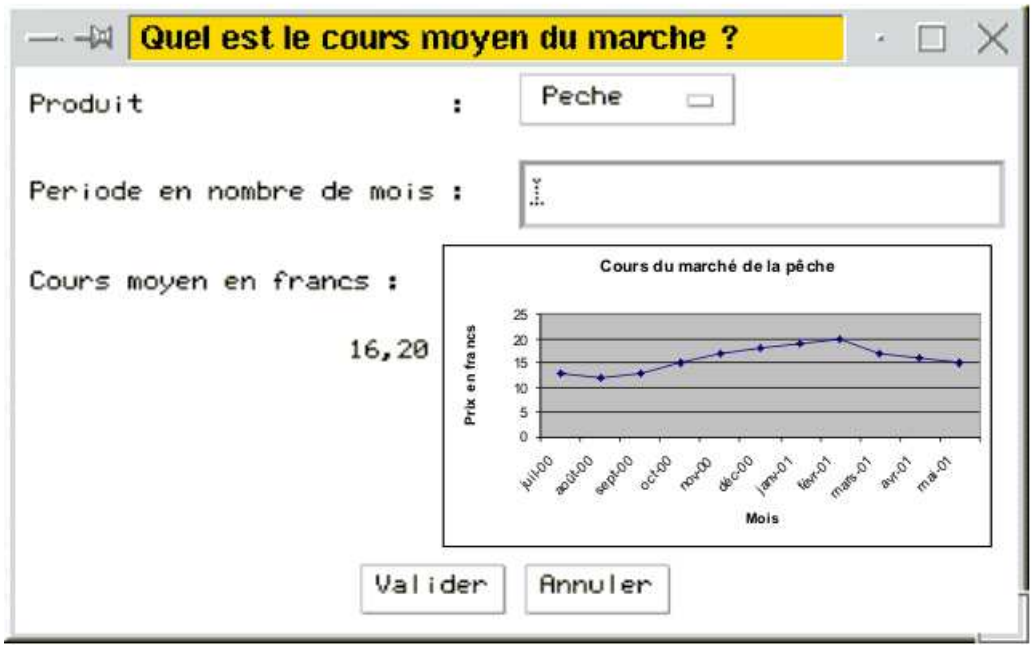

FIG. 6.5 – Modélisation du client dans une application de gestion de marché

6.2.5 Simulation et résultats

Le résultat de nos travaux a été l'élaboration d'une plate-forme distribuée, composée d'agents Anytime. L'application utilisée est une application de gestion de marché électronique, dans laquelle sont implémentés nos algorithmes (Duvallet and Sadeg, 2004). Le prototype réalisé est relatif à la gestion de produits alimentaires et un exemple de dialogue est illustré par la figure 6.5.

Les fonctionnalités principales considérées dans le système concernent l'interrogation des systèmes d'information afin de répondre aux requêtes des utilisateurs. Les réponses ainsi obtenues leur permettent de prendre la meilleure décision possible concernant des choix à effectuer (par exemple passer une commande à un fournisseur suivant différents critères : prix pratiqués, délais, respect des délais, etc.).

Une même question permet à un utilisateur d'obtenir une réponse plus ou moins précise suivant le temps dont il dispose pour prendre une décision.

6.2.6 Perspectives : les transactions *Anytime*

Après avoir travaillé sur les techniques Anytime appliquées aux agents dans les SMA[2], nous sommes en train de voir dans quelle mesure ces techniques peuvent s'appliquer à des entités moins abstraites et de granularité plus fine que les agents : ce sont les transactions temps réel. Nous sommes convaincus que l'utilisation de ces techniques sera d'un grand apport pour les transactions dans un contexte temps réel, notamment distribué. Une transaction distribuée Anytime[3], pourrait s'exécuter de manière "intelligente" : des paliers de résultats seront définis qui fournissent à chaque fois un résultat meilleur que le précédent, au fur et à mesure que la transaction dispose de plus de temps pour s'exécuter. Pour une transaction temps réel, il faut se baser sur la sémantique de la transaction pour définir quelles sont les sous-transactions et/ou les opérations[4] qui peuvent être omises en cas de risque de manquement d'échéances.

6.2.7 Encadrements, collaborations et diffusion des résultats

Ces travaux rentrent dans le cadre de la thèse de C. Duvallet. Le travail de thèse, dont je suis l'un des co-encadrants, a donné lieu a plusieurs publications dans des conférences internationales (Duvallet et al., 1999a; Duvallet et al., 1999c; Duvallet et al., 1999d; Duvallet et al., 2000a; Duvallet et al., 2000b; Duvallet and Sadeg, 2001) et dans des revues nationales (Duvallet et al., 1999b; Duvallet and Sadeg, 2004). L'autre co-encadrant de la thèse de M. C. Duvallet est le Professeur A. Cardon de l'université du Havre. La thèse a été soutenue en octobre 2001.

[2] Syst`emes Multi-Agents
[3] Transaction temps r´eel dans laquelle est impl´ement´e un algorithme Anytime
[4] Lecture ou Écriture

7 Ordonnancement par rétroaction

7.1 Introduction

Les algorithmes d'ordonnancement temps réel appartiennent à la catégorie des ordonnancements statiques ou à celle des ordonnancements dynamiques. Dans l'ordonnancement statique, l'algorithme dispose d'informations complètes sur les tâches à ordonnancer comme les échéances, le temps d'exécution, les contraintes de précédence et les dates de réveil. Un exemple de ces algorithmes est *Rate Monotonic* (Liu and Leyland, 1973) et ses extensions.

Dans l'ordonnancement dynamique, par contre, l'algorithme ne dispose que d'informations partielles sur les tâches. Par exemple, de nouvelles tâches, inconnues de l'algorithme quand il ordonnance la tâche courante, peuvent survenir à n'importe quel moment. Les algorithmes d'ordonnancement dynamique se subdivisent en deux catégories : ceux qui travaillent dans des environnements avec *ressources suffisantes* et ceux qui travaillent dans des environnements avec *ressources insuffisantes*. Les environnements avec *ressources suffisantes* sont des systèmes où même si les tâches arrivent de manière dynamique, l'algorithme offre des garanties a priori qu'à tout moment les tâches seront ordonnançables. Sous certaines conditions, l'algorithme EDF (Liu and Leyland, 1973) est optimal parmi les ordonnancements dynamiques dans des environnements avec ressources suffisantes.

Bien que que les concepteurs de systèmes temps réel tentent de concevoir des systèmes avec ressources suffisantes, il devient parfois impossible de les garantir, à cause des problèmes de coût ou d'environnements imprévisibles. Dans ce cas, les performances de l'algorithme EDF par exemple décroissent très rapidement.

Il existe des algorithmes qui fonctionnent relativement bien dans des environnements avec ressources insuffisantes. Ces algorithmes sont basés sur des politiques de contrôle d'admission adéquates, i.e. elles filtrent les tâches à accepter dans le système pour exécution en fonction de la charge du système à un instant donné. On peut citer l'algorithme d'ordonnancement Spring (Zhao et al., 1987).

Malgré des résultats significatifs obtenus avec ces approches, il existe encore beaucoup de problèmes du monde réel que ces algorithmes ne résolvent pas de manière satisfaisante.

Tous les algorithmes cités précédemment fonctionnent en "boucle ouverte". Ce fonctionnement fait référence au fait qu'une fois les séquences d'ordonnancement établies, on ne peut plus les "ajuster" en fonction de l'état réel du système. Ces algorithmes sont efficaces dans des environnements prévisibles. Ils le sont beaucoup moins dans des environnement imprévisibles, i.e. dont la charge ne peut pas être modélisée de façon précise.

Pour remédier à cette situation, Lu et al. ont proposé dans (Lu et al., 2002), une approche utilisant le contrôle par rétroaction pour l'ordonnancement des tâches temps réel. Récemment, des travaux se basant sur ce cadre ont été effectués (Amirijoo et al., 2003a; Amirijoo et al., 2003b) pour ordonnancer

les transactions dans les SGBD temps réel. Nos recherches s'inscrivent dans la continuité de ces travaux.

7.2 SGBDTR basé sur la rétroaction

Dans une application reposant sur l'utilisation de SGBD temps réel, des transactions en provenance des utilisateurs arrivent à des fréquences variables. Lorsque la fréquence augmente de façon considérable, l'équilibre du SGBDTR est mis en péril. Durant ces périodes de surcharge, le SGBDTR va potentiellement manquer de ressources, et un grand nombre de transactions temps réel risquent alors de manquer leurs échéances. Des travaux basés sur une approche «Qualité de Service» (QdS) (Kang et al., 2002; Amirijoo et al., 2003a) tentent de rendre les SGBDTR plus robustes face aux périodes d'instabilité (périodes de sous-utilisation et périodes de surcharge). Ces travaux s'appuient sur des techniques de contrôle avec rétroaction (Lu et al., 2001) et autorisent la manipulation de résultats imprécis (Liu et al., 1991). Dans cette section, nous allons détailler ces travaux sur lesquels s'appuie notre proposition.

Nous allons présenter plus particulièrement le modèle de données et de transactions que nous considérons.

Nous considérons des données temps réel, qui représentent l'état de l'environnement. Elles sont mises à jour périodiquement. Nous considérons également des données non temps réel, i.e. que l'on trouve dans toute base de données.

Nous considérons deux catégories de transactions :

- Les transactions de mise à jour : elles ont pour tâche de mettre à jour régulièrement les données temps réel. Elles sont exécutées périodiquement pour rafraîchir la valeur des données temps réel.
- Les transactions «utilisateur» : elles effectuent des opérations de lecture/écriture de données non temps réel et/ou des lectures de données temps réel. La non prévisibilité de leurs arrivées dans le système et de la charge qu'elles suscitent rend adéquate l'utilisation d'une architecture basée sur le retour d'expérience (ou rétroaction) pour gérer les variations importantes de charge (Lu et al., 2002).

Dans les premiers travaux que nous avons effectués, nous avons considéré qu'une transaction de mise à jour crée une nouvelle version d'une donnée lorsqu'elle souhaite écrire dans la base de données. La gestion des versions est décrite dans la suite. Les transactions «utilisateur» accèdent à la donnée la plus récente qui n'est pas verrouillée.

7.3 Modèle général

Le modèle général sur lequel nous avons fondé nos travaux est illustré par la figure 7.1. Nous allons présenter dans cette section les parties du modèle qui nous intéressent pour nos travaux.

- Le contrôleur d'admission (Admission control) : il a pour tâche de contrôler les transactions utilisateur qui sont acceptées dans le système. Il effectue ce contrôle en fonction de la charge d'utilisation calculée et des paramètres de qualité de service spécifiés par l'administrateur de la base. Son fonctionnement est contrôlé par la boucle de rétroaction qui lui fournit ses paramètres de fonctionnement.

- Les transactions qui sont admises dans le système sont placées dans une file d'attente avant d'être envoyées au déclencheur de transactions (Transaction handler), qui a pour fonction de

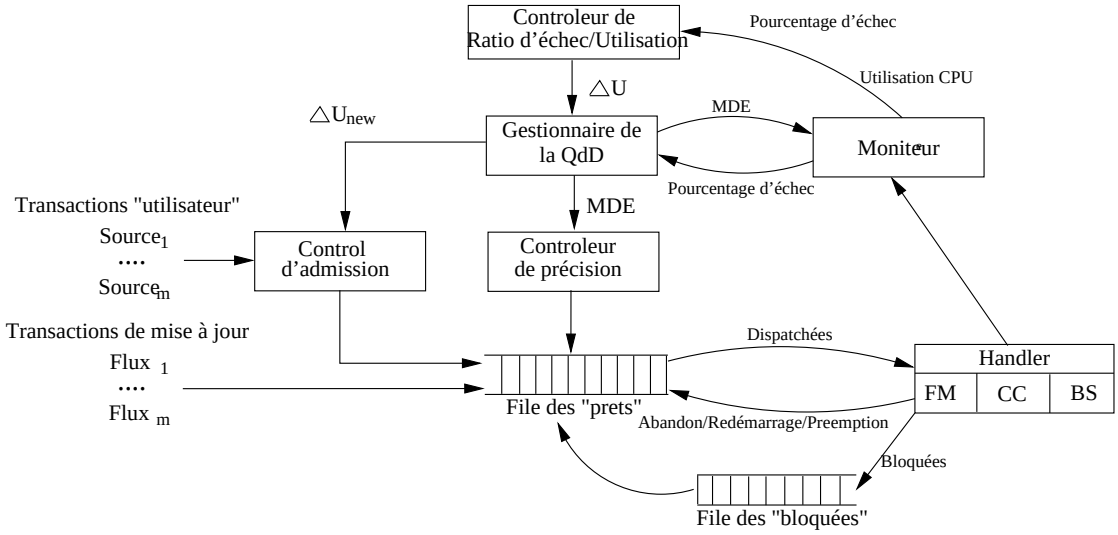

FIG. 7.1 – Architecture de base du modèle avec rétroaction

gérer l'exécution des transactions. Il dispose de plusieurs modules complémentaires :

- Un gestionnaire de fraîcheur (FM) : il vérifie la fraîcheur des données qui vont être accédées par une transaction. Si les données sont obsolètes (non fraîches) alors la transaction est mise en attente dans une file.

- Un contrôleur de concurrence (CC) : il est chargé de gérer les conflits d'accès aux données qui apparaissent entre les transactions. Dans la plupart des travaux, il s'agit du protocole 2PL-HP où la transaction de plus haute priorité est privilégiée.

- Un ordonnanceur de base (BS) : il s'agit bien souvent du protocole Earliest Deadline First (EDF) qui ordonnance les transactions en considérant que la transaction qui possède l'échéance la plus proche doit être exécutée en priorité.

Un moniteur permet de mesurer les performances du système en inspectant l'exécution des transactions (quantité de transactions terminées, abandonnées, qui ont raté leurs échéances, etc). Les valeurs ainsi mesurées permettant d'alimenter le contrôleur de qualité de service et font partie de la boucle de contrôle par rétroaction qui va contribuer à stabiliser le système.

Le contrôleur d'utilisation et d'échéances ratées (ou contrôleur de qualité de service) permet de réajuster les paramètres de QdS en fonction des valeurs déterminées par le moniteur et des paramètres de référence. Les valeurs ainsi obtenues sont transmises au contrôleur d'admission et au gestionnaire de qualité des données. Le gestionnaire de qualité des données permet de réajuster la valeur du paramètre MDE (maximum d'erreur tolérée sur la donnée) qui constitue le paramètre de QdD (qui fixe la qualité des données). La valeur de MDE est calculée en fonction de l'utilisation du système. Le paramètre MDE est ensuite fourni au contrôleur de précision qui écarte les transactions de mise à jour lorsque les données à mettre à jour sont suffisamment représentatives du monde réel en considérant la valeur de MDE.

7.4 Boucle de contrôle par rétroaction

La boucle de rétroaction a pour tâche de stabiliser le système durant les phases de surcharge. Pour cela, elle s'appuie sur le principe d'observation puis d'auto-adaptation. L'auto-adaptation a lieu tout au long du fonctionnement du système car les demandes des utilisateurs sont imprévisibles et la charge doit être ajustée en permanence. L'observation consiste à prendre en compte l'état de fonctionnement du système et à déterminer s'il correspond aux paramètres de qualité de service initialement spécifiés. Cette observation se fait via le moniteur. A partir de l'observation effectuée, le système adapte ses paramètres, via le contrôleur de qualité de service, afin d'augmenter ou de diminuer le nombre de transactions acceptées dans le système. Cette adaptation va provoquer une modification du comportement du système et donc des valeurs observées par le moniteur. Le fonctionnement de cette boucle doit tendre vers une stabilité du système autour d'une valeur de référence, fixée par l'administrateur de la base (par exemple, il peut s'agir d'un taux d'utilisation de 80%). Il s'agit donc de réduire l'oscillation du système autour de cette valeur de référence.

7.5 Protocoles de contrôle de concurrence

Dans le cadre des applications temps réel, les données sensorielles (données temps réel acquises par des capteurs) sont mises à jour périodiquement par des transactions dédiées appelées «transactions de mise à jour» et peuvent être utilisées par des transactions «utilisateur». Lorsqu'une transaction souhaite modifier une donnée, déjà accédée en lecture par une autre transaction, on doit résoudre un problème de conflit d'accès au moyen d'un protocole de contrôle de concurrence. Il en est de même si une transaction essaie de lire une donnée verrouillée en écriture par une autre. Considérons alors ces deux cas :

- Dans le premier cas, on compare les priorités des deux transactions et on abandonne la transaction de plus faible priorité, puis on la redémarre. Il s'agit du protocole 2PL-HP (Abbot and Garcia-Molina, 1988).

- Dans le second cas, on peut :
  soit procéder de la même façon,
  soit suspendre la transaction qui cherche à lire la donnée.

La suspension ou l'abandon puis le redémarrage de certaines transactions augmentent de façon considérable le risque qu'elles ratent leurs échéances. C'est pourquoi nous proposons une nouvelle approche pour réduire le nombre de conflits entre les transactions temps réel et diminuer ainsi le nombre de transactions qui ratent leurs échéances. L'idée est d'ajuster le nombre de transactions admises dans le système en fonction de son état courant (paramètres de qualité de service), ainsi que de la marge d'erreur tolérée.

7.6 Notre contribution : Protocole MVS-FCSA

7.6.1 Architecture proposée

MVS-FCSA est une architecture basée sur la gestion de données multi-versions. Il s'agit d'une adaptation de l'architecture de contrôle de l'ordonnancement temps réel avec rétroaction décrite dans (Lu et al., 2002). Elle est basée sur une approche de spécification de la qualité de service dans les SGBDTR (Kang et al., 2002; Amirijoo et al., 2003a).

Dans notre système, on étend l'architecture du contrôle d'ordonnancement par rétroaction en utilisant la notion de données multi-versions pour garantir la qualité des données (fraîcheur et précision), et

pour réduire le nombre de conflit d'accès. Par conséquent, le taux de manquement des échéances par les transactions devrait être minimisé.

Comme le montre la figure 7.2, nous avons notamment modifié la partie concernant la gestion des transactions (file d'attente des transactions prêtes et déclencheur de transactions) et nous avons conservé la partie concernant la circulation des informations de contrôle (taux d'échéances ratées, taux d'utilisation, etc.). Notre architecture se compose d'une file d'attente, d'un ordonnanceur, d'un gestionnaire de données temps réel multi-versions, d'un vérificateur d'échéances, d'un gestionnaire de fraîcheur des données et d'un déclencheur de transactions.

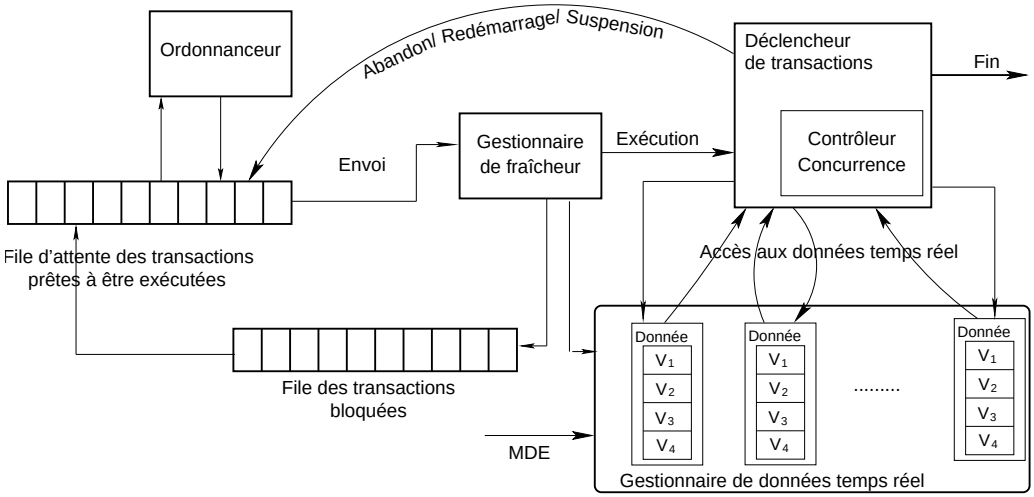

FIG. 7.2 – Architecture proposée pour le protocole MVS-FCSA

Dans la file des transactions *prêtes*, les transactions sont ordonnancées suivant la proximité de leurs échéances selon l'algorithme EDF. Ensuite, ces transactions vont transiter par deux composants avant d'être exécutées :

- Un vérificateur d'échéances qui contrôle que les transactions vont pouvoir être exécutées avant leurs échéances sinon elles sont immédiatement écartées (abandonnées).
- Un gestionnaire de fraîcheur qui vérifie la fraîcheur des données qui vont être accédées par la transaction. Si les données sont fraîches, la transaction peut être exécutée et elle est envoyée au déclencheur de transactions. Sinon elle est mise en attente dans une file des transactions bloquées. Elle est ensuite réinsérée, dès que les données auront été mises à jour, dans la file des transactions prêtes.

7.6.2 Principe de la gestion des données temps réel

Nous considérons que seules les transactions de mise à jour accèdent aux données temps réel en mode écriture. La gestion des données doit être effectuée de façon à garantir la fraîcheur des données d'une part, et à respecter le seuil d'erreur tolérée MDE sur la donnée d'autre part.

Lorsqu'une transaction de mise à jour souhaite écrire une donnée temps réel, une nouvelle version est créée. Lorsqu'une version d'une donnée ne respecte pas les paramètres de qualité des données, elle est supprimée de la base. Afin d'éviter le stockage de versions inutiles de données, celles qui ne seront pas accédées seront supprimées à la condition qu'il ne s'agisse pas de l'unique version accessible, i.e. il

existe au moins une autre version de cette donnée respectant les paramètres de la qualité des données et qui n'est pas verrouillée en écriture.

7.6.3 Commentaires et perspectives

Les travaux que l'on poursuit actuellement consistent à tester l'architecture que nous avons proposée d'une part, et de considérer plusieurs cas pour les versions des données d'autre part. Les choix du nombre de versions peuvent être : (1) un nombre fixe pour toutes les données, (1) un nombre différent pour chaque donnée. Ce nombre peut être fixe ou ajusté dynamiquement en fonction des performances obtenues dans la résolution des conflits.

7.6.4 Encadrements, collaborations et résultats préliminaires

Nos travaux sur l'ordonnancement par rétroaction rentrent dans le cadre de la thèse de la doctorante E. Bouazizi, commencée cette année (2003-2004) et que je co-encadre. Les résultats préliminaires sur le protocole MVS-FCSA que nous avons proposé sont publiés dans la conférence nationale MajecSTIC'04 (Bouazizi et al., 2004a).

En plus de la doctorante E. Bouazizi, les participants à ces travaux sont B. Sadeg et C. Duvallet, maîtres de conférences, co-encadrants de la thèse.

8 Étude probabiliste du comportement des transactions temps réel

8.1 Introduction

La plupart des travaux sur les systèmes temps réel sont relatifs à des systèmes déterministes, c'est-à-dire que l'on considère que les tâches temps réel possèdent des propriétés connues à l'avance. Il en est de même des caractéristiques du processeur et d'autres ressources accédées par les tâches. Ces éléments permettent d'évaluer les WCET[1] des tâches et donc d'effectuer des analyses déterministes d'ordonnancement de ces tâches.

Cependant, les hypothèses requises pour travailler avec le WCET deviennent de plus en plus difficiles à respecter dans les applications actuelles devenues de plus en plus complexes et sophistiquées. Depuis quelques années, des travaux sont apparus qui étudient le comportement des tâches temps réel en se servant des outils fournis par les probabilités et statistiques.

Les travaux sur les aspects probabilistes existants concernent les aspects probabilistes des tâches dans les systèmes temps réel (Atlas and Bestavros, 1998; Tia et al., 1995; Sha et al., 1994; David et al., 2003; Abeni and Buttazzo, 1999). C'est ainsi que dans (Atlas and Bestavros, 1998), des travaux ont été effectués par Atlas et Bestavros sur la manière d'effectuer une étude statistique de l'algorithme d'ordonnancement *Rate Monotonic*. D'autres recherches se sont focalisées sur l'analyse probabiliste des performances des systèmes temps réel (Tia et al., 1995). D'autres études ont été effectuées par Sha et al dans (Sha et al., 1994), qui concernent un cadre pour l'analyse de la politique d'ordonnancement Rate Monotonic généralisée. D'autres recherches ont été effectuées sur l'étude probabiliste des tâches temps réel périodiques avec temps d'exécution indéterminé, notamment le paramètre lié à la gigue (David et al., 2003). Dans (Abeni and Buttazzo, 1999), Abeni et buttazzo ont étudié la qualité de service dans les applications temps réel en utilisant des tâches à échéances probabilistes.

D'autre part, les aspects probabilistes des transactions dans les SGBD traditionnels (conflits d'accès, moyenne des temps de réponse, ...) concernent les performances des systèmes en termes de temps de réponse moyen des transactions ou, comme dans (Singhal and Smith, 1994), Singhal et al. ont travaillé sur l'acquisition probabiliste de verrous par les transactions de quelques SGBD relationnels commerciaux. Weinberger et Mitra ont proposé également dans (Weinberger and Mitra, 1984) une étude d'un modèle probabiliste du verrouillage dans les SGBD. Dans (Tseng et al., 1992), Tseng et al. ont proposé une approche probabiliste pour le traitement des transactions d'interrogation dans les bases de données hétérogènes.

Cependant, dans les SGBDTR sur lesquels les travaux ont commencé il y a plus de 15 ans, on ne trouve, à notre connaissance, qu'une étude probabiliste du comportement des transactions temps réel

[1] Worst Case Execution Time.

(Zhou et al., 1996). Les auteurs ont effectué des simulations et des mesures pour évaluer l'utilité des algorithmes d'ordonnancement temps réel traditionnels sur une architecture développée dans leur université (Michigan). Ils ont conclu que parmi les algorithmes *RM*, *EDF*, et *FIFO*, aucun n'est meilleur que l'autre en termes de minimisation du nombre de transactions qui manquent leurs échéances. Néanmoins, ils ont conclu que leur étude n'est pas satisfaisante, car leurs tests ont été réalisés sur un prototype basé sur le système d'exploitation temps réel QNX, qui contenait de nombreuses sources d'imprévisibilité. Nous pensons également que la raison vient du fait qu'ils avaient testé des algorithmes conçus spécifiquement pour l'ordonnancement des tâches temps réel.

Notre approche rentre donc dans le cadre de cette dernière étude. Dans ce travail, nous tentons de modéliser les transactions temps réel en termes probabilistes. Nous pensons que l'étude probabiliste se prête bien à ce genre d'environnements (les SGBDTR) qui sont la plupart du temps imprévisibles, i.e. les bases de données sont accédées de manière imprévisible.

8.2 Notre contribution : Comparaison de protocoles

8.2.1 Données et transactions temps réel

Nous considérons un modèle de SGBDTR comportant un nombre fini d'objets (données, transactions). Les transactions temps réel pouvant être de trois types (*hard, firm, soft*), nous ne considérons dans notre étude que les transactions de type *firm*.

Nous nous basons sur les travaux de (Ramamritham, 1993), qui sépare les données de la base en données temporelles[2] et données non temporelles. Ces dernières sont ensuite divisées en deux groupes : données de base et données dérivées.

1. Les données temporelles : ce sont des données ayant une durée de validité, au delà de laquelle elles deviennent obsolètes, ou perdent de la qualité (Amirijoo et al., 2003c). On distingue :
   - Les données de base : leurs valeurs représentent l'état spécifique d'entités de l'environnement. Exemple la température dans une centrale thermique.
   - Les données dérivées : leurs valeurs sont déduites (dérivées) à partir de valeurs des données de base.
2. Les données non temporelles : ce sont des données manipulées dans les SGBD traditionnels.

8.2.2 Réalisation d'un simulateur

Dans le but d'effectuer l'étude probabiliste des transactions temps réel, nous avons développé un simulateur de SGBDTR, dans lequel nous incorporons un certain nombre d'aléas : la génération de transactions, le type des transactions, les actions composant chaque transaction, les conflits d'accès potentiels aux données, le type de données.

Comme il n'existe pas de SGBDTR en utilisation normale de manière à avoir des données réelles de test, et qu'un prototype est plus difficile à développer et demande davantage de temps, notre choix s'était porté sur le développement d'un simulateur. Il existe beaucoup de simulateurs de SGBDTR, mais ils sont conçus pour un contexte particulier, avec des paramètres spécifiques tels qu'ils sont difficiles, voire impossibles, à exploiter pour nos besoins.

[2] Au sens temps r´eel du terme et non historique

Les composants du simulateur que nous avons développé ainsi que son fonctionnement (entrées, sorties, forme des résultats, ...) sont décrits dans un rapport de recherche du laboratoire (Semghouni et al., 2003). La figure 8.1 illustre le schéma synthétique du simulateur.

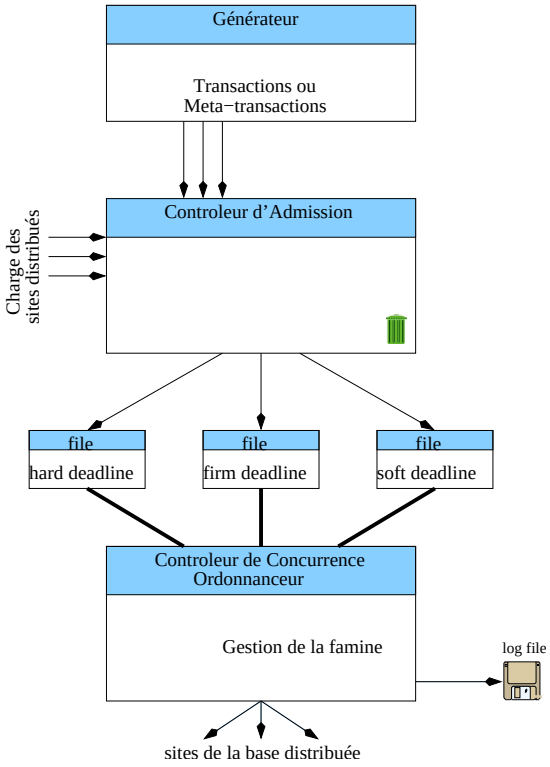

FIG. 8.1 – Architecture générale du simulateur

Nous considérons une base de données centralisée, résidant en mémoire centrale. Nous supposons que la durée d'une opération d'écriture est égale à deux fois la durée d'une lecture.

Les transactions arrivent dans le système selon un processus de Poissson avec la moyenne λ variant entre 0.3 et 1 avec un pas de 0.1.

Les premiers travaux que nous avons effectués concernent la comparaison de deux protocoles de contrôle de concurrence et d'ordonnancement des transactions. Après avoir généré les transactions qui arrivent sur le système selon un processus de Poisson, nous avons simulé les conflits potentiels d'accès aux données, et leur résolution avec un protocole pessimiste, 2PL-HP et avec un protocole optimiste WAIT-X et sa variante WAIT-50 (Haritsa et al., 1992) et Paragraphe 3.2.1.

8.2.3 Simulations et résultats

L'ordonnanceur sélectionne la transaction qui va être exécutée en fonction de sa priorité. Si un conflit d'accès aux données survient avec d'autres transactions, le contrôleur de concurrence en collaboration avec l'ordonnanceur le résout en préemptant ou en faisant attendre certaines transactions (en fonction du protocole).

La politique d'affectation des priorités aux transactions générées est donnée par la formule suivante :

$$D_T = A_T + B_T * (H(B_T) + 1)$$

où :

- $D_T =$ échéance de la transaction T.
- $A_T =$ date d'arrivée de T.
- $B_T =$ meilleur temps d'exécution estimé de T.
- $H(X) = \alpha * (1 - exp(-\beta * X))$ est un facteur multiplicatif.

  α et β sont deux paramètres dont les valeurs sont fixées par expérimentation.

Quand $\beta * X \rightarrow \infty^+$, alors la fonction $H(X)$ est maximisée par α. Dans nos expérimentations, les valeurs de $H(B_x)$ sont dans l'intervalle $[0, \alpha]$.

Chaque transaction se voit affecter un temps additionnel selon son meilleur temps d'exécution, calculé en fonction du nombre d'opérations lire/écrire dont elle est composée et de la capacité CPU.

Dans nos simulations, nous avons fait varier α entre 1 et 3, avec une valeur fixée de $\beta = 0.3$[3]

La représentation de la fonction H(X) est donnée dans la figure 8.2 où β = 0.3 et α = 1, 2 ou 3 et en faisant varier le meilleur temps d'exécution entre 1 et 10 unités.

Les comparaisons entre les performances des différents protocoles sont données dans les figures 8.3, 8.4 et 8.5. Ces résultats sont obtenus en faisant varier la charge du système (variation de H(X)).

D'après les simulations effectuées, nous constatons que le protocole optimiste OCC-Wait-50 et dans une moindre mesure OCC-WAIT, fournissent de meilleures performances quand le système n'est pas surchargé, alors que quand le système devient surchargé, le protocole pessimiste 2PL-HP devient meilleur.

Lorsque le système est surchargé, le nombre de redémarrages de transactions devient élevé, et donc le nombre de transactions qui risquent de manquer leurs échéances devient élevé également.

8.2.4 Conclusion et futurs travaux

D'autres résultats ont montré que lorsque la base de données est de petite taille ($<$ 100 objets), le protocole pessimiste présente de meilleures performances que le protocole optimiste, en prenant l'hypothèse que la probabilité qu'une transaction soit composée du même nombre d'opérations de lecture que d'écriture, est de 0.5.

L'objectif ultime de nos travaux est de pouvoir prédire de manière probabiliste, à partir des résultats de simulation, le nombre de transactions temps réel qui vont respecter leurs échéances, en fonction des probabilité d'accès aux données dans un mode conflictuel et de certaines hypothèses. Nous citons parmi ces hypothèses le type des transactions, la taille de la base, la popularité des données (les plus accédées

[3]Ces valeurs de α et β sont choisies apr`es des tests car elles offrent des r´esultats plus r´ealistes.

et dans quel mode), le type des données (temporelles, non temporelles).

8.2.5 Encadrements, collaborations et résultats préliminaires

Les travaux décrits dans ce chapitre entrent dans le cadre de la thèse du doctorant S. Semghouni, commencée en 2003-2004 et que je co-encadre.

Les premiers résultats que nous avons obtenus concernant la comparaison des taux de succès des transactions temps réel en fonction du protocole utilisé sont parus dans la conférence nationale MajecS-TIC'04 (Semghouni et al., 2004).

Les autres participants à ces travaux sont L. Amanton et moi-même (maîtres de conférences), et A. Berred (Professeur), co-encadrants de la thèse.

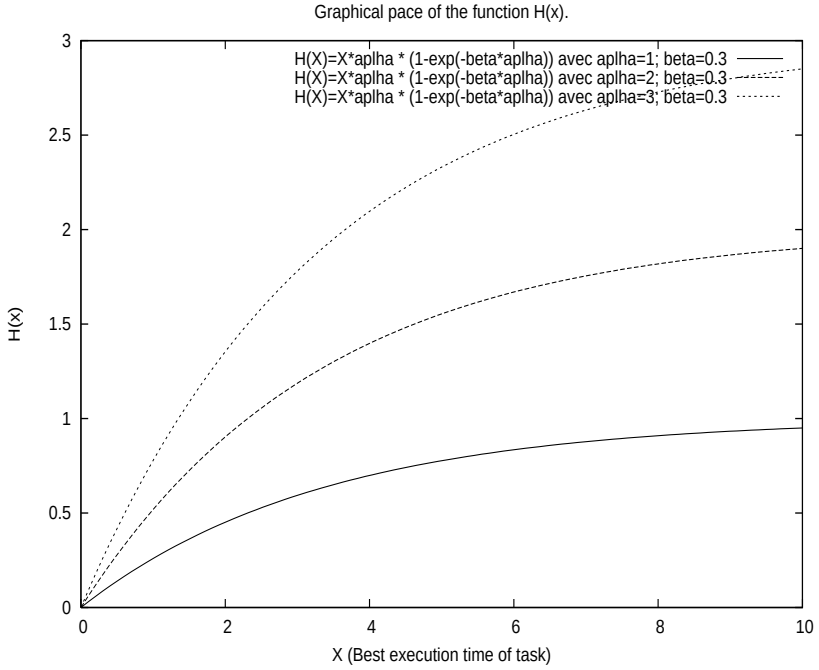

FIG. 8.2 – Représentation de la fonction H(X)

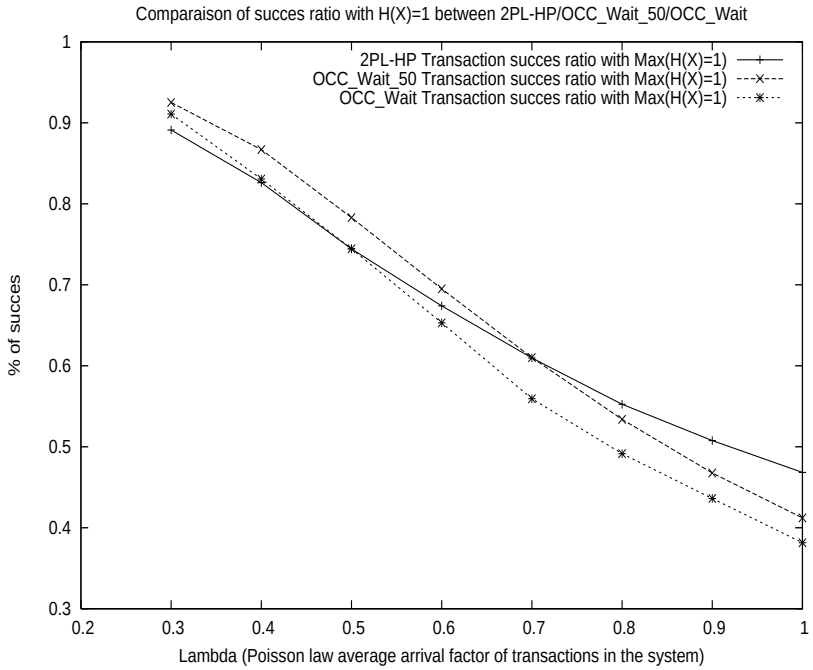

FIG. 8.3 – Comparaison du taux de succès entre des protocoles de CC&O pour H(X)=1

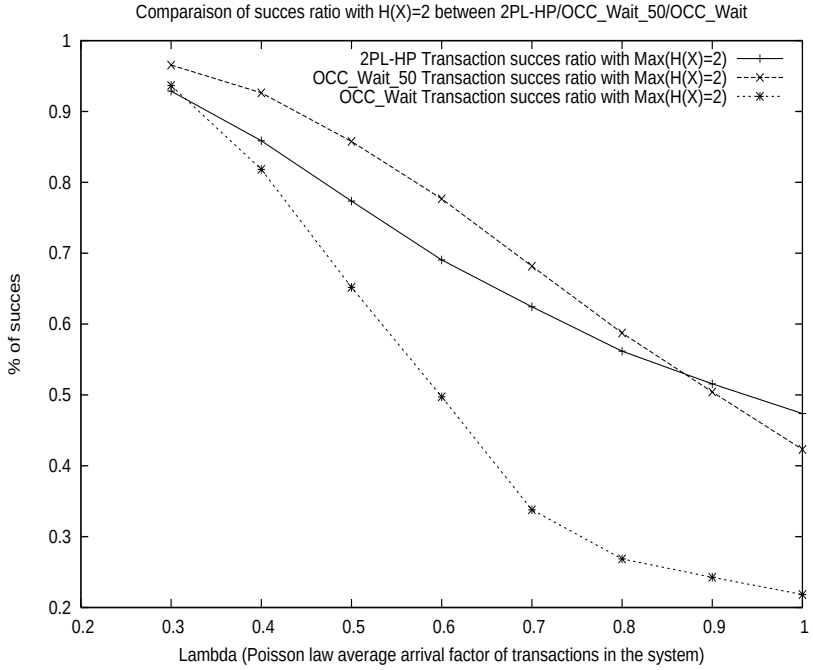

FIG. 8.4 – Comparaison du taux de succès entre des protocoles de CC&O pour H(X)=2

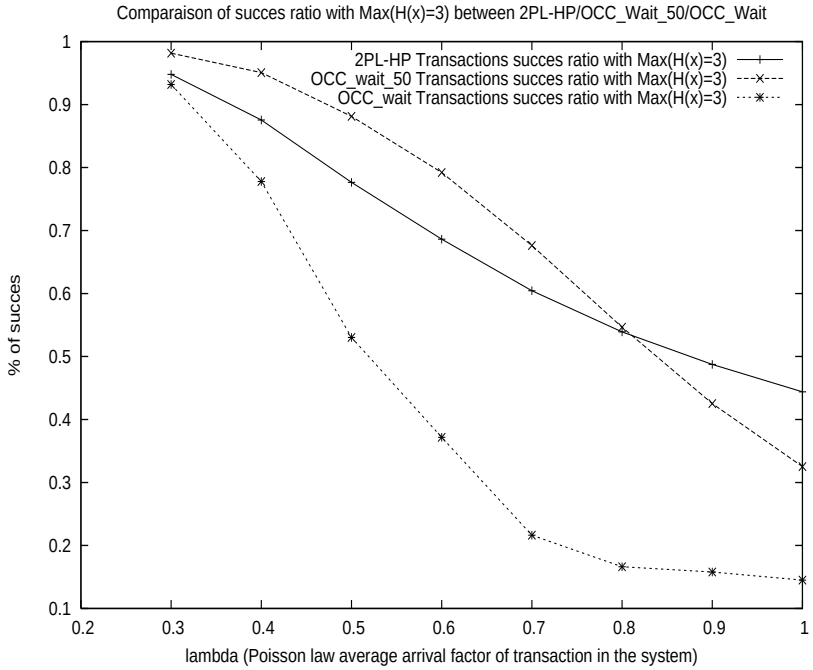

FIG. 8.5 – Comparaison du taux de succès entre des protocoles de CC&O pour H(X)=3

9 Conclusions et perspectives

Dans ce mémoire, nous avons présenté notre contribution dans un domaine de recherche en plein essor : les SGBD temps réel. En effet, la plupart des applications ont actuellement de grands besoins en traitement de données temps réel. Pour cela, les recherches ont débuté à la fin des années 80 sur des systèmes qui devraient être capables à la fois de gérer de grands volumes de données et de respecter des contraintes temporelles associées aux actions et aux données. Ces contraintes prennent souvent la forme d'échéances pour les transactions et de durées de validité pour les données. Les échéances des transactions peuvent provenir de contraintes spécifiques à l'environnement ou de contraintes dues à la manipulation de données ayant des durées de validité.

Au vu des très nombreux travaux de recherche sur les SGBD temps réel, nous pouvons dire que ces systèmes ont maintenant atteint leur stade de maturité. La majorité des problèmes ont été étudiés, comme la gestion des transactions, en passant par la gestion de la mémoire, les entrées/sorties, le recouvrement, etc. À partir de ces recherches, nous pensons pouvoir donner quelques éléments relatifs à la manière de concevoir les applications actuelles.

Dans les applications, il est nécessaire de distinguer plusieurs composants. Un composant qui gère les fonctions vitales de l'application, i.e. isoler les données et transactions vitales pour le système. À ce composant, il est nécessaire d'appliquer les techniques déjà proposées et éprouvées depuis plusieurs années dans les systèmes temps réel. Le non-respect des échéances de ce type de transactions pourrait conduire à des catastrophes. Les données critiques dans ce cas doivent résider en mémoire centrale, et utiliser des techniques efficaces de journalisation pour garder une version de la base sur support stable. Les accès aux données doivent être maîtrisés, c'est-à-dire que le temps d'accès maximal pour les lectures et les écritures doit être connu. Ensuite, il faut isoler un autre composant qui gère les fonctions moins critiques de l'application. Il s'agit là d'essayer d'offrir des mécanismes qui favorisent le respect des échéances d'un maximum de transactions de ce type. Le non-respect des échéances n'est pas catastrophique : il s'agit seulement d'abandonner les transactions en cause car elles deviennent sans valeur pour l'application.

Le dernier composant comportera les autres transactions qui permettent d'ajuster la qualité de service de l'application en fonction de certains paramètres. Plus il y a de transactions qui respectent leurs échéances, meilleure sera la qualité de service offerte par l'application. Dans le cas où toutes les transactions respectent leurs échéances, la qualité de service offerte par l'application sera maximale, c'est-à-dire que les résultats fournis seront complets, exacts et exploitables en l'état au moment de leur délivrance.

Des problèmes de standardisation persistent qui pourraient être résolus en réunissant les résultats des recherches actuelles sur les SGBD temps réel dans des conférences spécialisées, comme celles qui avaient eu lieu en 1995, 1996 et 1997, sur les SGBDTR et sur les SGBDTR actifs (Berndtsson and Hansson, 1996; Andler and Hansson, 1997).

Nous espérons, à travers ce document, avoir contribué à susciter l'intérêt des chercheurs français en bases de données d'une part, et en systèmes temps réel d'autre part, pour engager davantage de travaux sur la prise en compte de contraintes temporelles dans les SGBD.

Les activités actuelles de notre équipe de recherche au sein du laboratoire d'informatique du Havre ont déjà conduit à plusieurs collaborations, que nous espérons renforcer et d'autres que nous allons établir. Ces collaborations sont, notamment :

- avec l'équipe systèmes temps réel et réseaux (équipe STRR) de l'IRIT-Toulouse, sur les aspects qualité de service, qualité des données et qualité des transactions temps réel, notamment dans le cadre d'un projet ACI Jeunes-Chercheurs auquel nos deux équipes participent.
- avec l'équipe Techniques Formelles et à Contraintes (TFC) du laboratoire LIFC de Besançon, sur les aspects modélisation des contraintes temporelles dans les bases de données temps réel,
- avec l'équipe *RTDBS group* du *Department of Computer and Information Science Linköping University*, Suède, sur les aspects ordonnancement par rétroaction des transactions temps réel,
- avec l'équipe Probabilités/Statistiques du laboratoire LMAH de l'université du Havre, notamment dans le cadre d'un projet ACI Jeunes-Chercheurs auquel nos deux équipes participent.

D'autres collaborations sont également à établir entre notre équipe et des équipes travaillant :

i sur les bases de données, en particulier celles qui font partie du nouveau groupe : *Impact du Grid/peer-to-Peer computing sur les bases de données*, du GDR I3, auquel notre équipe participe. Nous pensons introduire la notion de contraintes temporelles dans de tels systèmes de bases de données, notamment pour garantir une certaine qualité de service aux utilisateurs.

ii sur les aspects réactivité et mobilité dans les systèmes d'information géographiques (SIG). Dans ce cadre, nous avons commencé à établir un rapprochement avec une équipe travaillant sur les SIG à l'INSA de Rouen (Mainguenaud, 2000), et nous avons commencé à établir des contacts pour intégrer le GDR SIGMA (appelé jusqu'à maintenant CASSINI (Laurini, 2003))

Comme projet de recherche et en plus des améliorations que nous comptons apporter aux travaux que nous avons déjà effectués, nous avons défini deux nouveaux axes de recherche sur lesquels nous pensons travailler. Ce sont :

1. La validation et la tolérance aux fautes des transactions distribuées temps réel

   Dans les SGBD distribués temps réel, l'un des principaux problèmes qui s'ajoute aux problèmes des SGBDTR est la validation des transactions distribuées. Le problème de la validation des transactions est très étudié dans le contexte traditionnel. Les études sur cet aspect dans un contexte temps réel sont rares.

   L'objectif de ces travaux est d'étudier le processus de commit des transactions temps réel, en tenant compte des diverses possibilités : transactions avec parties obligatoires et optionnelles, tolérance aux fautes, commit progressif, etc. Il s'agit en particulier de proposer des protocoles de validation qui respectent les échéances des transactions, qui maintiennent la cohérence de la base de données distribuée et qui soient résistants aux pannes de sites ou de réseaux, sachant que lorsque les deux types de pannes sont présents, il n'y a pas de solution satisfaisante au problème (Bernstein et al., 1987).

   Toujours dans le cadre de ce travail, il s'agit d'étudier la validation progressive (Soparkar, 1994) d'une transaction distribuée, qui consiste à récupérer des résultats exploitables si l'échéance de la transaction risque d'être dépassée.

2. Les méthodes de conception de bases de données temps réel

   Dans le domaine des SGBD temps réel, beaucoup de travaux ont été effectués. Ils ont permis de faire des avancées significatives dans le domaine comme nous l'avions indiqué dans le chapitre 2. Il s'agit notamment de l'étude des protocoles de contrôle de concurrence et d'ordonnancement, de l'epsilon sérialisabilité, de la gestion de la qualité de service, ... D'autres travaux ont concerné la conception de simulateurs et de prototypes de SGBDTR.

   Cependant, il n'existe pas à notre connaissance de travaux sur la conception de bases de données temps réel. Des travaux sur les méthodes de conception des bases de données existent et beaucoup de méthodes issues de ces travaux ont largement fait leur preuve (MERISE, Modèle Entité/Association, Relationnel, Relationnel-Objet, ...). Sur la conception de systèmes temps réel, il y a également beaucoup de méthodes de spécification et de vérification et des recherches intensives (Toussaint et al., 1997; Diaz et al., 2000; Mammeri, 1998) s'effectuent encore pour la prise en compte des contraintes temporelles dans les méthodes formelles pour les étendre à la conception de systèmes temps réel robustes.

   Des méthodes de conception de bases de données sont nécessaires si l'on souhaite que les travaux sur les SGBD temps réel soient mis en application de façon plus importante.

   Des voies de recherche possibles sont l'ajout de spécifications d'attributs temps réel dans les objets (début et durée de validité par exemple), mais il est également nécessaire de spécifier des contraintes temporelles sur les transactions (dates de réveil, échéances, périodicité, ...). Ensuite, il s'agit de voir s'il est possible d'étendre les méthodes existantes pour tenir compte des ces contraintes, ou bien concevoir de nouvelles méthodes. Ces méthodes peuvent être formelles, ou bien en partie formelles pour spécifier les contraintes temporelles, et en partie graphiques pour spécifier d'autres caractéristiques.

   L'un des objectifs est de pouvoir passer d'un modèle conçu selon la méthologie proposée vers un modèle plus facile à implémenter sur machine et qui permette une manipulation aisée avec un langage tel que RT-SQL (Prichard and Fortier, 1997).

Bibliographie

Abbot, R. and Garcia-Molina, H. (1988). Scheduling Real-Time Transactions : A Performance Evaluation. In *Proc. of the 14th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 1–12, Los Angeles, California.

Abbot, R. and Garcia-Molina, H. (1989). Scheduling Real-Time Transactions with Disk Resident Data. In *Proc. of the 15th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 385–396, Amsterdam, The Netherlands. Morgan Kaufmann.

Abbot, R. and Garcia-Molina, H. (1992). Scheduling Real-Time Transactions : a Performance Evaluation. *ACM Transactions on Database Systems*, 17(3) :513–560.

Abdallah, M., Guerraoui, R., and Pucheral, P. (1998). One-Phase Commit : Does it make Sense ? In *Intl. Conf. on Parallel and Distributed Systems (ICPADS)*, pages 182–192.

Abdallah, M. and Pucheral, P. (1998). A Single-Phase Non-Blocking Atomic Commitment Protocol. In *Database and Expert Systems Applications (DEXA'98)*, pages 584–595.

Abdouli, M., Sadeg, B., and Amanton, L. (2003a). Gestion des Transactions Temps R´eel Embo$\hat{\imath}$t´ees. In *Conference Nationale MajecSTIC'03*, Marseille, France.

Abdouli, M., Sadeg, B., and Amanton, L. (2004a). Enhancing the Success Ratio of Distributed Real-Time Nested Transactions. In *Proceedings of ICEIS'04 - Database and Information Systems topic*, volume 1, pages 233–240, Porto, Portugal. ed. INSTICC Press in collaboration with ACM, AAAI. ISBN 972-8865-00-7.

Abdouli, M., Sadeg, B., and Amanton, L. (2004b). Une adaptation d'un protocole d'h´eritage descendant de verrous pour les transactions temps r´eel embo$\hat{\imath}$t´ees. In *8th Maghrebian Conference on Computer Technology ($MCSEAI'04$)*, Sousse, Tunisia.

Abdouli, M., Sadeg, B., Amanton, L., and Berred, A. (2003b). Management of Distributed Nested Real-Time Transactions. In *Intl. Conf. on Parallel and Distributed Computing Systems (PDCS'03)*, pages 230–235, Reno, Nevada, USA. Ed. S.M.Yoo and H.Y.Yoon. ISBN 1-880843-48-X.

Abeni, L. and Buttazzo, G. (1999). QoS Guarantee Using Probabilistic Deadlines. In *Proc. of Euromicro Conf. on Real-Time Systems (ECRTS'99)*, York.

Abouaissa, H., Amanton, L., and Lorenz, P. (2001). An Oriented Diffusion Algorithm for Routing Paths in MPLS Network. In *4th IEEE Intl. Conf. on ATM and High Speed Intelligent Internet (ICATM 2001)*, Olympic Parktel, Seoul, Korea. april.

Acharya, A., Badrinath, B., and Imielinski, T. (1994). Structuring distributed algorithms for mobile hosts. In *14th Intl. Conf. on Distributed Computing Systems*.

Adelberg, B. and Kao, B. (2001). Prototypes : Programmed Stock Trading. In Lam, K. and Kuo, T.-W., editors, *Real-Time Database Systems : Architecture and Techniques*, chapter 19, pages 261–277. Kluwer Academic Publishers.

Adiba, M. and Collet, C. (1993). *Objets et bases de données, le SGBD O2*. Herm`es.

Al-Houmaily, Y. J. and Chrysanthis, P. K. (1999). Atomicity with Incompatible Presumptions. In *Proc. of the 8th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Phildelphia, PA, USA.

Amanton, L., Bouzefrane, S., and Sadeg, B. (2002a). The Causal-Phase Ordering Protocol to Manage Distributed Real-Time Transaction Commit. In *Research Report - LIH-01-02*, University of Le Havre.

Amanton, L. and Naimi, M. (1999). A New Causal Ordering Protocol for Overlapped Broadcasts. *International Journal on Informatics*, pages 47–66.

Amanton, L. and Na¨ı mi, M. (1996). Total and causal ordering implementation in distributed computations. In *8th Intl. Conf. of Computing and Information (ICCI'96)*, volume 2.

Amanton, L. and Na¨ı mi, M. (1998). Algorithme d'ordonnancement causal dynamique par phases avec initiateurs multiples. In *Conf. Intl. sur les Nouvelles TEchnologies de la REpartition (NOTERE'98)*, Montr´eal, Canada.

Amanton, L. and Na¨ı mi, M. (1999a). A Causal-Phase Ordering Protocol with Multi-Initiators for Overlapped Broadcasts. In *24th IEEE Annual Conference on Local Computer Networks (LCN'99)*, Boston, Massachusetts, USA.

Amanton, L. and Na¨ı mi, M. (1999b). A new causal ordering protocol for overlapped broadcasts. In *OPODIS'99, 3rd Intl. Conf. on Principles of Distributed Systems*, Hano¨ı , Vietnam.

Amanton, L. and Na¨ı mi, M. (1999c). Ordonnancement causal dynamique tol´erant aux fautes pour groupes de processus mobiles. In *DNAC'99, Congrès des Nouvelles Architectures pour les Communications*, Minist`ere charg´e des Telecoms, Paris.

Amanton, L. and Sadeg, B. (2003). Towards the Commit Process of Distributed real-time transactions. In *Proceedings of IEEE ISSPIT'03*, Darmstadt, Germany.

Amanton, L., Sadeg, B., and Bouzefrane, S. (2002b). RT_DIS_COM : A Protocol Suitable for Real-Time Distributed Transaction Commit. In *IEEE Intl. Symp. on Parallel and Distributed Computing (IEEE ISPDC'02) - Appeared in Informatica journal, Tome XI*, pages 337–348.

Amanton, L., Sadeg, B., and Saad-Bouzefrane, S. (2001). Disconnection Tolerance in Soft Real-Time Mobile Databases. In *16th ISCA Intl. Conf. on Computers And Their Applications (CATA'2001)*, Seattle, Washington, USA.

Amirijoo, M., Hansson, J., and Song, S. (2003a). Algorithms for Managing QoS for Real-Time Data Services Using Imprecise Computation. In *Proc. of Intl. Conf. on Real-Time and Embedded Computing Systems and Applications (RTCSA'03)*, Taiwan, R.O.C.

Amirijoo, M., Hansson, J., and Song, S. (2003b). Error-Driven QoS Imprecise Management in Imprecise Real-Time Databases. In *Proc. of Euromicro Conf. on Real-Time Systems (ECRTS'03)*, Portugal.

Amirijoo, M., Hansson, J., and Song, S. (2003c). Specifi cation and Management of QoS in Imprecise Real-Time Databases. In *Proc. of Datatabases Engineering and Application Symposium (IDEAS'03)*, Hong Kong.

Andler, S. and Hansson, J., editors (1997). *Active, real-Time and Temporal Database Systems (ARTDB'97)*, Como, Italy. Proceedings of the Second International Workshop on Active, Real-Time and Temporal Database Systems, Springer.

Andler, S., Hansson, J., Eriksson, J., Mellin, J., Berndtsson, M., and Eftring, B. (1996). DeeDS : Towards a Distributed and Active Real-Time Database System. $ACM\ SIGMOD$ *Record*, 15(1) :38–40.

Arahna, R., Gianti, V., Narayanan, S., C.R. Munthukrishnan, S. P., and Ramamritham, K. (1996). Implementation of a Real-Time Database System. *Information Systems*, 21(1).

Atlas, A. and Bestavros, A. (1998). Statistical rate monotonic scheduling. In *IEEE Real-Time Systems Symposium*, Madrid, Spain.

Audsley, N., Burns, A., Richardson, M., and Wellings, A. (1991). Hard Real-Time Scheduling : The Deadline Monotonic Approach. In *Proc. of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, USA.

Aydin, H., Melhem, R., Moss´e, D., and Mejia-Alvarez, P. (2001). Optimal Reward-Based Scheduling for Periodic Real-Time Tasks. *IEEE Transactions on Computers*, 50(2) :111–130.

Ayeb, B. (1999). Fault identifi cation algorithmic : A new formal approach. In *20th International Symposium on Fault-Tolerant Computing*.

Badrinath, B. and Sudame, P. (1996). To Send or not to Send : Implementing Deferred Transmissions in a Mobile Host. In *Intl. Conf. on Distributed Computing Systems*, pages 327–333.

Baldoni, R., Mostefaoui, A., and Raynal, M. (1996a). Causal deliveries of messages with real-time data in unreliable networks. *Real-time systems*, 10(3).

Baldoni, R., Mostefaoui, A., and Raynal, M. (1996b). Causal multicast in unreliable networks for multimedia applications. In *15th IEEE International Conference on Computers and Communications*, pages 51–57, Phoenix, AZ.

Banerjee, S. and Chrysanthis, P. (1997). Performance Evaluation of the Group Two-Phase Locking Protocol. In *Proc. of the 10th Intl. Conf. on Parallel and Distributed Computing Systems (PDCS'97)*, pages 428–432.

Banerjee, S. and Chrysanthis, P. (1998). Network Latency Optimization in Distributed Database Systems. In *Proc. of the Intl. Conf. on Data Engineering (ICDE'98)*, Orlando, Florida, USA.

Banerjee, S. and Chrysanthis, P. (1999). Group Two-Phase Locking : A Scalable Data Sharing Protocol. *IEICE Transactions on Information Sytems*, E82-D(1) :236–245.

Bayer, R., Elhardt, E., Heigert, H., and Reiser, A. (1982). Dynamic Timestamp Allocation for Transactions in Database Systems. In Schneider, H., editor, *Proc. of the 2nd Intl. Symp. on Distributed Databases*, Berlin. North Holland Publishing Company.

Berndtsson, M. and Hansson, J. (1996). Workshop Report : The First International Workshop on Active and Real-Time Database Systems (ARTDB-95). *SIGMOD Record*, 25(1) :64–66.

Bernstein, P. and Goodman, N. (1983). Multiversion Concurrency Control - Theory and Algorithms. *ACM Transactions on Database Systems (TODS)*, 8(4) :465–483.

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

Bertino, E., Catarci, T., Elmagarmid, A. K., and Hacid, M.-S. (2003). Quality of Service Specifi cation in Video Databases. *IEEE Multimedia*, 10(4) :71–81.

Besancenot, J., Cart, M., Ferri´e, J., Guerraoui, R., Pucheral, P., and Traverson, B. (1997). *Les systèmes transactionnels : concepts, normes et produits*. ed. Hermes.

Bestavros, A. (1992). Speculative Concurrency Control. Technical Report TR-16-92, Boston University, Boston, MA.

Bestavros, A. and Braoudakis, S. (1995). Value-cognizant Speculative Concurrency Control. In *Proc. of the International Conference on Very Large Databases (VLDB)*.

Bestavros, A. and Braoudakis, S. (1996). Timeliness via Speculation for Real-Time Databases. In *proc. of the* 15^{th} *IEEE Real-Time Systems Symposium (RTSS'96)*, San Juan, Puerto Rico.

Bestavros, A., Braoudakis, S., and Panagos, E. (1993). Performance Evaluation of Two-Shadow Speculative Concurrency Control. Technical Report 1993-001, Boston University, Boston, MA.

Bestavros, A., Lin, K.-J., and Son, S. (1996a). Special section on advances in real-time database systems. *ACM SIGMOD Record*, 25(1).

Bestavros, A., Lin, K.-J., and Son, S. (1996b). Workshop Report : 1^{st} International Workshop on Real-Time Database Systems, New beach, CA. Technical report.

Bhargava, B. K. (1999). Concurrency Control in Database Systems. *Knowledge and Data Engineering*, 11(1) :3–16.

Birman, K., Shiper, A., and Stephenson, P. (1991). Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3) :272–314.

Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3) :637–654.

Boksenbaum, C., Cart, M., Ferrie, J., and Pons, J. (1987). Concurrent Certifi cation by Intervals of Time stamps in Distributed Database Systems. *IEEE T.S.E.*, 13(4) :409–419.

Bonnet, C. and Demeure, I. (1999). *Introduction aux systèmes temps réel*. Herm`es.

Bouazizi, E., Duvallet, C., and Sadeg, B. (2004a). Utilisation de donn´ees multi-versions et de l'ordonnancement contrˆol´e par r´etro-action pour les transactions temps r´eel. In *MajecSTIC'04*, Calais, France.

Bouazizi, E., Sadeg, B., and Duvallet, C. (2004b). Applicabilit´e du crit`ere d'epsilon s´erialisabilit´e pour les transactions temps r´eel. In 8^{th} *Maghrebian Conference on Computer Technology (*$MCSEAI'04$*)*, Sousse, Tunisia, Article pr´e-s´electionn´e en juin 2004 pour apparaˆıtre dans un num´ero sp´ecial de ISDM journal.

Bouzefrane, S. and Sadeg, B. (2000a). How to Manage Real-Time Transactions in Non-Strict Context. In *Proc. of the 3rd IEEE Intl. Conf. on Application-Specifi c Systems, Software Eng. and Technology (IEEE ASSET'2000)*, pages 179–186, Dallas, Texas, USA. IEEE Computer Society - Order Number PR00559. ISBN 0-7695-0559-7.

Bouzefrane, S. and Sadeg, B. (2000b). Relaxing the Correctness Criteria in Real-Time DBMSs. *International Journal of Computers and Their Applications*, 7(4) :209–217.

Bouzefrane, S., Sadeg, B., and Amanton, L. (2001). Soft Real-Time Transactions Scheduling in a Wireless Environment. In *4^{th} IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (IEEE ISORC-2001)*, Magdeburg, Germany.

Bouzeghoub, M., Fabret, F., Matulovic, M., and Simon, E. (1999). Update propagation and datawarehouse refreshment. In Jarke, Lenzerini, Vassiliou, and Vassiliadis, editors, *Fundamentals of Data Warehouses*, chapter 14. Springer.

Braoudakis, S. (1995). *Concurrency Control Protocols for Real-Time Databases*. PhD thesis, Boston University, USA.

Bricker, A., Gien, M., Guillemont, M., Lipkis, J., Orr, D., and Rozier, M. (1991). Architectural Issues in Microkernel-based Operating Systems : the CHORUS Experience. *Computer Communications*, 14(6) :347–357.

Carey, M., jahauri, R., and Livny, M. (1989). Priority in DBMS Resource Scheduling. In *Proc. of the 15^{th} VLDB*.

Chakravarthy, S., Krishnaprasad, V., Anwar, E., and Kim, S. (1994). Composite Events for Active Database : Semantics, Contexts and Detection. In 20^{th} *Intl. VLDB Conf.*, pages 606–617, Santiago, Chile.

Chakravarthy, S. and Mishra, D. (1994). Snoop : An event specifi cation language for active databases. *Knowledge and Data Engineering*, 13(3).

Chakravarty, S., Hong, D., and Johnson, T. (1998). Real-time transaction scheduling : a framework for synthetizing static and dynamic factors. *Journal of RTS*, 14 :135–170.

Chen, Y. and Gruenwald, L. (1996). Effects of Deadline Propagation on Scheduling Nested Transactions in Distributed Real-Time Database Systems. *Information Systems*, 21(1).

Chrysanthis, P. and Ramamritham, K. (1992). *ACTA : The Saga Continues*. Transactions Models for Advances Database Applications (Chap. 10) - Morgan Kaufmann Publishers.

CNRS (Groupe de R´eflexion Temp R´eel) (1988). Le temps r´eel. *Technique et Science Informatiques*, 7(5) :493–500.

Collet, C. (1998). *The NAOS rule system*, chapter In Active Rules for Databases, pages 279–296. Springer Verlag.

Collet, C. and Coupaye, T. (1998). The NAOS system. *ACM Sigmod Record*, 24(2).

Collet, C., Coupaye, T., and Svensen, T. (1994). NAOS – Effi cient and Modular Reactive Capabilities in an Object-Oriented Database System. In *Proceedings of the 20^{th} International Conference on Very Large Databases*, pages 132–143, Santiago, Chile.

Collet, C., Habraken, P., and Roncancio, C. (1996). R`egles actives dans les SGBD. *Ingénierie des Systèmes d'Information (ISI)*, 4(3).

Cottet, F., Delacroix, J., Kaiser, C., and Mammeri, Z. (2000). *Ordonnancement Temps Réel*. Herm`es.

Coupaye, T. and Collet, C. (1998). Mod`eles de comportement des SGBD actifs : caract´erisation et comparaison. *Technique et Science Informatiques (TSI)*, 17(3) :299–328.

Date, C. J. (1985). *An Introduction to Database Systems*. Addison-Wesley.

Date, C. J. (1997). Introduction aux Bases de Donn´ees. ed. International Thomson Publishing - France. Traduction Fr´ed´eric Cuppens.

Datta, A., Celik, A., Kim, J., and VanderMeer, D. (1997). Adaptive broadcast protocol to support power conservant retrieval by mobile users. In *Proc. of the 13^{th} intl. Conf. on Data Engineering*, Birmingham, U.K.

Datta, A. and Mukherjee, S. (2001). Buffer Management in Real-Time Active Database Systems. In Lam, K. and Kuo, T.-W., editors, *Real-Time Database Systems : Architecture and Techniques*, chapter 7, pages 77–95. Kluwer Academic Publishers.

Datta, A., Mukherjee, S., and Viguier, I. (1998). Buffer management in real-time active database systems. *Systems and Software*, 42 :227–246.

David, L., Cottet, F., and Nissanke, N. (2003). Ordonnancement de tˆaches p´eriodiques `a dur´ees d'ex´ecution ind´etermin´ees : une approche probabiliste. In *Actes de la Conf. Real-Time and embedded Systems (RTS'03)*, Paris.

Dayal, U. (1989). Active Database Management Systems. *ACM Sigmod Record*, 18(3) :150–169.

Dayal, U., Blaustein, B., Buchmann, A., Chakravarthy, S., Hsu, M., Ladin, R., McCarthy, D., Rosenthal, A., Sarin, S., Carey, M., Livny, M., and Jauharu, R. (1988). The HiPAC project : Combining active databases and timing constraints. *ACM Sigmod Record*, 17(1).

Dean, T. and Boddy, M. (1988). Solving time-dependant planning problems. In *Proceedings of the* 7^{th} *National Conference of Artifi cial Intelligence*, pages 419–454, Minneapolis, Minnesota.

Delobel, C. and Adiba, M. (1982). *Bases de données et systèmes relationnels*. DUNOD.

DeWitt, D. (1985). Benchmarking Database Systems. Past Effords and Future Directions. *IEEE Database Eng. Bull.*, 8(1) :2–9.

Diaz, G., Thomesse, J., and Mammeri, Z. (2000). An Object-Oriented Modeling of Co-operative Multimedia Systems. In *Proc. of EuroMicro*.

Dogdu, E. (1998). *Real-Time Databases : Extended Transactions and the Utilization of Execution Histories*. PhD thesis, Case Western Reserve University, USA.

Duvallet, C. (2001). *Des systèmes d'aide à la décision temps réel et distribués : modélisation par agents*. PhD thesis, Universit´e du Havre.

Duvallet, C., Mammeri, Z., and Sadeg, B. (1999a). Analyse des protocoles de contrˆole de concurrence et des propri´et´es ACID dans les SGBD temps r´eel. In *Intl. Conference on Real-Time and embedded Systems (RTS'1999)*, pages 124–138, Paris, France, ed. Teknea.

Duvallet, C., Mammeri, Z., and Sadeg, B. (1999b). Les SGBD temps r´eel. *revue Technique et Science Informatiques (TSI)*, 18(5) :479–517.

Duvallet, C. and Sadeg, B. (2001). Concevoir un syst`eme d'aide `a la d´ecision bas´e sur un syst`eme multi-agents Anytime. In *JFIADSMA'2001*, Montr´eal, Canada.

Duvallet, C. and Sadeg, B. (2004). Des syst`emes multi-agents Anytime pour la conception de syst`emes d'aide `a la d´ecision. *Revue Technique et Science Informatiques (*TSI*) -à paraî tre*.

Duvallet, C., Sadeg, B., and Cardon, A. (1999c). An Anytime Multi-Agent System to Manage an Electronic Marketplace. In *Intl. Conf. on Artifi cial Intelligence in Electronic Commerce (AIEC'99)*, Sydney, Australia.

Duvallet, C., Sadeg, B., and Cardon, A. (1999d). Real-Time in Multi-Agent Systems. In *Intl. ISCA CAINE'99 Conf.*, pages 212–215, Atlanta, Georgia, USA, ed. Q. Yang.

Duvallet, C., Sadeg, B., and Cardon, A. (2000a). An Anytime Multi-Agent System to Manage Electronic Commerce Transactions. In *OOIS-2000, Springer Verlag ed.*, London, England.

Duvallet, C., Sadeg, B., and Cardon, A. (2000b). How to Build Real-Time Multi-Agent Systems Using Anytime Technics. In *Intl. Conf. on Computers and Their Applications (CATA'2000)*, pages 326–329, New-Orleans, Louisiana, USA.

Effelsberg, W. and Harder, T. (1984). Principles of Database Buffer Management. *ACM Transactions on Database Systems*, 9(4) :560–595.

Elhadef, M. and Ayeb, B. (2001). Effi cient comparison-based fault diagnosis of multiprocessor systems using an evolutionary approach. In 15^{th} *International Parallel and Distributed Processing Symposium (IPDPS'01)*.

Elloy, J. (1988). Groue de r´eflexion temps r´eel du CNRS : Le temps r´eel. *Technique et Science Informatiques (TSI)*, 7(5) :493–500.

Elmagarmid, A. (1992). *Transaction Models for Advanced Applications*. Morgan Kaufmann.

Eswaran, K., Gray, J., Lorie, R., and Traiger, I. (1976). The notions of consistency and predicate locks in a database system. *Communication of the ACM*, 19(11) :624–630.

Ezhilchelvan, P. D., Macedo, R. A., and Shrivastava, S. K. (1995). Newtop : A Fault-Tolerant Group Communication Protocol. In *International Conference on Distributed Computing Systems*, pages 296–306.

Garcia-Molina, H., Labio, W. J., Wiener, J., and Zhuge, Y. (1998). Distributed and parallel computing issues in data warehousing. In *PODC*.

Gardarin, G. (2000). *Bases de Données Objet et Relationnelles*. Eyrolles.

Gardarin, G. (2002). *XML : Des bases de données aux services WEB*. Dunod.

Gardarin, G. (2003). *Les Bases de données : systèmes et langages*. Eyrolles (5^{eme} ´edition).

Gardarin, G., Mensch, A., Dang-Ngoc, T.-T., and Smit, L. (2002). Integrating Heterogeneous Data Sources with XML and XQuery. In *DEXA Workshops*, pages 839–846.

Gatziu, S. and Dittrich, K. (1993). Events in an Active Object-Oriented Database System. In Paton, N. and Williams, H., editors, *Proc. of the* 1^{st} *Intl. Workshop on Rules in Database Systems (RIDS)*, Edinburgh, UK. Springer-Verlag, Workshops in Computing.

Gauthier, J. (1999). Construction de syst`emes temps r´eel et int´egr´es avec JAVA. In *Proc of 30th Annual Soft. Tech. Conf.*

Gehani, N., Jagadish, H., and Schmueli, O. (1993). COMPOSE : A system for composite event specifi cation and detection. In Adam, N. R. and Bhargava, B., editors, *Advanced Database Concepts and Research Issues. Lecture Notes in Computer Science*. Springer-Verlag.

Gien, M. (1995). Evolution of the Chorus open microkernel architecture : the stream project. In *Proceeding of the* 5^{th} *International Workshop on Confi gurable Distributed Systems*, pages 7–13.

Graham, M. (1992). Issues in real-time data management. *RTS Journal*, 4(3) :185–202.

Graham, M. (1993). How to get serialisability for real-time transactions without having to pay for it. In *Proc of the* 14^{th} *IEEE Real-Time Systems Symposium*, pages 56–65.

Grass, J. and Zilberstein, S. (1995). Programming with Anytime Algorithms. In *Proceedings of the 14th Internationnal Joint Conference on Artifi cial Intelligence*, pages 22–27, Montreal.

Gray, J. (1978). Notes on Database Operating Systems. *Lecture Notes in Computer Science*, 60 :393–481.

Gray, J. (1993). *The Benchmarks HandBook for DataBases and Transaction Processing Systems*. Morgan Kaufmann.

Gray, J. and Reuter, A. (1993). *Transaction Processing : Concepts and Technics*. Morgan Kaufmann.

Gruenwald, L. and Cheng, J. (1997). A Logging Technique Based on Transaction Types in Real-Time Databases. In *Proc. of RTDB'97*.

Guerraoui, R. and Schiper, A. (1995). The Decentralized Non-Blocking Atomic Commitment Protocol. In *Proc of the 7th IEEE Symposium on Parallel and Distributed Processing (SPDP-7)*, pages 2–9, San Antonio, Texas, USA.

Guessoum, Z. (1997). DIMA, une plate-forme multi-agents en Smalltalk. *Objet*, 3(4) :393–409.

Gupta, R., Haritsa, J., and Ramamritham, K. (1997a). More optimism about real-time distributed commit processing. In *proc. of the* 18^{th} *IEEE Real-Time Systems Symposium (RTSS'97)*, pages 123–133, San Francsico, California.

Gupta, R., Haritsa, J., and Ramamritham, K. (1997b). Revisiting Commit Processing in Distributed Database Systems. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 486–497, Tucson, Arizona, USA.

Gupta, R., Haritsa, J., Ramamritham, K., and Sechadri, S. (1996). Commit Processing in Distributed Real-Time Database Systems. In *proc. of the* 17^{th} *IEEE Real-Time Systems Symposium (RTSS'96)*, pages 220–229.

Gutknecht, O. and Ferber, J. (2000). MadKit : A generic multi-agent platform. In 4^{th} *Intl. Conf. on Autonomous Agents (AGENTS'00)*.

Hacid, M., Terzi, E., and Vakali, A. (2001). Querying XML with Constraints. In *Proceedings of the* 2^{nd} *International Conference on Internet Computing (IC'01), Las Vegas, USA.*

Hamdaoui, M. and Ramanathan, P. (1995). A Dynamic Priority Assignment Technique for Streams with (m, k)-$Firm$ Deadlines. *IEEE Tansactions on Computers*, 44(4) :1325–1337.

Hameurlain, A., Basex, P., and Morvan, F. (1996). *Traitement parallèle dans les bases de données relationnelles*. C´epadu`es Editions.

Hansson, J. and Son, S. (2001). Overload Management in RTDBs. In *Real-Time Database Systems : Architecture and Techniques*, chapter 10, pages 125–140. Kluwer Academic Publishers.

Hansson, J., Son, S., Stankovic, J., and Andler, S. (1998). Dynamic transaction scheduling and reallocation in overloaded real-time database systems. In *Proc. of the* 5^{th} *Conf. on Real-Time Computing Systems and Applications (RTCSA'98)*, pages 293–302. IEEE Computer Press.

Harder, T. and Rothermel, K. (1993). Concurrency Control Issues in Nested Transactions. *VLDB Journal*, 2(1) :39–74.

Haritsa, J., Carey, M., and Livny, M. (1990). On Being Optimistic about Real-Time Constraints. In 9th *ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*.

Haritsa, J., Carey, M., and Livny, M. (1992). Data Access Scheduling in Firm Real-Time Database Systems. *Real-Time Systems*, 4(3) :203–241.

Haritsa, J. and Ramamritham, K. (2000a). Adding PEP to Real-Time Distributed Commit Processing. In 21^{st} *Real-Time Systems Symposieum (RTSS'00)*, Orlando, Florida.

Haritsa, J. and Ramamritham, K. (2000b). Real-Time Databases in the New Millenium. *Real-Time Systems Journal*.

Haritsa, J., Ramamritham, K., and Gupta, R. (1997). Characterization and Optimization of Commit Processing Performance in Distributed Database Systems. In *ACM SIGMOD Intl. Conf. on Management of Data*.

Haritsa, J., Ramamritham, K., and Gupta, R. (2000). The PROMPT Real-Time Commit Protocol. *IEEE Transactions on Parallel and Distributed Systems*, 11(2) :160–181.

Haritsa, J., Ramamritham, K., and Gupta, R. (2001). Real-Time Commit Processing. In *Real-Time Database Systems : Architecture and Techniques*, pages 227–243. Kluwer Academic Publishers.

Haubert, J., Amanton, L., and Sadeg, B. (2004a). Un protocole sp´eculatif pour le contrˆole de concurrence et l'ordonnancement des transactions temps r´eel. In *Intl. Real-Time and embedded Systems Conference*, pages 205–222, Paris. T´ekn´ea.

Haubert, J., Sadeg, B., and Amanton, L. (2003a). ESCC : a new deadline-driven extension of the SCC protocol. In *Proceedings of IEEE Intl. Symp. on Parallel and Distributed Computing (ISPDC'03)*, pages 111–116, Ljubljana, Slovenia. IEEE Computer Society.

Haubert, J., Sadeg, B., and Amanton, L. (2003b). Improving the SCC protocol for real-time transaction concurrency control. In *Proceedings of IEEE ISSPIT'03*, Darmstadt, Germany.

Haubert, J., Sadeg, B., and Amanton, L. (2003c). Reducing the blocking time in weighted real-time distributed transaction management. In *Proceedings if EEE ISSPIT'03*, Darmstadt, Germany.

Haubert, J., Sadeg, B., and Amanton, L. (2004b). Extension d'un protocole de contrˆole de concurrence des transactions temps r´eel. *Intl. Journ. of ISDM (Special Issue)*, 13 :56–63.

Haubert, J., Sadeg, B., and Amanton, L. (2004c). (m,k)-Firm Real-Time Distributed Transactions. In *WIP-Proceedings of EuroMicro International Conference (ECRTS'04)*, Porto, Portugal.

Haubert, J., Sadeg, B., and Amanton, L. (2004d). Relaxing the Real-Time Constraints in Distributed Real-Time Management Systems. In *Proc. of the* 10^{th} *International Real-Time Computing Systems and Applications Conference (RTCSA'04)*, G¨otteborg, Sweden. Springer Verlag.

Haubert, J., Sadeg, B., Amanton, L., and Coma, R. (2004e). J-RADEx : un simulateur de SGBDTR convivial. *Intl. Journ. of ISDM (Special Issue)*, 13 :64–71.

Hong, D., Jonhson, T., and Chakravarty, S. (1993). Real-Time Transactions Scheduling : A Cost Conscious Approach. In *proc. of ACM SIGMOD Intl. Conf. on Management of Data*, pages 197–206.

Horvitz, E. J. (1990). Reasoning about Beliefs and Actions Under Computational Resource Constraints. In Henrion, M., Shachter, R. D., Kanal, L. N., and Lemmer, J. F., editors, *Uncertainty in Artifi cial Intelligence 5*, pages 301–324. Elsevier Science Publishers B.V., North Holland.

Huang, J. and Gruenwald, L. (1994). Logging Real-Time Main Memory Databases. In *Proc. of Intl. Computer Symposium*, pages 1291–1296.

Huang, J. and Gruenwald, L. (1996). An Update-Frequency-Valid-Interval Partition Checkpoint Technique for Real-Time Main Memory Databases. In 1^{st} *Workshop on Real-Time Databases(RTDB'96)*, pages 130–137.

Huang, J. and Stankovic, J. (1990). Concurrency Control in Real-Time Database System : Optimistic Scheme vs. Two-Phase Locking. Technical Report UM-CS-1990-066, University of Massachusetts.

Huang, J. and Stankovic, J. (1991). On Using Inheritance in Real-Time Databases. In *Proc. of Real-Time Systems Symposium (IEEE RTSS'91)*.

Huang, J., Stankovic, J., Ramamritham, K., Towsley, D., and Purimetla, B. (1991). Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes. In *Proc. of the* 17^{th} *Intl. Conf. on Very Large Databases (VLDB'91)*, Barcelona, Spain.

Jauhary, R., Carey, M., and Livny, M. (1990). Priority-hints : An algorithm for priority-based buffer management. In *Proc. of the* 16^{th} *conf. on Very Large Databases (VLDB)*, pages 708–721, Brisban, Australia. Morgan Kaufmann.

Jung, M. and Wesley, W. A Resilient Commit Protocol for Real Time Systems. In *Proc. of the* 6^{th}*IEEE Real-Time Systems Symposium*, page 1985.

Kamath, M. and Ramamritham, K. (1993). Performance Characteristics of Epsilon Serialisability with Hierarchical Inconsistency Bounds. In *Proceedings of the* 9^{th} *Intl. Conf. on Data Engineering*, pages 587–594. IEEE Computer Society Press.

Kang, K.-D., Son, S., and Stankovic, J. (2002). Service Differentiation in Real-Time Main Memory Databases. In 5^{th} *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (IEEE ISORC'02)*, Washington D.C.

Kao, B. and Cheng, R. (2001). Disk scheduling. pages 97–108, Real-Time Database Systems : Architecture and Techniques (Chapter). Kluwer Academic Publishers.

Kao, B. and Garcia-Molina, H. (1995). An Overview of Real-Time Database Systems. In *Advances in Real-Time Systems (ed. S.H. Son)*, pages 463–486. Prentice Hall.

Kao, B., Lam, K., and Adelberg, B. (2001). Updates and View Maintenance. pages 185–202, Real-Time Database Systems : Architecture and Techniques (Chapter). Kluwer Academic Publishers.

Kavi, K., editor (1992). *Real-Time Systems : Abstractions, Languages, and Design Methodologies*. IEEE Computer Society Press.

Kayan, E. and Ulusoy, Ö. (1999). An Evaluation of Real-Time Transaction Management Issues in Mobile Database Systems. *The Computer Journal (Special Issue on Mobile Computing)*, 42(6) :501–510.

Keller, A. M., Densmore, O., Huang, W., and Razavi, B. (1997). Zippering : Managing Intermittent Connectivity in DIANA. *ACM NOMAD (Mobile Networks and Applications)*, 2(4) :357–364.

Khrouf, K. and Soul´e-Dupuy, C. (2001). Conception d'entrepˆots de documents d´ecisionnels. In *Congres de l'INFormatique des Organisations et Systemes d'Information et de Décision (INFORSID'01) , Martigny - Suisse*, pages 387–401. Inforsid Edition.

Kim, W. and Srivastava, J. (1991). Enhancing Real-Time DBMS Performance with Multiversion Data and Priority Based Disk Sheduling. In *12^{th} Real-Time Systems Symposium*.

Kim, Y. and Son, S. (1994). Predictability and Consistency in Real-Time database systems. In *Principles of Real-Time Systems*, pages 502–524. Prentice-Hall.

Kim, Y. and Son, S. (1997). Developing a real-time database : The StarBase experience. In A. Bestavros, K. L. and Son, S., editors, *Real-Time Database Systems : Issues and Applications*, pages 305–324. Kluwer Academic Publishers, Boston, Mass.

Kiviniemi, J., Niklander, T., Porkka, P., and Raatikainen, K. (1997). Transaction Processing in The RODAIN Real-Time Database System. In *Real-Time Database and Information Systems*, pages 355–377. Kluwer Academic Publishers.

Kung, H. and Robinson, J. (1981). On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems*, 6(2) :213–226.

Kuo, T. and Mok, A. (1993). SSP : a Semantics-Based Protocol for Real-Time Data Access. In *IEEE Real-Time Systems Symp. (RTSS'93)*, pages 76–86.

Kuo, T.-W., Liang, M.-C., and Shu, L.-C. (2001). Abort-Oriented Concurrency Control for Real-Time Databases. *IEEE Transactions on Computers*, 50(7) :660–673.

Kuo, T.-W., Wu, J., and Hsih, H. (2002). Real-Time Concurrency Control in a Multiprocessor Environment. *IEEE Transactions on Parallel and Distributed Systems*, 13(6) :659–671.

Lam, K. and Yan, W. (1998). On Using Similarity for Concurrency Control in Real-Time Database Systems. *Journal of Systems and Software*, 43(3) :223–232.

Lam, K.-W., Lee, V., and Hung, S.-L. (2000). Transaction Scheduling in Distributed Real-Time Systems. *Real-Time Systems*, 19 :169–193.

Lam, K.-W., Son, S. H., and Hung, S.-L. (1997a). A Priority Ceiling Protocol with Dynamic Adjustment of Serialization Order. In *Proc. of ICDE*, pages 552–561.

Lam, K.-Y., Cao, J., Pang, C.-L., and Son, S. (1997b). Resolving Conflicts with Committing Transactions in Distributed Real-time Databases. In *Proc. of ICECCS*.

Lam, K.-Y. and Kuo, T.-W. Mobile distributed real-time database systems. In *Real-Time Database Systems : Architecture and Techniques*, chapter 18, pages 245–258. Kluwer Academic Publishers.

Lam, K.-Y., Pang, C.-L., Son, S., and Cao, J. (1999). Resolving Executing-Committing Conflicts in Distributed Real-Time Database Systems. *journal of Information Systems*, 42(8) :674–692.

Laurini, R. (2003). *CASSINI - SIGMA, Systèmes d'Information Géographique : Méthodologies et Applications", Groupement de Recherche 2340 du CNRS*. http ://cassini.univ-lr.fr/.

Lee, S. K., Hwang, C.-S., and Yu, H. (1999). Supporting Transactional Cache Consistency in Mobile Database Systems. In *MobiDE'99*.

Lee, V. C. and Lam, K.-W. (2000). Conflict Free Transaction Scheduling Using Serialisation Graph for Real-Time Databases. *Journal of Systems and Software*, 55(1) :57–65.

Lehr, M., Kim, Y., and Son, S. (1995). Managing contention and timing constraints in a real-time database system. In 16^{th} *IEEE RTSS*, Pisa, Italy.

Lementec, J. and Brunessaux, S. (1992). ATOME-TR, a real-time control for BlackBoard scheduling. In *Proceedings ECAI*, pages 255–256.

Lesser, V., Palvin, J., and Durfee, E. (1988). Approximate processing in real-time problem solving. *Artificial Intelligence Magazine*, 9(1) :49–61.

Lin, K.-J. and Peng, C.-S. (1996). Enhancing External Consistency in Real-Time Transactions. In *Special Section on RTDBS of ACM SIGMOD*.

Lin, Y. and Son, S. H. (1990). Concurrent Control in Real-Time Databases by Dynamic Adjustment of Serialization Order. In *11^{th} IEEE Real-Time Systems Symposium (RTSS'90)*, Orlando, Florida.

Lindstr¨om, J. (2003). *Optimistic Concurrency Control Methods for Real-Time Database Systems*. PhD thesis, University of Helsinki, Finland.

Liu, C. and Leyland, J. (1973). Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment. *Journal of the ACM*, 1(20) :46–61.

Liu, G., Mok, A., and Konana, P. (1998). A unified approach for specifying timing constraints and composite events in active real-time database systems. In *Proc. of the 4^{th} Real-Time Technologie and Applications Symposium (RTAS'98)*.

Liu, J., Lin, K., Shih, W., Yu, A., Chung, J., and Zhao, W. (1991). Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5) :58–68.

Liu, J. and Lin, K.-J. (1994). Use of Imprecise Computation to Enhance Dependability of Real-Time Systems. In *Foundations of Dependable Computing : Paradigms for Dependable Applications*. Koob and Lau eds. Kluwer Academic Publishers.

Liu, Y., Liao, G.-Q., Li, G., and Xia, J. (2002). Relaxed Atomic Commit for Real-Time Transactions in Mobile Computing Environment. In *Proc. of WAIM*, pages 397–408.

Locke, D. (2001). Applications and system characteristics. In *Real-Time Database Systems : Architecture and Techniques*, pages 17–26. Kluwer Academic Publishers.

Lortz, V., Shin, K., and Kim, J. (2000). MDARTS : A Multiprocessor Database Architecture for Hard Real-Time Systems. *IEEE Trans. on Knowledge and Data Engineering*, 12(4) :621–644.

Lortz, V. B. (1994). *An object-oriented real-time database system for multiprocessors*. PhD thesis, University of Michigan.

Lu, C., Abdelzaher, T. F., Stankovic, J. A., and Son, S. H. (2001). A feedback control architecture and design methodology for service delay guarantees in web servers. Technical Report CS-2001-06.

Lu, C., Stankovic, J., and Song, S. (2002). Feedback Control Real-Time Scheduling : Framework, Modeling, and Algorithms. *Real-Time Systems Journal*, 23(2) :85–126.

Madria, S., Maheshwari, S., Chandra, B., and Bhargava, B. (1998). An Open and Safe Nested Transaction Model : Concurrency and Recovery. *Journal of Systems and Software*, 55 :151–165.

Mainguenaud, M. (2000). Query Models and Languages for Geographical Information Systems. *Lecture Notes in Computer Science*, 1929 :511–520.

Mammeri, Z. (1997). Synchronisation d'horloges dans les syst`emes r´epartis. In *Ecole d'été temps réel*, pages 102–115.

Mammeri, Z. (1998). Expression et d´erivation des contraintes temporelles dans les applications temps r´eel. *APII - JESA*, 32(5-6) :609–644.

Martinez, J. (1992). Contribution aux probl`emes de contrˆole de concurrence et de reprise dans les bases de donn´ees `a objets. Technical report, PhD thesis, Universit´e de Nantes, http ://www.sciences.univ-nantes.fr/info/perso/permanents/martinez/Researches /CCR/These92.html.

Mellin, J. (2000). Predictable and efficient memory management for composite events. In *Proc. of the Workshop on Real-Time Programming (WRTP'00)*, Palma, Mallorca.

Mellin, J., Eriksson, J., and Andler, S. (2001). Reactive Mechanisms. In *Real-Time Database Systems : Architecture and Techniques*, chapter 13, pages 171–184. Kluwer Academic Publishers.

Menasce, D. and Nakanishi, T. (1982). Optimistic versus pessimistic concurrency control mechanisms in database management systems. *Information Systems*, 7(1).

Milo, T., Abiteboul, S., Amann, B., Benjelloun, O., and Ngoc, F. D. (2003). Exchanging intensional XML data. In *Proceedings of ACM SIGMOD international conference on on Management of data - SESSION : Data integration and sharing*, pages 289–300, San Diego, California.

Moss, J. (1985). Nested Transactions : An Approach to Reliable Distributed Computing. In *The MIT Press*, .

Mostefaoui, A. and Raynal, M. (1993a). Causal Multicasts in Overlapping Groups : Towards a Low Cost Approach. In *IEEE Intl. Conf. on Future Trends of Distributed Systems*, pages 136–142, Lisboa, Portugal.

Mostefaoui, A. and Raynal, M. (1993b). Definition and Implementation of a Flexible Communication Primitive for Distributed Programming. Technical Report 765, IRISA, Rennes, France.

Mostefaoui, A. and Theel, O. (1996). Reduction of Timestamp Sizes for Causal Event Ordering. Technical Report 1062, IRISA, Rennes, France.

Mouaddib, A. (1997). Progressive Negotiation For Time-Constrained Autonomous Agents. In *Proceedings of the* 1^{st} *International Conference on Autonomous Agents*, pages 8–15, Marina del Rey, CA, USA. ACM Press, New York.

Mouaddib, A. (1999). Anytime Coordination for Progressive Planning Agents. In *Proceedings of the* 6^{th} *National Conference on Artificial Intelligence and* 11^{th} *Conference on Innovative Applications of Artificial Intelligence*, pages 564–569, Orlando, Florida. AAAI Press / The MIT Press.

Mouaddib, A. and Zilberstein, S. (1998). Optimal Scheduling of Dynamic Progressive Processing. In *Proceedings of the* 13^{th} *European Conference on Artificial Intelligence*, pages 499–503, Brighton, UK. John Wiley and Sons, Chichester.

Mullender, S. (1993). *Distributed systems*. Addison-Wesley Publishers Ltd.

Nakazato, H. (1993). *Issues on Synchronizing and Scheduling Tasks in Real-Time Database Systems*. PhD thesis, University of Illinois at Urbana-Champaign, USA.

O'Neil, P., Ramamritham, K., and Pu, C. (1995). A Two-Phase Approach to Predictably Scheduling Real-Time Transactions. In *Performance of Concurrency Control Mechanisms in Centralized Database Systems, V. Kumar ed.*, pages 494–522. Prentice-Hall.

Ozsoyoglu, G. and Snodgrass, R. (1995). Temporal and Real-Time Databases : A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4) :513–532.

Özsu, T. and Valduriez, P. (1999). *Principles of Distributed Database Systems*. Prentice Hall International Inc., international edition. Second edition.

Phatak, S.-H. and Badrinath, B. (1999). Multiversion Reconciliation for Mobile Databases. In *ICDE'99*, pages 582–589.

Pitoura, E. and Chrysanthis, P. (1999). Exploiting Versions for Handling Updates in Broadcast Disks. In *Intl. Conf. on Very Large Databases (VLDB'99)*, pages 114–125.

Pradhan, D., Krishna, P., and Vaidya, N. (1995). Recoverable Mobile Environments : Design and Trade-off Analysis. Technical Report 95-053, dept. of Comp. Science, Texas A& M University.

Prakash, R., Raynal, M., and Singhal, M. (1997). An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments. *Journal of Parallel and Distributed Computing*, 41(2) :190–204.

Prichard, J., DiPippo, L., Packham, J., and Fay-Wolfe, V. (1994). RTSORAC : A Real-Time Object-Oriented Database Model. In Springer-Verlag, editor, *LNCS-Proc. of the* 5^{th} *Intl. Conf. on Database and Expert Systems Applications (DEXA'94*, pages 601–610, London, UK.

Prichard, J. and Fortier, P. (1997). *Standardizing real-time databases - RTSQL*, chapter In Real-Time Database and Information Systems, pages 289–310. Kluwer Academic Publihers.

Pu, C. (1991). Generalized Transactions Processing with Epsilon Sarializability. In *Proceedings of* 4^{th} *International Workshop on High Performance Transactions Systems*, Asilomar, California.

Pu, C. (1993). Relaxing the Limitations of Serializable Transactions in Distributed Systems. In *Proceeding of ACM SIGOPS European Workshop and Operation*.

Pu, C., Hseush, W., Kaiser, G. E., Wu, K. L., and Yu, P. S. (1997). Divergence Control for Distributed Database Systems. Technical Report Technical Report No. OR 97006, Department of Computer Science and Engineering, Columbia University, New York NY 10027.

Pu, C. and Leff, A. (1991). Execution Autonomy in Distributed Transaction Processing. Technical Report Technical Report No. CUCS-024-91, Department of Computer Science and Engineering, Columbia University, New York NY 10027.

Pu, C. and Leff, A. (1992). Autonomous Transaction Execution with Epsilon Sarializability. In *Proceeding of 1992 RIDE Workshop on Transaction and Query Processing*. IEEE Computer Society.

Qin, B. and Liu, Y. (2003). High performance distributed real-time commit protocol. *Journal of Systems and Software*, 68(2) :145–152.

Ramakrishnan, R. and Gehrke, J. (2000). *Database Management Systems*. Mac Graw Hill.

Ramamritham, K. (1993). Real-time databases. *Distributed and Parallel Databases (Invited Paper)*, 1(2) :199–226.

Ramamritham, K. (1996). Where do Time Constraints Come From and Where Do They Go ?). *International Journal of Database Management (Invited Paper)*, 7(2) :4–10.

Ramamritham, K. and Chrysanthis, P. K. (1993). In Search of Acceptability Criteria : Database Consistency Requirements and Transaction Correctness Properties. In Ozsu, T., Dayal, U., and Valduriez, P., editors, *Distributed Object Management*, pages 211–229. Morgan Kaufmann Publisher, San Mateo, California.

Ramamritham, K. and Chrysanthis, P. K. (1996). A Taxonomy of Correctness Criteria in Database Applications. *Journal of Very Large Databases*, 4(1) :85–97.

Ramamritham, K. and Chrysanthis, P. K. (1997). *Advances in Concurrency Control and Transaction Processing - Executive Briefi ng*. IEEE Computer Society Press.

Ramamritham, K. and Pu, C. (1991). A Formal Characterization of Epsilon Serialisability. Technical Report Technical Report CUCS-044-91, Department of Computer Science, Columbia, University.

Ramamritham, K. and Pu, C. (1995). A Formal Characterization of Epsilon Serialisability. *IEEE Transaction Knowledge and Data Engineering*, 7(6) :997–1007.

Ramamritham, K. and Soparkar, N. (1996). DART'96 - Databases : Active and Real-Time (Concepts meet Practice). *ACM*.

Ramamritham, K. and Stankovic, J. (1994). Scheduling algorithms and operating systems support for real-time systems). *IEEE*, 82(1) :55–67.

Rammanathan, P. and Agrawal, P. (1998). Adapting Packet Fair Queing Algorithms to Wireless Networks. In *Proceeding of ACM/IEEE Intl. Conf. on Mobile Computing and Networking*.

Raynal, M. and Schiper, A. (1995). From causal consistency to sequentiel consistency in shared memory systems. Technical Report 926, IRISA, Campus Universitaire de Beaulieu - 35042 Rennes Cedex - France.

Raynal, M. and Singhal, M. (1996). Logical time : capturing causality in distributed systems. *Computer Magazine, IEEE Computer Society*, pages 49–65.

Raynal, M., Thia-Kime, G., and Ahamad, M. (1996). An adaptative architecture for causally consistent distributed services. Technical Report IP 1039, IRISA.

Resende, R. and Abbadi, A. E. (1994). On the Serializability Theorem for Nested Transactions. *Information Processing Letters*, 50(4).

Restifi car, A. C. (1996). Probabilistic diagnosis as an update problem. In *PRICAI Workshops*, pages 256–266.

Rodrigues, L., Baldoni, R., Anceaume, E., and Raynal, M. (2000). Deadline-constrained causal order. In *The* 3^{rd} *IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'00)*.

Sadeg, B., Amanton, L., and Bouzefrane, S. (2001a). A Causal-Phase Protocol to Order Soft Real-Time Transactions. In *Proc. of the* 26^{th} $IEEE$ *Conference on Local Compter Networks (IEEE LCN'01)*, Tampa, Florida, USA. IEEE Computer Society.

Sadeg, B., Amanton, L., and Bouzefrane, S. (2001b). Disconnection Tolerance in Real-Time Mobile Databases. *Special Issue of INFORMATION : An International Journal*, 4(4) :445–456.

Sadeg, B., Amanton, L., and Bouzefrane, S. (2002a). Real-Time Transactions Management in a Distributed Environment. In 20^{th} *IASTED Applied Informatics : Databases and Applications*, Innsbruck, Austria.

Sadeg, B. and Bouzefrane, S. (1999). Enhancing Concurrency Control and Scheduling of Real-Time Transactions. In *IEEE WIP-Intl. Real-Time System Symposium (RTSS'99)*, pages 80–84, Phoenix, Arizona, USA.

Sadeg, B. and Bouzefrane, S. (2000a). Gestion des Transactions temps r´eel `a ´ech´eances non strictes. In *Proc. of the 8^{th} Intl. Conf. on Real-Time and embedded Systems (RTS'2000)*, pages 232–246, Paris, France. T´ekn´ea.

Sadeg, B. and Bouzefrane, S. (2000b). Relaxing Correctness Criteria in Real-Time DBMSs. In *Intl. Conf. on Computers and Their Applications (CATA'2000)*, pages 64–67, New-Orleans, Louisiana, USA.

Sadeg, B., Bouzefrane, S., and Amanton, L. (2001c). Management of Real-Time Query Transactions in a Mobile Computing Environment. In *9^{th} Intl. Conf. on Real-Time and embeddeed Systems (RTS'2001)*, Paris, France.

Sadeg, B., Bouzefrane, S., and Amanton, L. (2002b). $D\_ANTICIP$: a Protocol Suitable for Distributed Real-Time Transactions. In *Proc. of the 49^{th} Intl. Conference on Enterprise Information Systems - Databases and Informations Systems topic (ICEIS'02)*, volume 1, pages 171–178, Universidad de Castilla-La mancha, Ciudad Real - Spain. In collaboration with AAAI.

Sadeg, B. and Haubert, J. (2002). Improving the Commit Protocol Process for Distributed Real-Time Transactions. In *7^{th} Maghrebian Conference on Computer Technology ($MCSEAI'02$)*, Annaba, Algeria.

Sadeg, B., Haubert, J., and Amanton, L. (2003). Tradding Precision for Timeliness in Real-Time Database Systems. In *Intl. Conference on Enterprise Information Systems - Post. in Databases and Information Systems topic (ICEIS'02)*, Angers, France.

Sadeg, B., Haubert, J., Amanton, L., and Bouzefrane, S. (2002c). WEP : an adaptation of an 1-PC Protocol to Distributed Real-Time Transactions. In *IEEE ISSPIT'02*, Marrakech, Marocco.

Sadeg, B., Saad-Bouzefrane, S., and Amanton, L. (2001d). A-ANTICIP : A protocol to enhance distributed real-time transactions performances. In *Rap. Tech. RAP-3-01*, Universit´e du Havre.

Semghouni, S., Sadeg, B., Berred, A., and Amanton, L. (2003). Conception d'un simulateur de SGBD temps r´eel. In *Rap. Tech. RAP-4-03*, Universit´e du Havre.

Semghouni, S., Sadeg, B., Berred, A., and Amanton, L. (2004). Approche pour une ´etude probabiliste des protocoles de contrˆole de concurrence dans les base de donn´ees temps r´eel. In *MajecSTIC'04*, Calais, France.

Sha, L., Rajkumar, R., and Lehoczky, J. (1987). Priority-Inheritance Protocols : An Approach to Real-Time Synchronization. Technical report, Tech. Rep. CMU-CS-87-181 - Carnegie Mellon Univ.

Sha, L., Rajkumar, R., and Lehoczky, J. (1990). Priority-Inheritance Protocols : An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 9(39) :1175–1185.

Sha, L., Rajkumar, R., and Lehoczky, J. (1998). Concurrency Control for Distributed Real-Time Databases. *ACM Sigmod Record*, 17(1) :82–98.

Sha, L., Rajkumar, R., and Sathaye, S. (1994). Generalized Rate-Monotonic Scheduling Policy : a Framework for Developping Real-Time Systems. In *Proc. of the IEEE*, pages 68–82.

Sha, L., Rajkumar, R., Son, S., and Chang, C. (1991). A real-time locking protocol. *IEEE Transactions on Computers*, 7(40).

Shakkottai, S. and Srikant, R. (1999). Scheduling Real-Time Traffi c with Deadlines over a Wireless channel. In *Proc. of ACM Wireless Mobile Multimedia*.

Shu, L. (2003). On avoiding remote blocking via real-time concurrency control protocols. *Journal of Systems and Software*, 68(2) :121–136.

Singhal, V. and Smith, A. (1994). Characterization of Contention in Real Relational Databases. Technical Report Report Num. UCB/USD 94/801.

Sivasankaran, R., Ramamritham, K., and Stankovic, J. (2001). System Failure and Recovry. In *Real-Time Database Systems - Architecture and Techniques*, pages 109–124. Kluwer Academic Publishers.

Sivasankaran, R., Stankovic, J., Towsley, D., Purimetla, B., and Ramamritham, K. (1996). Priority Assignement in Real-Time Active Databases. *Journal of Very Large Databases*, 5(1) :19–34.

Snodgrass, R. (1987). The temporal query languahe TQuel. *ACM Transactions on Database Systems (TODS)*, 12(2) :247–298.

Son, S. H., Georges, D., and Kim, Y. (1993). Developping a database system for time-critical applications. In *Intl. Conf. on Database and Expert Systems (DEXA'93)*, pages 313–324, Prague, Czech.

Son, S. H. and Kouloumbis, S. (1993). A Token-Based Synchronization Scheme Using Epsilon-Serialisability and its Performance for Real-Time Distributed Database Systems. In *Proceedings of 3^{rd} International Symposium on Database Systems for Advanced Applications*, Korea.

Son, S. H., Lee, J., and Lin, Y. (1992). Hybrid protocols using dynamic adjustement of serialization order for real-time concurrency control. *Journal of RTS*, 4(3) :269–276.

Soparkar, N. (1994). Adaptive Commitment for Distributed Real-Time Transactions. In *Proceedings of CIKM'94*, pages 187–194.

Stankovic, J. (1988). Misconceptions about Real-Time Transactions : A Serious Problem for Next Generation Systems. *IEEE Computer*, 21(10) :10–19.

Stankovic, J., Ramamritham, K., and Towsley, D. (1991). Scheduling in Real-Time Transaction Systems. *Foundations of Real-Time Computing : Scheduling and Resource Allocation (Chapter 18), A.V. Tilborg and G. Koob ed.*, pages 157–184.

Stankovic, J., Son, S., and Hansson, J. (1999). Misconceptions about Real-Time Databases. *IEEE Computer*, 32(6) :29–36.

Stankovic, J., Son, S., and Liebeherr, J. (1997). BeeHive : Global Multimedia Database Support for Dependable, Real-Time Applications. In *Real-Time Database and Information Systems*, pages 409–422. Kluwer Academic Publishers.

Stankovic, J., Spuri, M., Natale, M. D., and Buttazzo, G. (1995). Implications of Classical Scheduling Results in Real-Time Systems. *IEEE Computer*, 28(6) :16–25.

Stankovic, J. and Zhao, W. (1988). On Real-Time Transactions. In *ACM Sigmod Record*, volume 17, pages 4–18.

Tansel, A., Clifford, J., Gadia, V., Segev, A., and Snodgrass, R. (1993). Temporal databases : theory, design and implementation. The Benjamen/Cummings Publishing Company, Redwood city, California.

Theel, O. and Raynal, M. (1996). Static and dynamic adaptation of transactional consistency. Technical Report 2999, INRIA.

Tia, T.-S., Deng, Z., Shankar, M., storch, M., Sun, J., Wu, L.-C., and Liu, J.-S. (1995). Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proc. of RTAS sysmposium*, Chicagi, Illinois.

Tokuda, H. and Mercer, C. (1989). ARTS : A Ditributed Real-Time Kernel. *ACM Operating System Review*, 23(3) :29–53.

Tokuda, H., Nakajima, T., and Rao, P. (1990). Real-Time Mach : Toward Predictable Real-Time Systems. In *USENIX Mach Workshop*.

Toussaint, J., Simonot-Lion, F., and Thomesse, J. (1997). Time Constraints Verifi cation Methods Based on Time Petri Nets. In *Proc. of FTDCS*.

Tsang, M. K., Wu, K. L., and Yu, P. S. (1997). Multiversion Divergence Control of Time Fuzziness. Technical Report OR 97291-1000, Department of Computer Science and Engineering, Yorktown Heights, NY 10598.

Tseng, F.-C., Chen, A., and Yang, W.-P. (1992). A Probabilistic Approach to Query Processing in Heterogeneous Database Systems. In 2^{nd} *International Woskshop on Research Issues on Data Engineering : Transactions and Query Processing*, pages 176–183.

Ullman, J. (1980). *Principles of Database Systems*. Computer Science Press, Inc., Maryland.

Ulusoy, Ö. (1993a). An Approximate Analysis of a Real-Time Database Concurrency Control Protocol via Markov Modeling. *Performance Evaluation Review*, 20(3).

Ulusoy, Ö. (1993b). Lock-Based Concurrency Control in Distributed Real-Time Database Systems. *Journal of Data Management*, 4(2).

Ulusoy, Ö. (1994a). Applying Database Technology to Real-Time Systems. *DEXA'94 - Lecture Notes in Computer Science (Springer Verlag)*, 856.

Ulusoy, Ö. (1994b). Processing Real-Time Transactions in Replicated Database Systems. *International Journal of Parallel and Distributed Databases*, 2(4) :405–436.

Ulusoy, Ö. (1995a). A Study of Two Transaction Processing Architecture for Distributed Real-Time Database Systems. *Journal of Systems and Software*, 31(2).

Ulusoy, Ö. (1995b). An Annoted Bibliography on Real-Time Database Systems. *ACM SIGMOD Record*, 24(4).

Ulusoy, Ö. (1995c). Research Issues in Real-Time Database Systems. *Information Sciences*, 87(1-3) :123–151.

Ulusoy, Ö. (1998a). Analysis of Concurrency Control Protocols for Real-Time Database Systems. *Information Sciences*, 111(1-4) :19–47.

Ulusoy, Ö. (1998b). Real-Time Data Management for Mobile Computing. In *Intl. Workshop on Issues and Applications of Database Technology (IADT-98)*, pages 233–240.

Ulusoy, Ö. (1998c). Transaction Processing in Distributed Active Real-Time Database Systems. *Journal of Systems and Software (Special Issue on Active Real-Time Databases)*, 42(3).

Ulusoy, Ö. and Belford, G. G. (1993). Real-Time Transaction Scheduling in Database Systems. *Information Systems Journal*, 18(8).

Ulusoy, Ö. and Buchman, A. (1996). Exploiting Main Memory DBMS Feature to Improve Real-Time Concurrency Control Protocols. *ACM SIGMOD Record*, 25(1) :123–151.

Ulusoy, Ö. and Buchman, A. (1998). A Real-Time Concurrency Control Protocol for Main-Memory Database Systems. *Information Systems*, 23(2).

Vassiliadis, P., Bouzeghoub, M., and Quix, C. (1999). Towards Quality-Oriented Data Warehouse Usage and Evolution. *Lecture Notes in Computer Science*, 1626.

Wang, J. Z., Wiederhold, G., Firschein, O., and Wei, S. X. (1997). Content-based image indexing and searching using daubechies' wavelets. *Int. J. on Digital Libraries*, 1(4) :311–328.

Weinberger, P. and Mitra, D. (1984). Probabilistic models of database locking : solutions, computational algorithms, and asymptotics. *Journal of the ACM*, 31 :855–878.

Wells, L. D., Blakely, J., and Thompson, C. (1992). Architecture of an open object-oriented database management system. *IEEE Computer*, 25(10) :74–82.

Wong, J. S. K. and Mitra, S. (1996). A Nonblocking Timed Atomic Commit Protocol for Distributed Real-Time Database Systems. *Journal of Systems and Software*, 34 :161–170.

Woods, W. (1970). Transition Network Grammars for Natural Language Analysis. *Communication of the ACM*, 13 :591–602.

Wu, K. L., Yu, P. S., and Pu, C. (1992). Divergence Control Algorithms for Epsilon Serialisability. In *Proceedings of* 8^{th} *Intl. Conf. on Data Engineering*, pages 506–515. IEEE Computer Society.

Wu, L., Faloutsos, C., Sycara, K., and Payne, T. (2001). Multimedia Queries by Example and Relevance Feedback. *IEEE Data Engineering Bulletin*, 24(3) :14–21.

Xiong, M. and Ramamritham, K. (1997). Specifi cation and Analysis of Transactions in Real-Time Active Databases. In *Real-Time Database and Information Systems, A. Bestavros and V. Fay-Wolfe ed.*, pages 327–354. Kluwer Academin Publishers.

Xiong, M., Sivasankaran, R., Ramamritham, K., Stankovic, J., and Towsley, D. (1997). Scheduling Access to Temporal Data in Real-Time Databases. In *Real-Time Database Systems : Issues and Applications, S-H. Son, K-J. Lin and A. Bestavros ed.*, pages 167–192. Kluwer Academin Publishers.

Xuan, P., Gonzalez, O., Fernandez, J., and Ramamritham, K. (1997). Broadcast on Demand : Effi cient and Timely Dissemination of Data in Mobile Environments. In *Proc. of the* 3^{rd} *IEEE Real-Time Technology Application Symposium (RTAS)*.

Yavatkar, R. (1992). MCP : a Protocol for Coordination and Temporal Synchronization in Multimedia Collaborative Applications. In 12^{th} *IEEE Conf. on Distributed Computing Systems*, pages 606–613.

Yoon, Y.-I., Han, M., and Cho, J.-H. (1996). Real-Time Commit Protocol for Distributed Real-Time Database Systems. In *Proc. of ICECCS*.

Zhang, Y. and Yang, Y. (1994). On Operation synchronization in Cooperative Editing Environments. In *IFIP Transaction A-54, Proc. of Intl. Conf. on Business Process Re-engineering*, pages 635–644.

Zhao, W., Ramamritham, K., and Stankovic, J. (1987). Preemptive Scheduling under Time and Resource Constraints. *IEEE Transactions on Computers*, 36(8) :949–960.

Zhou, L., Rundensteiner, E., and Shin, K. (1995). OODB support for real-time open-architecture controllers. In *Proceedings of* 4^{th} *Intl. Conf. on Database Systems for Advanced Applications*, pages 206–213.

Zhou, L., Rundensteiner, E., and Shin, K. (1997). Schema evolution of an object-oriented real-time database system for manufacturing automation. *IEEE Trans. on Knowledge and Data Engineering*, 9(6) :956–977.

Zhou, L., Shin, K. G., Rundensteiner, E., and Soparkar, N. (1996). Probabilistic Real-Time Data Access with Interval Constraints. In 1^{st} *Intl. Workshop on Real-Time Database Systems*, New Beach, CA.

Zilberstein, S. and Russell, S. (1993). Anytime Sensing, Planning and Action : A Practical Model for Robot Control. In *13th International Joint Conference on Artifi cial Intelligence*, Chamb´ery.

Zilberstein, S. and Russell, S. (1996). Optimal Composition of Real-Time Systems. *Artifi cial Intelligence*, 82(1-2) :181–213.

()

Liste des URL de quelques références bibliographiques

(Abdallah and Pucheral, 1998) : http ://citeseer.ist.psu.edu/abdallah98singlephase.html
(Abdallah et al., 1998) : http ://citeseer.ist.psu.edu/abdallah98onephase.html
(Acharya et al., 1994) :http ://citeseer.ist.psu.edu/badrinath94structuring.html
(Al-Houmaily and Chrysanthis, 1999) : http ://citeseer.ist.psu.edu/266891.html
(Amirijoo et al., 2003a) : http ://www.ida.liu.se/ meham/publications.shtml
(Amirijoo et al., 2003b) : http ://www.ida.liu.se/ meham/publications.shtml
(Amirijoo et al., 2003c) : http ://www.ida.liu.se/ meham/publications.shtml
(Andler et al., 1996) : http ://citeseer.ist.psu.edu/andler96deeds.html
(Atlas and Bestavros, 1998) : http ://citeseer.ist.psu.edu/atlas98statistical.html
(Audsley et al., 1991) : http ://citeseer.ist.psu.edu/audsley91hard.html
(Aydin et al., 2001) : http ://citeseer.ist.psu.edu/496257.html
(Banerjee and Chrysanthis, 1997) : http ://citeseer.ist.psu.edu/banerjee97performance.html
(Berndtsson and Hansson, 1996) : http ://citeseer.ist.psu.edu/berndtsson96workshop.html
(Bestavros and Braoudakis, 1996) : http ://citeseer.ist.psu.edu/108124.html
(Collet et al., 1994) : http ://citeseer.ist.psu.edu/collet94naos.html
(Datta et al., 1997) : http ://citeseer.ist.psu.edu/datta97adaptive.html
(Ezhilchelvan et al., 1995) : http ://citeseer.ist.psu.edu/ezhilchelvan95newtop.html
(Gatziu and Dittrich, 1993) : http ://citeseer.ist.psu.edu/gatziu93events.html
(Gehani et al., 1993) : http ://citeseer.ist.psu.edu/gehani94compose.html
(Gien, 1995) : http ://citeseer.ist.psu.edu/gien95evolution.html
(Guerraoui and Schiper, 1995) : http ://citeseer.ist.psu.edu/guerraoui95decentralized.html
(Gupta et al., 1996) : http ://citeseer.ist.psu.edu/gupta96commit.html
(Gupta et al., 1997a) : http ://citeseer.ist.psu.edu/gupta97more.html
(Gupta et al., 1997b) : http ://citeseer.ist.psu.edu/gupta97revisiting.html
(Hamdaoui and Ramanathan, 1995) : http ://citeseer.ist.psu.edu/hamdaoui94dynamic.html
(Hansson et al., 1998) : http ://citeseer.ist.psu.edu/430539.html
(Haritsa et al., 1992) : http ://citeseer.ist.psu.edu/haritsa92data.html
(Haritsa and Ramamritham, 2000a) : http ://citeseer.ist.psu.edu/haritsa00adding.html
(Haritsa et al., 2000) : http ://citeseer.ist.psu.edu/haritsa00prompt.html
(Hong et al., 1993) : http ://citeseer.ist.psu.edu/hong93realtime.html
(Horvitz, 1990) : http ://citeseer.ist.psu.edu/horvitz87reasoning.html
(Huang and Gruenwald, 1996) : http ://citeseer.ist.psu.edu/huang96updatefrequencyvalidinterval.html
(Kamath and Ramamritham, 1993) : http ://citeseer.ist.psu.edu/kamath93performance.html
(Kang et al., 2002) : http ://citeseer.ist.psu.edu/522543.html
(Kayan and Ulusoy, 1999) : http ://citeseer.ist.psu.edu/kayan99evaluation.html
(Keller et al., 1997) : http ://citeseer.ist.psu.edu/keller97zippering.html
(Kim and Son, 1994) : http ://citeseer.ist.psu.edu/son93predictability.html
(Kim and Son, 1997) : http ://citeseer.ist.psu.edu/kim97developing.html
(Lehr et al., 1995) : http ://citeseer.ist.psu.edu/lehr95managing.html
(Lu et al., 2001) : http ://citeseer.ist.psu.edu/577876.html
(Mostefaoui and Raynal, 1993a) : http ://citeseer.ist.psu.edu/mostefaoui93causal.html
(Mouaddib and Zilberstein, 1998) : http ://citeseer.ist.psu.edu/mouaddib98optimal.html
(Ozsoyoglu and Snodgrass, 1995) : http ://citeseer.ist.psu.edu/ozsoyoglu95temporal.html
(Prakash et al., 1997) : http ://citeseer.ist.psu.edu/prakash97adaptive.html
(Ramamritham, 1993) : http ://citeseer.ist.psu.edu/ramamritham93realtime.html
(Stankovic et al., 1995) : http ://citeseer.ist.psu.edu/stankovic94implications.html
(Tokuda et al., 1990) : http ://citeseer.ist.psu.edu/tokuda90realtime.html
(Ulusoy, 1994b) : http ://citeseer.ist.psu.edu/ulusoy94processing.html
(Ulusoy, 1998a) : http ://citeseer.ist.psu.edu/ulusoy98analysis.html

(Xiong et al., 1997) : http ://citeseer.ist.psu.edu/xiong97scheduling.html
(Yavatkar, 1992) : http ://citeseer.ist.psu.edu/yavatkar91mcp.html
(Zhou et al., 1995) : http ://citeseer.ist.psu.edu/zhou95oodb.html
(Zhou et al., 1997) : http ://citeseer.ist.psu.edu/158695.html
(Zilberstein and Russell, 1996) : http ://citeseer.ist.psu.edu/zilberstein96optimal.html

Acronymes

Sigle	Num. chapitre ou paragraphe
– 1PC, 2PC, 3PC : {1, 2, 3}-Phase Commit	4
– 2PL : Two-Phase Locking, verrouillage `a 2 phases	3.2.1 ; 1.1.2
– 2PL-HP : Two-Phase Locking-Hight Priority	3.2.1.1
– ACID : Atomicit´e, Coh´erence, Isolation, Durabilit´e	1
– ACP : Atomic Commitment Protocol	4
– ARS : Action de R´esolution de Surcharge	2.5.1
– ARTCC : Air Route Traffi c Control Center	2.4.2
– BS : Basic Scheduler (Ordonnanceur de Base)	7
– CC : Contrˆoleur de Concurrence	3-7
– CCA (protocole de contrˆole de concurrence) : Cost Conscious Approach	3.2.2
– CC&O : Contrˆole de Concurrence et Ordonnancement	2
– DeeDS : Distributed activE real-timE Database System	2.6.2
– DM : Deadline Monotonic	1.2
– ECA : Év´enement-Condition-Action	2.5.2
– EDF : Earliest Deadline First	1.2
– EDF-CR : EDF-Conditional Restart	3.2.2.5
– EDF-HP : EDF-High Priority	3.2.2.5
– ER : mode de verrouillage Essential-Read	5
– EW : mode de verrouillage Essential-Write	5
– FIFO : First In First Out (1^{er} entr´e, 1^{r} sorti)	1.2
– FM : Freshness Manager (Gestionnaire de Fraˆıcheur)	7
– FS : Facteur de Sant´e	4.2
– GD : Gestionnaire de Donn´ees	4 ;7
– GT : Gestionnaire de Transactions	7-4.1
– ICS : Intervalle Critique de Surcharge	2.5.1
– IFR : Instrument Flights Rules	2.4.2
– LRU : Least Recently Used, le (la) moins r´ecemment utilis´e(e)	2.5.3
– LSF : Least Slack First	1.2
– MC : M´emoire Centrale	2.5.3
– MCC : Multiversion Concurrency Control	3.2.4
– MS : M´emoire Secondaire	2.5.3
– N-ACID : Nested-Atomicit´e, Coh´erence, Isolation, Durabilit´e	5
– NER : mode de verrouillage Non-Essential-Read	5
– NEW : mode de verrouillage Non-Essential-Write	5
– NT : Nested Transactions (Transactions Emboˆıt´ees)	5
– OCC : Optimistic Concurrency Control	1.1.2
– OCC-BC : Optimistic Concurrency Control-Broadcast Commit	3.2.2.1
– ORD : Ordonnanceur	4.1
– PA : Presumed Abort	4
– PC : Presumed Commit	4
– PCP : Priority Ceiling Protocol (Protocol de Priorit´e Plafond)	3.2.1.3
– PHP : Protocol d'H´eritage de Priorit´e	3.2.1
– PROMPT : Permits Reading Of Modifi ed Prepared-data for Timeliness	4
– PRS : Plan de R´esolution de Surcharge	2.5.1
– RM : Rate Monotonic	1.2
– RODAIN : Real-time Object-oriented Database Architecture for Intelligent Networks	2.5.4
– RTSORAC : Real-Time Semantic Objects Relationships and Constraints	2.6.4
– RTSQL : Real-Time Structured Query Language	2.6.4
– S et X (verrous) : Shared (partag´e) et eXclusive (exclusif)	3
– SCC : Speculative Concurrency Control	3.2.3

– SCC-2S : Speculative Concurrency Control-2 Shadows 3.2.3
– SGBD : Syst`emes de Gestion de Bases de Donn´ees 1
– SGBDTR : Syst`emes de Gestion de Bases de Donn´ees Temps R´eel 2
– STR : Syst`eme Temps R´eel 1.2
– TE : Transaction Enfant 5
– TG : Transaction Globale 5
– TLT : Top Level Transaction (Transaction de plus Haut Niveau) 5
– TP : Transaction Parent 5
– WCET : Worst Case Execution Time 8
– WEP : Weighted Early Prepare 4.3.1

Contributions à la gestion des transactions dans les SGBD temps réel

Résumé : Les SGBD temps r´eel sont des syst`emes conç us pour g´erer la majorit´e des applications actuelles qui manipulent de grands volumes de donn´ees et qui n´ecessitent un traitement temps r´eel de ces donn´ees. Ces syst`emes h´eritent des principales contraintes et objectifs des SGBD et de ceux des syst`emes temps r´eel traditionnels. Ainsi, ils doivent non seulement respecter les contraintes logiques de la base, i.e. les contraintes d'int´egrit´e, mais ´egalement respecter les contraintes temporelles individuelles des transactions, i.e. respecter les ´ech´eances principalement. Un des probl`emes cruciaux dans les SGBD est de contrˆoler les acc`es concurrents des transactions aux mˆemes donn´ees. Dans les SGBD temps r´eel, le probl`eme est plus complexe, car en plus du contrˆole des acc`es aux donn´ees, il faut faire en sorte qu'un nombre maximum de transactions respectent leurs ´ech´eances. D'autres probl`emes s'ajoutent quand les bases de donn´ees sont distribu´ees. Dans ce document, nous d´ecrivons bri`evement les caract´eristiques des SGBD traditionnels, notamment la gestion des transactions. Ensuite, nous passons en revue les principales caract´eristiques des syst`emes temps r´eel, en particulier les protocoles de base pour l'ordonnancement des tˆaches p´eriodiques et ap´eriodiques. Nous d´ecrivons ensuite quelques axes de recherche sur les SGBD temps r´eel, comme les probl`emes de surcharge ou de sauvegarde/restauration, en insistant plus particuli`erement sur le contrˆole de concurrence des transactions. Sur ce dernier axe, nous pr´esentons nos contributions, i.e. la proposition de protocoles pour am´eliorer les performances des SGBD temps r´eel sous certaines hypoth`eses. Nous d´ecrivons ensuite nos travaux sur la validation des transactions temps r´eel, le mod`ele de transactions imbriqu´ees temps r´eel et l'utilisation de techniques *Anytime*. Avant de conclure, nous pr´esentons les r´esultats pr´eliminaires que nous avons obtenus sur l'ordonnancement par r´etroaction et l'´etude probabiliste des transactions temps r´eel. La conclusion consiste `a esquisser notre projet de recherche.

Mots-clés : Bases de donn´ees, transactions, contrˆole de concurrence, temps r´eel, relaxation des propri´et´es $ACID$

Contribution to Transactions Management in Real-Time DBMSs

Abstract : Real-Time Database Systems (RTDBSs) are designed to manage the majority of current applications which manipulate a large volume of data and have a great need of real-time computation. These systems inherit the main objectives and constraints of both the traditional DBMSs and RTSs (real-time systems). Hence, RTDBSs must not only respect the database logical constraints (integrity constraints), but they must also maintain its temporal consistency, *i.e.* each transaction has to meet its individual deadline. One of the main issues in the DBMSs is to control the access to the same data items by the transactions in incompatible mode. In RTDBSs, this problem becomes more complicated since the transaction manager must not only avoid data access conflicts, but it has also to provide mechanisms that help transactions to meet their deadlines, *i.e.* to maximize the transactions success ratio. Additional issues pertaining notably to the commit process arise in a distributed context. In this dissertation, we describe briefly the main characteristics of the traditional DBMSs, particularly the issues associated to transaction management. We also give an overview of the real-time systems characteristics, especially the basic periodic and aperiodic task scheduling protocols. We then present the main research issues in the RTDBS field such as the overload management and the logging and recovery processes. We focus mainly on the transaction concurrency control and scheduling protocols. We introduce, among others, new protocols for the the purpose of managing efficiently the data access conflicts, under certain assumptions. We proceed by describing our contribution in the distributed real-time transaction commit process, the real-time nested transactions management and the use of Anytime techniques. Afterwards, we give some preliminary results on feedback real-time scheduling and probabilistic appraoch of the real-time transactions behavior. We close this document by presenting some research prospects.

Key-words : Databases, transactions, concurrency control, real-time, relaxation of $ACID$ properties