Les SGBD temps réel

Claude Duvallet [*] — Zoubir Mammeri [**] — Bruno Sadeg [*]

[*]LIH, Université du Havre, 25, rue Philippe Lebon, 76058 Le Havre Cedex
{duvallet, sadeg}@fst.univ-lehavre.fr
[**]IRIT, Université Paul Sabatier, 118, route de Narbonne, 31062 Toulouse
mammeri@irit.fr

RÉSUMÉ. *Certaines applications temps réel manipulent des quantités importantes de données et les traitements effectués sur ces données sont sujets à des contraintes temporelles. Traditionnellement, ces applications sont gérées par des systèmes temps réel, bien adaptés pour la gestion des contraintes temporelles, mais qui sont beaucoup moins efficaces pour manipuler des quantités importantes de données. Pour gérer de manière efficace ces données, les SGBD sont les mieux adaptés. Cependant, les SGBD traditionnels sont conçus avec comme principal objectif, en termes de performances, la minimisation du temps de réponse moyen des transactions. Ils ne sont donc pas adaptés aux applications temps réel où chaque transaction doit respecter ses propres contraintes de temps. Cet article propose une synthèse des travaux effectués sur des systèmes qui combinent les caractéristiques des systèmes temps réel et celles des SGBD traditionnels pour fournir des SGBD temps réel.*

ABSTRACT. *Some real-time applications access and manage great quantities of data and tasks composing these applications must meet timing constraints. Traditionally, these applications are implemented using real-time systems (RTSs) that provide powerful mechanisms to deal with time constraints, but that are less efficient to manage significant quantities of data. To efficiently manage data, the DBMS (DataBase Management Systems) are the most suitable. Nevertheless classical DBMS are designed with one main performance objective, that is to reduce the mean response time of transactions. Thus, they are not suitable for real-time applications where each transaction must meet its time constraints. From the combination of classical DBMSs and RTSs, a new generation of DBMS is born: the real-time DBMSs. This paper presents a review of the most important studies that deal with the design of real-time DBMS.*

MOTS-CLÉS : *Bases de données temps réel, systèmes temps réel, ordonnancement de transactions, contraintes temporelles, évaluation de performances.*

KEY WORDS : *Real-time databases, real-time systems, transactions scheduling, temporal constraints, performance evaluation.*

1. Introduction

Dans les applications temps réel, l'exécution des opérations (tâches) doit respecter des contraintes temporelles explicitement spécifiées pour tenir compte de la dynamique de l'environnement physique des applications. Ces applications manipulent souvent des quantités importantes de données (qui sont, en général, des paramètres représentant les états et les caractéristiques de l'environnement); l'utilisation d'un Système de Gestion de Bases de Données (SGBD) peut s'avérer utile pour gérer ces données de manière efficace [MAM 85], en particulier pour faciliter le partage de données entre les diverses fonctions intervenant dans une installation industrielle (fonctions d'automatisme, de gestion de production, de suivi et contrôle de produits, de maintenance, etc.), en mettant à la disposition de toutes ces fonctions un ensemble de données cohérent. Souvent, on parle de base de données industrielle pour désigner des données manipulées par des fonctions dans une application temps réel, sans que ces données soient réellement structurées sous la forme d'une vraie base de données. En général, il s'agit plutôt de données en mémoire commune. D'où une idée qui fait son chemin et qui consiste à structurer les données industrielles sous la forme d'une base de données. Malheureusement, les SGBD traditionnels ne permettent pas de répondre de manière efficace aux besoins des applications temps réel [RAM 93a], car ils n'intègrent pas de mécanismes de prise en compte de contraintes temporelles.

Depuis une dizaine d'années, une nouvelle génération de SGBD a commencé à voir le jour : c'est celle des SGBD temps réel, qui combinent les mécanismes développés dans le cadre des SGBD traditionnels (relationnels et objets notamment) et des systèmes temps réel. Les premières études sur les SGBD temps réel sont rendues publiques en 1988 avec l'article de Abbot et al. [ABB 88] et celui de Son [SON 88]. Cependant, vu la complexité du domaine, la plupart des problèmes rencontrés dans la conception des SGBD temps réel sont encore au stade d'études préliminaires et les solutions préconisées sont testées par simulation.

Contrairement à la gestion des données dans une base de données traditionnelle, les données dans une base de données temps réel doivent être gérées de telle façon qu'elles soient non seulement cohérentes de manière logique (c'est-à-dire qu'elles doivent satisfaire aux règles d'intégrité), mais elles doivent, en plus, satisfaire la contrainte de cohérence temporelle [SON 92b] (c'est-à-dire que les transactions agissant sur la base doivent être validées avant l'expiration des échéances qui leur sont fixées). Le respect des échéances des transactions nécessite l'utilisation de techniques d'ordonnancement temps réel de ces transactions. Par ailleurs, la nature même des données temps réel (qui ne sont valides que durant des intervalles de temps bien limités) fait que certaines propriétés (notamment les propriétés ACID - Atomicité, Cohérence, Isolation, Durabilité) et l'intégrité des données (qui se fonde essentiellement sur l'ordonnancement de transactions sérialisables) doivent être revues pour être applicables dans un contexte temps réel.

Dans cet article, nous proposons une synthèse des principaux travaux réalisés dans le domaine des SGBD temps réel qui sont pour leur quasi-totalité consacrés au problème de contrôle de concurrence des transactions. Dans la section 2, nous

présentons brièvement les applications/systèmes temps réel ainsi que les principales techniques permettant de spécifier les contraintes temporelles. Dans la section 3, nous présentons les caractéristiques des SGBD temps réel, en particulier celles des données et des transactions temps réel. Nous justifions ensuite le besoin de concevoir des SGBD temps réel en montrant les limites des SGBD traditionnels, actifs et temporels pour la manipulation des données et des transactions temps réel. Dans la section 4, nous étudions les problèmes de contrôle de concurrence et d'ordonnancement des transactions temps réel, ainsi que les solutions envisagées. Dans la section 5, nous abordons d'autres problèmes liés à la conception et à l'implantation de SGBD temps réel, en particulier l'ordonnancement des entrées/sorties, la localisation des bases de données (mémoire principale et/ou mémoire secondaire) et les systèmes d'exploitation pour supporter des SGBD temps réel. La conclusion porte sur différents problèmes qui restent ouverts et qui constituent tous des travaux de recherche qui ne manquent pas d'intérêt.

2. Les applications temps réel et leur spécification

2.1. *Caractéristiques des applications temps réel*

Une application temps réel est généralement gérée par un système contrôleur (le système informatique) qui agit sur un système contrôlé (l'environnement physique de l'application) à l'aide d'actionneurs. Le fonctionnement général des applications temps réel peut se résumer ainsi :

- acquisition de données depuis l'environnement à l'aide de capteurs,
- traitement des données et élaboration des résultats au bout d'un délai limité,
- envoi d'ordres de commande de l'environnement à l'aide d'actionneurs.

A ces fonctions de base, et selon le domaine d'application considéré, viennent s'ajouter d'autres fonctions telles que la maintenance de l'installation industrielle et des équipements informatiques, la gestion du personnel, le service après-vente, …

Un exemple d'application temps réel est celui de la commande d'une chaîne d'assemblage composée de convoyeurs, de stations d'assemblage et de robots. Dans cette application, des robots munis de caméras servent à acquérir les caractéristiques des objets (transportés par les convoyeurs) qui défilent devant leur « champs de vision ». Cette acquisition est contrainte par l'intervalle de temps durant lequel l'objet est présent devant la caméra du robot, ce qui revient à dire que l'action d'acquisition des caractéristiques doit se terminer avant la disparition de l'objet : on dit que l'action possède une échéance.

Réservés il y a quelques années aux installations industrielles (telles que les laminoirs, les raffineries et les usines de fabrication de véhicules), les systèmes temps réel font leur apparition dans beaucoup d'autres secteurs tels que le transport, le multimédia, les consoles de jeux, le suivi de malades, etc. En termes de

complexité, les systèmes temps réel couvrent un large spectre allant du simple microcontrôleur (pour le contrôle du système de freinage d'une voiture, par exemple), jusqu'aux systèmes répartis (pour le contrôle du trafic aérien, par exemple). Les enjeux économiques et les intérêts scientifiques liés aux systèmes temps réel sont multiples. C'est la raison pour laquelle on assiste, depuis les années soixante-dix, à une profusion de langages, de méthodes, d'algorithmes et de protocoles de communication pour le temps réel.

De nombreuses définitions ont été proposées pour clarifier la notion de *système* (et *application*) *temps réel*. Cependant, comme les caractéristiques des systèmes temps réel sont très variées, aucune des définitions proposées n'est vraiment complète pour tenir compte de tous les domaines d'applications. En effet, selon l'aspect abordé dans les systèmes temps réel, on choisit une définition qui s'approche le plus de la problématique traitée. Ainsi, on assimile, selon le cas, un système temps réel à un système rapide, à un système en interaction directe avec un procédé physique, à un système réactif, à un système qui ne fournit pas de réponse en différé, à un système avec un comportement prédictible, à un système qui travaille sur des données « fraîches », à un système robuste, etc.

Cependant, toutes les applications temps réel ont en commun la prépondérance du facteur temps. En effet, les applications temps réel doivent réagir en tenant compte de l'écoulement du temps. Cette caractéristique fondamentale qui distingue globalement les applications temps réel des autres types d'applications informatiques (de gestion ou autres) est exprimée par la définition suivante qui est la plus couramment citée dans la littérature sur le temps réel :

« A real-time system is defined as a system whose correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced » [STA 88].

En plus de l'existence de contraintes de temps, et selon les domaines d'applications, les systèmes temps réel doivent satisfaire d'autres contraintes primordiales, notamment la prédictibilité des comportements et la tolérance aux fautes. En effet, dans les applications temps réel dites temps critique (telles que la commande de procédés industriels ou d'engins militaires), le respect des contraintes de temps est une nécessité en toutes circonstances.

Toute application temps réel est en interaction (forte ou faible selon les cas) avec son *environnement*. Lequel environnement peut être un procédé industriel, un moteur d'avion, un malade, un groupe de participants à une téléconférence, etc. La nature de l'environnement a une incidence directe sur la « criticité » des actions entreprises dans une application temps réel. La notion de « *criticité* » (ou « criticality » en anglais) est utilisée comme critère pour pouvoir classer les applications temps réel selon la sévérité, en termes de coût engendré par le non respect des contraintes temporelles. Ainsi, on classe les applications temps réel en deux types : les applications à contraintes temporelles strictes (« hard real-time applications »), où le non-respect des contraintes de temps peut conduire à des défaillances avec des conséquences pouvant être graves et les applications à

contraintes temporelles relatives (« soft real-time applications »), où le dépassement des échéances est considéré comme une faute bénigne.

2.2. *Spécification des contraintes temporelles*

Les contraintes temporelles peuvent prendre des formes diverses (périodes, échéances absolues ou relatives, etc.) et s'appliquer à divers composants d'une application, c'est-à-dire à des actions (par exemple, une transaction doit se terminer avant une échéance fixée), à des données (par exemple, la validité temporelle des données est limitée) ou à des événements (par exemple, tel événement doit apparaître x unités de temps après tel autre événement). Pour plus de détails sur l'expression des contraintes temporelles, le lecteur pourra se référer à [MAM 98, RAM 96b].

Une fois les contraintes de temps élaborées, ces contraintes doivent être clairement spécifiées avant d'entamer la phase de conception d'applications. La qualité de service de l'application finale repose en grande partie sur la rigueur de la spécification. En effet, la phase de spécification est importante pour toute application informatique et elle est plus cruciale pour les applications temps réel étant données les conséquences que peut avoir une spécification incorrecte ou incomplète. Les applications temps réel sont des plus difficiles à spécifier, en particulier lorsqu'elles sont réparties et qu'elles exigent un haut degré de sûreté de fonctionnement. C'est à partir des années quatre-vingts que la communauté de recherche en informatique a pleinement pris conscience de l'importance de l'élaboration de méthodes et langages (en particulier, ceux fondés sur un formalisme rigoureux) pour la spécification des contraintes temporelles.

Dans les techniques de spécification proposées, on distingue les techniques opérationnelles et les techniques descriptives. Les techniques opérationnelles sont celles définies en termes d'états et de transitions ; elles sont proches de l'exécution. Les techniques descriptives sont souvent fondées sur un formalisme mathématique et produisent des spécifications précises, rigoureuses et donnent une vue abstraite du système. Le système est décrit par des propriétés, forçant le spécifieur à exprimer ce que le système doit faire plutôt que de dire comment le système va le faire. Les spécifications formelles peuvent être traitées automatiquement pour en vérifier la complétude et la cohérence. Il faut noter que ces techniques sont peu utilisées dans le domaine du temps réel, comparées aux techniques opérationnelles, à cause, à la fois, du manque d'outils d'utilisation et des difficultés de spécifier formellement des systèmes complexes tels que les systèmes temps réel. Plusieurs classifications des techniques descriptives ont été proposées, dont voici une :

- Techniques descriptives fondées sur les méthodes algébriques : elles utilisent les types abstraits. En particulier, des extensions de LOTOS, comme RT-LOTOS [COU 93] et ET-LOTOS [LED 93], commencent à émerger pour la spécification de systèmes temps réel.

– Techniques descriptives fondées sur les logiques mathématiques : elles permettent de décrire le comportement d'un système à l'aide de règles qui spécifient comment le système peut évoluer. Pour prendre en compte les aspects temporels, des extensions des logiques temporelles classiques sont utilisées : CTL (Computational Tree Logic) [EME 86], RTL (Real-Time Logic) [JOH 86], RTIL (Real-Time Interval Logic) [RAZ 89], MTL (Metric temporal logic) [KOY 89], RTTL (Real Time Temporal Logic) [OST 89], TCTL (Timed CTL) [ALU 90], TPTL (Timed Propositional Temporal Logic) [ALU 90], TRIO [GHE 90] et TRIO+ [MOR 94].

Les techniques opérationnelles se divisent en deux catégories : les techniques fondées sur les modèles à transitions (machines d'états et réseaux de Pétri) et les techniques fondées sur des notations abstraites. Les machines d'états finis (MEF) permettent de vérifier des propriétés telles que l'atteignabilité des états. Pour être adaptées aux systèmes temps réel, les MEF doivent fournir des possibilités d'expression des contraintes de temps (échéances, périodes, …). Beaucoup d'extensions de MEF ont ainsi été proposées. Des langages fondés sur les MEF, comme PAISLey (Process-oriented Applicative and Interpretable Specification Language), SDL (Specification and Description Language) [UIT 93], ESTEREL [BER 84] et Statecharts [HAR 87], sont utilisés pour la spécification de systèmes temps réel. En ce qui concerne les techniques opérationnelles fondées sur les réseaux de Pétri (RdP), plusieurs extensions des RdP de base ont été proposées pour prendre en compte des contraintes temporelles. Il s'agit notamment des RdP temporisés [BER 91] et des RdP stochastiques [MOL 85].

Les techniques basées sur des notations abstraites sont adaptées à l'analyse et à la conception de systèmes, par décomposition d'un système en sous-systèmes. Elles sont souvent destinées à fournir une représentation visuelle du système pour réduire en quelque sorte l'effort du spécifieur. En général, elles ne modélisent pas l'aspect comportemental des systèmes et par conséquent elles ne sont pas directement utilisables pour une simulation ou pour une exécution du système. Par ailleurs, elles sont souvent informelles ou semi-formelles. Ces techniques incluent des extensions des méthodes SADT (telles que DARTS [GOM 86] – Design Approach for Real-Time Systems – et SDRTS [WAR 86] – Structured Design for Real-time Systems) ou HOOD (telles que HRT-HOOD [BUR 94] – Hard Real-Time HOOD), largement utilisées dans la spécification d'applications non temps réel.

Il existe également des techniques mixtes telles que ESM/RTTL qui est une approche intégrant les machines d'états (Extended State Machines) et la logique temporelle RTTL [OST 87].

Enfin, il faut souligner que dans la pratique, une technique de spécification n'est utile pour un spécifieur que lorsqu'elle est accompagnée d'outils permettant de spécifier, vérifier et simuler des comportements, de générer des implantations, etc. Actuellement, les outils qui existent sont surtout sous forme de prototypes de recherche. Au niveau commercial, peu de produits existent pour appréhender les contraintes temporelles. Le lecteur pourra se référer à [BUC 95] pour une présentation de différents outils et techniques de spécification d'applications temps

réel. Une fois spécifiées, les contraintes temporelles d'une application doivent être prises en compte par des mécanismes adéquats. Nous consacrons le reste de cet article aux travaux d'extension de SGBD afin de répondre aux besoins de garantie de contraintes temporelles, en supposant qu'elles ont été spécifiées par ailleurs.

3. Caractéristiques des SGBD temps réel

3.1. *Données temps réel*

3.1.1. *Taxonomie des données*

Les applications temps réel ont souvent besoin de stocker et de manipuler des volumes importants d'informations qui proviennent des environnements physiques commandés (laminoirs, chaînes d'assemblage, etc.). Ces applications peuvent aussi manipuler des données multimédia (sons, images, graphiques et textes) pour la surveillance et la conduite par les opérateurs. La plupart des données manipulées par ces applications sont dites temps réel, c'est-à-dire qu'elles sont utiles au système seulement si elles sont utilisables et utilisées dans les temps. Chaque donnée temps réel possède une durée de validité. Les conséquences résultant de l'accès d'une transaction à une donnée en dehors de sa durée de validité dépendent des besoins particuliers de l'application et de la sémantique des données. Les applications temps réel manipulent également :

- des données actives : ce sont des données dont la mise à jour provoque une réaction du système qui se matérialise par le déclenchement d'actions (mises à jour d'autres données, déclenchement d'une alarme, ...),
- des données temporelles : ce sont des données datées ; ce qui permet de gérer l'historique de leur évolution [SCH 98].

Il y a donc différents types de SGBD :

- SGBD classiques : ils permettent la gestion des données persistantes (au sens traditionnel).
- SGBDT (SGBD temporels) : ils permettent la gestion des historiques des données (par exemple, les différentes dates d'évolution des salaires des employés dans une entreprise) [ADI 95, TAN 93].
- SGBDA (SGBD actifs) : ils permettent de gérer des données actives. Par exemple, dans les applications de gestion de stock en flux tendu, quand la quantité en stock est au-dessous d'un seuil minimum, une procédure de réapprovisionnement est mise en œuvre automatiquement [COU 98].
- SGBDTR (SGBD temps réel) : ils doivent pouvoir gérer des données temps réel, en prenant en compte les contraintes temporelles imposées aux transactions qui accèdent à ces données.

Le tableau 1 illustre les différents types de données manipulées par les applications temps réel et les systèmes qui les gèrent.

	SGBD	SGBDT	SGBDA	SGBDTR
Données persistantes	*oui*	*oui*	*oui*	*oui*
Données temporelles		*oui*		*oui*
Données actives			*oui*	*oui*
Données temps réel				*oui*

Tableau 1. *Les différents types de données et les SGBD qui les gèrent*

3.1.2. *Cohérence temporelle*

Un des nombreux problèmes soulevés par la conception de SGBD temps réel est le maintien de la cohérence des données dans la base [SON 95a, XIO 96a]. En effet, l'état de l'environnement tel qu'il est perçu par le système contrôleur doit refléter le plus fidèlement possible l'état réel de cet environnement. Cette exigence a une conséquence sur la conception de SGBD temps réel, qui doit non seulement respecter les contraintes d'intégrité (cohérence logique), mais aussi respecter les contraintes de cohérence temporelle des données. La cohérence temporelle regroupe en réalité deux notions [SON 92b] :

- la cohérence absolue qui est liée directement au fait que l'image de l'environnement dans le système doit refléter l'état réel de cet environnement,
- la cohérence relative qui est liée aux données utilisées pour dériver d'autres données dans la base qui doivent être cohérentes entre elles.

Pour illustrer ces deux notions, reprenons la définition d'une donnée temps réel d proposée dans [RAM 93b] : $d = (d_{valeur}, d_{estampille}, d_{dva})$, où : d_{valeur} est la valeur actuelle de la donnée d, $d_{estampille}$ représente l'instant où cette valeur a été mesurée ou calculée et d_{dva} est la durée de validité absolue de la valeur de d.

Un ensemble de cohérence relative R est l'ensemble des données utilisées pour dériver une nouvelle donnée. À tout ensemble R est associée une durée de validité relative R_{dvr}. Soit $d \in R$. On dit que d est dans un état correct, si et seulement si :

1. d_{valeur} est *logiquement cohérente* (elle satisfait aux contraintes d'intégrité),
2. d est *temporellement cohérente* :
   - cohérence absolue : $(instant\_courant - d_{estampille}) \leq d_{dva}$
   - cohérence relative : pour tout $d' \in R$, $| d_{estampille} - d'_{estampille} | \leq R_{dvr}$

Exemple : on considère un système avec deux données, une température et une pression, telles que :

$Température_{dva} = 5$, $Pression_{dva} = 10$,

Ensemble $R = \{température, pression\}$,

Durée de validité relative R_{dvr} = 2.

Si *instant_courant* est égal à 100, on a alors :

- les triplets *température = (347, 96, 5)* et *pression = (50, 97, 10)* sont temporellement cohérents.
- Les triplets *température = (347, 96, 5)* et *pression = (50, 92, 10)* ne sont pas temporellement cohérents (la cohérence relative n'est pas respectée).

Les nombreux travaux effectués sur les SGBD traditionnels peuvent être exploités pour concevoir les SGBD temps réel, en les adaptant afin de prendre en compte la double contrainte qui consiste à exiger que les transactions s'exécutent avant leurs échéances et que ces mêmes transactions accèdent à des données ayant des durées de validité limitées. Ceci entraîne des conséquences sur la manière de concevoir ces systèmes. En particulier, les propriétés ACID (Atomicité, Cohérence, Isolation et Durabilité) des transactions des bases de données classiques nécessitent d'être revues [RAM 93a]. Il faut, ou bien les adapter au contexte temps réel ou bien identifier des propriétés équivalentes pour les SGBD temps réel. En effet :

- la notion de sérialisabilité dans les SGBD temps réel est la même que celle dans les SGBD classiques à laquelle on rajoute des contraintes temporelles,
- le respect des échéances des transactions est souvent un critère plus important que celui de la précision des résultats, c'est-à-dire qu'une réponse partielle qui arrive avant échéance peut être plus utile pour l'application qu'une réponse précise (ou complète) fournie en retard. Ceci implique que la propriété d'atomicité des transactions peut ne pas être respectée (puisque certaines des actions qui composent la transaction sont annulées si leur échéance est dépassée).
- la notion de durabilité n'a pas le même sens dans les SGBD temps réel que dans les SGBD classiques. Dans les SGBD classiques, cette notion signifie que les modifications effectuées sur une base de données deviennent permanentes (persistantes) dès que les transactions de mise à jour sont validées[1]. Dans les SGBD temps réel, et notamment pour ce qui concerne les données sensorielles (issues des capteurs), les modifications s'effectuent de façon périodique. Donc la persistance de telles données n'est vérifiée que durant leurs durées de validité.

On voit donc que le critère de correction des transactions temps réel n'est plus le même que dans les SGBD classiques. Comme l'affirment Kim et Son [KIM 94], puisque les systèmes temps réel sont utilisés souvent pour répondre à des stimuli extérieurs ou pour contrôler des dispositifs physiques, un résultat, même approximatif, obtenu dans les temps par des transactions non sérialisables peut s'avérer plus utile qu'un résultat obtenu en retard par des transactions sérialisables. Selon les applications, tant que les résultats des transactions restent cohérents par rapport à l'environnement, le fait que la base soit cohérente ou ne le soit pas encore

[1] Terminées avec succès (les modifications sur la base deviennent visibles pour les autres transactions).

peut ne pas être important pour l'application. Donc ces applications privilégient le critère de cohérence temporelle absolue. Alors que d'autres applications peuvent privilégier le critère de sérialisabilité des transactions et d'autres encore privilégient les deux critères, la sérialisabilité et la cohérence des transactions [GRA 92].

3.2. *Traitements temps réel*

3.2.1. *Taxonomie des transactions*

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 [HAR 92, PUR 94, RAM 93a] :

- Transactions à échéances strictes et critiques (« hard deadline transactions ») : une transaction qui rate son échéance peut avoir des conséquences graves sur le système ou sur l'environnement contrôlé. Par exemple [MAM 98], 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 rate son échéance, elle devient inutile pour le système. Ses effets sont nuls. Elle est donc abandonnée[2] dès qu'elle rate son échéance. En reprenant l'exemple du robot qui capte des informations sur des objets défilant sur un convoyeur (§2.1), si l'opération d'acquisition n'est pas achevée 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 rate 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 un peu en retard, elle peut toujours être présentée à l'utilisateur si le décalage par rapport au son n'est pas trop important. 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 il rend la scène incompréhensible.

[2] Terminée avec échec.

L'objectif global d'un SGBD temps réel est double : (1) la garantie absolue de l'exécution et de la validation (avant leurs échéances) des transactions à échéances strictes et critiques et (2) la minimisation du nombre des transactions à échéances strictes et non critiques ou à échéances non strictes qui ratent leurs échéances.

Les principaux travaux actuels sur les SGBD temps réel portent sur les systèmes où les transactions possèdent des échéances strictes et non critiques ou des échéances non strictes. Ces travaux concernent particulièrement des techniques de contrôle de concurrence des transactions et leur ordonnancement ; l'objectif du système est de minimiser le nombre de transactions qui ratent leurs échéances.

Pour les transactions à échéances strictes et critiques, il n'existe pas à notre connaissance d'algorithmes d'ordonnancement des transactions qui garantissent leur terminaison avant l'échéance. La principale raison provient de l'indéterminisme de ces transactions et du manque de méthodes permettant d'estimer avec précision le temps d'exécution des transactions. Il existe cependant quelques tentatives de modélisation des ordonnanceurs de transactions à échéances strictes et critiques [BES 94, BRA 95a, SON 93]. Ils sont généralement basés sur un surdimensionnement des ressources du système.

Les transactions temps réel peuvent manipuler des objets de base ou des objets dérivés. Les objets de base représentent des entités concrètes de l'environnement (température, pression, etc.). Les valeurs d'objets dérivés sont calculées à partir de celles d'autres objets. Par exemple, les données acquises sur la position et la vitesse d'objets en mouvement sont utilisées pour dériver la nouvelle commande à appliquer au robot qui permet de déplacer ces objets. Cette distinction entre types d'objets conduit à un deuxième classement des transactions [XIO 96a] :

- Transactions sensorielles : ce sont des transactions qui mettent à jour les données sensorielles (objets de base). Elles sont à écriture seule.
- Transactions déclenchées de mise à jour : ce sont des transactions déclenchées par les mises à jour des objets de base. Elles mettent à jour les objets dérivés. Ce sont donc des transactions en lecture/écriture.
- Transactions utilisateur : ce sont celles exécutées par l'utilisateur, avec des échéances. Elles sont également en lecture/écriture.

3.2.2. *Échéances des transactions*

Dans les SGBD temps réel, les transactions possèdent des contraintes de temps, souvent sous forme d'échéances qui peuvent provenir de deux sources [RAM 96b] :

- des contraintes temporelles liées à l'environnement (par exemple, l'action qui corrige la trajectoire d'un robot téléguidé doit s'exécuter avant que le robot ne rentre en collision avec un obstacle),
- des contraintes temporelles liées à des choix de conception et d'implantation d'une application (par exemple, durée d'exécution d'une opération sur un processeur choisi ou délai de communication de messages).

Pour illustrer cette notion d'échéance, prenons l'exemple présenté dans [XIO 96b]. Soit *T* une transaction de lecture et deux données *Y* et *Z* (figure 1). La transaction *T* a besoin de lire les deux données temps réel *Y* et *Z* pour produire un résultat. Les lectures (à partir de l'environnement) se font aux instants t_b et t_c respectivement. On dispose des éléments suivants :

- Echéance de *T* : l'instant t_f (initialement égale à 13),
- Durée de validité absolue de *Y* : 7 unités de temps,
- Durée de validité absolue de *Z* : 3 unités de temps.

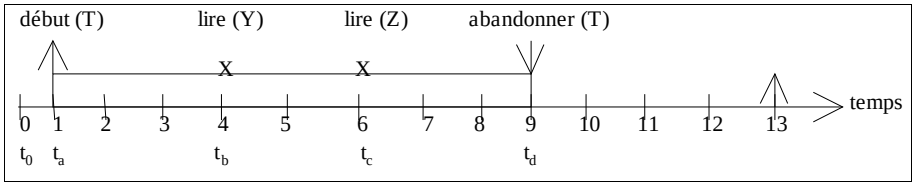

Figure 1. *Exécution d'une transaction temps réel*

Supposons que la transaction *T* commence à l'instant $t_a = 1$. A l'instant $t_b = 4$, la transaction lit *Y*. Son échéance devient $t_f = t_b + 7 = 11$ (car au-delà de $t = 11$ la donnée *Y* n'est plus valide). À l'instant $t_c = 6$, elle lit la donnée *Z*. Son échéance devient $t_f = t_c + 3 = 9$ (car au-delà de $t = 9$, la donnée *Z* devient invalide). Si *T* ne se termine pas avant $t = 9$, elle est abandonnée. Noter qu'elle pourrait être redémarrée, en utilisant les versions suivantes de *Y* et *Z* (acquises par exemple aux instants 11 et 12), et elle pourrait se terminer avant son échéance initiale $t_f = 13$. Ainsi, une transaction peut utiliser une valeur d'une donnée parmi l'ensemble de ses valeurs valides dans l'intervalle de temps séparant l'instant de début de cette transaction et son échéance. On parle dans ce cas de « similarité » des valeurs [KUO 96].

3.3. *Pourquoi un SGBD temps réel ?*

Dans de nombreuses applications temps réel actuelles, l'objectif est la gestion efficace des données en respectant les contraintes temporelles qui leur sont imposées. Les systèmes adaptés pour la gestion d'applications de ce type sont les systèmes temps réel (STR), car ils disposent d'algorithmes efficaces pour l'ordonnancement des tâches temps réel de manière à respecter leurs échéances, de mécanismes de communication via des mémoires communes, etc. Mais, ils ne sont pas satisfaisants pour la gestion efficace des données lorsque celles-ci sont volumineuses et seul un système de type SGBD pourrait répondre à ce besoin. Cependant, les types de SGBD connus jusqu'à maintenant présentent des insuffisances/incapacités lorsqu'il s'agit de prendre en compte des données dans un contexte où des contraintes temporelles sont présentes. Nous allons analyser brièvement les limites des différents systèmes de gestion de données.

3.3.1. *Limites des systèmes temps réel traditionnels*

Comme nous l'avons déjà mentionné, les STR ne sont pas satisfaisants pour la gestion efficace des applications temps réel quand celles-ci manipulent des volumes importants de données. Ceci est dû au fait que la gestion des tâches dans un STR est différente de la gestion des transactions dans un SGBD [GRA 92]. L'une des différences fondamentales entre les systèmes temps réel et les SGBD réside dans la gestion de la cohérence des données. Cette gestion est au centre de tout SGBD, alors qu'elle est absente dans les STR. Dans les SGBD, la garantie de la cohérence des données est fondée sur la sérialisation des transactions et sur les mécanismes de validation. La conception d'une application correcte impose des transactions sérialisables. Dans les STR, les tâches partagent des données communes (gérées évidemment par exclusion mutuelle quand cela est nécessaire). L'ordre dans lequel les tâches accèdent aux données partagées est souvent inconnu a priori et il n'y a pas de validation des opérations d'écriture de données à la fin de l'exécution des tâches. Dans les STR, la cohérence des données incombe aux tâches et donc à la manière dont a été conçue la coopération entre tâches. L'une des conséquences de cette différence est que les algorithmes d'ordonnancement des tâches temps réel ne peuvent pas être appliqués tels quels à l'ordonnancement des transactions temps réel.

3.3.2. *Limites des SGBD traditionnels*

Dans les SGBD traditionnels, il n'est pas tenu compte des contraintes temporelles des transactions. L'objectif en termes de performances est de minimiser *le temps de réponse moyen* des transactions. Il peut donc arriver que le temps de réponse de certaines transactions soit très important, ce qui n'est pas acceptable dans une application temps réel où chaque transaction doit s'exécuter durant un intervalle de temps précis.

3.3.3. *Limites des SGBD actifs*

Une donnée est dite active si elle conduit à l'activation d'une action du système quand elle est mise à jour, à la suite d'un événement. Ce sont les SGBD actifs qui gèrent ce type de données [COL 94, WID 95, RAM 96a]. Le paradigme associé à ce type de données est appelé règle E-C-A (Evénement-Condition-Action). Il s'exprime selon le schéma général suivant :

Quand *un événement se produit*
Si *une condition est satisfaite*
Alors *déclencher une action.*

Dans les systèmes temps réel, les événements peuvent être de différentes natures : événements générés à la suite de changement d'état de l'environnement, événements temporels (une période écoulée, etc.), début ou fin d'une action, etc.

Les conditions portent généralement sur les valeurs des données (par exemple : température > 1000°). L'action peut être n'importe quelle opération à déclencher (par exemple, mise à jour d'une autre donnée).

Considérons un exemple avec des contraintes temporelles et exprimons-le à l'aide de la règle E-C-A. Soit un système de contrôle d'atterrissage d'avion dont l'objectif est de s'assurer qu'une fois que la décision d'atterrir est prise, les étapes nécessaires à l'atterrissage soient terminées avant un certain délai (une échéance de 10 minutes, par exemple). Les étapes pourraient être : entamer la décélération, réduire l'altitude, sortir le train d'atterrissage, etc. L'exécution de ces étapes dépend de facteurs tels que la disponibilité des pistes, des contraintes spécifiques à l'aéroport, la visibilité, le type d'avion, … Ces facteurs pourraient être stockés dans une base de données que le système pourrait consulter. Si le système constate, au vu des informations extraites de la base, que les étapes de l'atterrissage ne peuvent pas se terminer dans les temps (avant 10 minutes), on devrait pouvoir annuler l'atterrissage, en se donnant une échéance de 5 minutes (probablement parce que dans ce cas, il existe d'autres alternatives !). Cette situation peut se traduire dans le système par le paradigme E-C-A suivant :

Quand *10 minutes se sont écoulées depuis l'amorce de l'atterrissage,*
Si *toutes les étapes ne sont pas terminées,*
Alors *annuler l'atterrissage dans les 5 minutes.*

L'exemple précédent, comme la plupart des exemples d'applications temps réel, se décrivent facilement à l'aide du paradigme E-C-A, mais avec des contraintes temporelles associées aux événements et actions. Malheureusement les SGBD actifs ne permettent pas de prendre en compte ce type de contraintes. L'aspect « actif » qui exprime la réaction aux événements est nécessaire, mais pas suffisant, pour que les SGBD actifs répondent aux besoins des applications temps réel.

3.3.4. *Limites des SGBD temporels*

Les SGBD temporels [ADI 95, SCH 98, SNO 95, TAN 93] fournissent des mécanismes permettant la gestion de l'historique des données (c'est-à-dire des différentes versions de données estampillées). L'aspect *temps* y est donc présent dans une certaine mesure. Malheureusement, ce type de SGBD ne permet pas de prendre en compte des contraintes temporelles imposées aux transactions dans un contexte temps réel.

Il est important de souligner que les deux types de SGBD (SGBD temps réel et SGBD temporels) bien qu'ayant des objectifs distincts, ne sont pas si éloignés que cela l'un de l'autre [RAM 93b]. En effet, dans les systèmes temps réel tels que ceux développés pour le contrôle des réacteurs nucléaires ou pour la gestion des réseaux, on examine d'abord le comportement antérieur du système contrôlé pour décider

des actions à entreprendre sur ce système. La technologie des SGBD temporels fournit les mécanismes adéquats pour sauvegarder cet historique. Ils manipulent également la notion de validité des données [RAM 93b] dont l'équivalent pour les SGBD temps réel est la cohérence temporelle des données. Ces deux types de systèmes sont donc complémentaires et l'intégration des fonctions de l'un dans l'autre ne peut être que bénéfique [XIO 97b].

Les limites des quatre types de systèmes esquissées précédemment ont suscité et suscitent encore des études pour voir dans quelle mesure on peut exploiter les nombreux travaux sur la conception des SGBD traditionnels, les SGBD actifs, les SGBD temporels et les systèmes temps réel pour concevoir et réaliser des systèmes pour la gestion de bases de données temps réel.

4. Contrôle de concurrence d'accès aux données dans les SGBD temps réel

4.1. *Problématique*

Les données enregistrées dans une base de données peuvent être accédées simultanément par plusieurs transactions. Ces dernières doivent donc pouvoir s'exécuter de manière concurrente. Dans les SGBD traditionnels, l'ordonnancement sérialisable est le critère accepté par tous pour le maintien de la cohérence de la base en cas d'accès concurrents. En d'autres termes, si des transactions concurrentes sont sérialisables, alors la base de données est maintenue dans un état cohérent après l'exécution de ces transactions. Deux transactions sont sérialisables si le résultat de leur exécution concurrente est le même que celui de leur exécution séquentielle.

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 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 (§3.3.1), les algorithmes conçus pour l'ordonnancement des tâches temps réel [STA 95] ne peuvent pas être appliqués directement aux transactions des bases de données pour intégrer les contraintes temporelles. Les travaux sur le contrôle de concurrence des transactions dans les SGBD temps réel s'appuient sur les techniques traditionnelles de contrôle de concurrence (CC) des transactions dans un SGBD d'une part et sur les techniques d'ordonnancement des tâches dans les STR d'autre part. Les techniques de contrôle de concurrence des transactions dans les SGBD temps réel sont basées principalement sur deux politiques :

- Politique de CC optimiste : où une transaction en conflit avec d'autres transactions est abandonnée et redémarrée. Les algorithmes de cette classe

permettent aux transactions de s'exécuter en concurrence en prenant le risque d'avoir à les redémarrer si des incohérences apparaissent dans la base. Cette politique est dite optimiste car l'hypothèse considérée est qu'il existe une faible probabilité pour que deux transactions rentrent en concurrence sur un même granule[3] de données. Les transactions s'exécutent donc jusqu'au moment de la validation. Dans ce cas, si deux transactions sont en concurrence sur un objet, l'une d'elles sera abandonnée et redémarrée.

- Politique de CC pessimiste : où une transaction en conflit est bloquée (mise en attente) jusqu'à la validation des transactions avec lesquelles elle est en conflit. Les algorithmes de cette classe évitent toute exécution concurrente de transactions tant qu'il existe des conflits potentiels. Cette politique est dite pessimiste car l'hypothèse sous-jacente est que toute paire de transactions qui s'exécute en concurrence est susceptible de rentrer en conflit. L'une des transactions se met alors en attente jusqu'à la validation de l'autre.

Dans la suite de cette section, nous allons présenter les principales techniques d'ordonnancement de transactions temps réel. Selon les auteurs, on parle d'*algorithme* d'ordonnancement ou de *protocole* d'ordonnancement, pour décrire une technique d'ordonnancement de transactions ou de tâches temps réel. Afin d'éviter certaines ambiguïtés de traduction, nous conserverons les termes utilisés par les auteurs des différentes techniques d'ordonnancement que nous présenterons.

4.2. *Techniques optimistes*

Dans les systèmes où ces techniques sont implantées, toutes les transactions sont servies dès qu'elles en font la demande. Mais 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[4] afin de détecter si elle est en conflit avec d'autres transactions qui auraient effectué la validation depuis le début de son exécution. Une transaction passe donc par trois phases : lecture, certification et validation (écriture). Le test de certification peut s'effectuer de deux manières :

- Certification « en arrière ». Il s'agit d'une certification par rapport aux transactions validées. Les objets lus par la transaction T_c en phase de validation sont examinés. Si l'un d'eux est écrit par une transaction qui a effectué sa validation après le début de l'exécution de T_c alors la transaction T_c est abandonnée, car elle pourrait ne pas avoir pris connaissance de la nouvelle valeur de l'objet (les écritures sont différées dans les protocoles optimistes).
- Certification « en avant ». Il s'agit d'une certification par rapport aux transactions actives. Les objets écrits par la transaction T_c sont comparés à

[3] Donnée élémentaire d'une BD accédée par les transactions.
[4] Test de sérialisibilité réalisé à la fin d'une transaction dans les protocoles optimistes [KUN 81].

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.

4.2.1. *Le protocole OCC-BC (Optimistic Concurrency Control-Broadcast Commit)*

Dans l'algorithme OCC de base [KUN 81], les conflits entre transactions ne sont détectés qu'au moment de la phase de validation. Les ressources détenues par des transactions, qui atteignent cette phase et qui devront être redémarrées sont donc perdues. Afin de minimiser les pertes de ressources et de redémarrer les transactions le plus tôt possible, Menasce et al. [MEN 82] ont proposé l'algorithme OCC-BC, 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, augmentant ainsi leurs chances de se terminer avant leurs échéances (figure 2 et 3).

Dans [HAR 92], 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 (T_c) > priorité (T_i) $\forall i$ $i=1,...,n$ alors les transactions T_i sont abandonnées.
- Cas 2 : S'il existe des transactions $T_{k\ (k \in [1,n])}$ telles que priorité (T_k) > priorité (T_c), alors T_c 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 (1er 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 augmentation 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.

La figure 2 montre l'exemple de deux transactions T_1 et T_2 en conflit. T_2 lit une donnée x, après que T_1 l'ait mise à jour. Avec OCC de base, la transaction T_2 doit être redémarrée dès qu'elle rentre dans sa phase de certification (car elle est en conflit sur la donnée x avec une transaction qui a déjà effectué sa validation). Dans ce cas, la probabilité que T_2 s'exécute dans les temps, après son redémarrage, s'affaiblit. La variante OCC-BC (figure 3) évite d'attendre jusqu'à la certification

de T_2 pour la redémarrer, puisque toutes les transactions en conflit avec T_1 sont averties de sa validation. T_2 est donc redémarrée immédiatement après la validation de T_1.

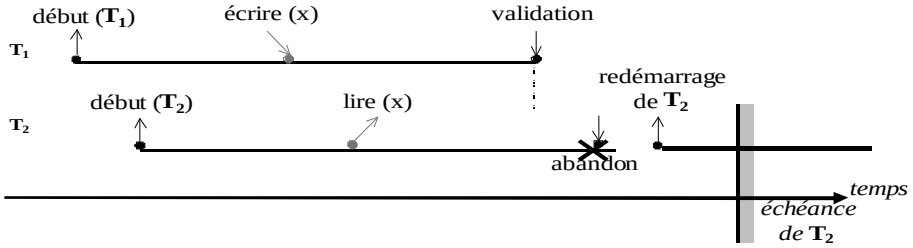

Figure 2. *Ordonnancement avec OCC de base*

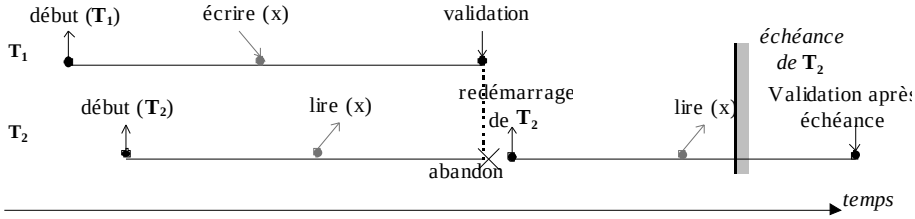

Figure 3. *Ordonnancement avec OCC-BC*

4.2.2. *Le protocole wait-50*

Pour pallier les problèmes rencontrés dans le protocole OCC-BC, Haritsa et al. [HAR 92] ont proposé une autre variante qui se présente comme une stratégie de compromis. En plus de « l'attente par priorité », ils ont rajouté 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 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 [HAR 92, HUA 96], 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 satisfaire 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.

4.2.3. *Le protocole CCA (Cost Conscious Approach).*

Dans [HON 93], Hong et al. présentent l'algorithme CCA pour l'ordonnancement des transactions temps réel. Cet algorithme améliore les performances des algorithmes EDF-HP (Earliest Deadline First-High Priority) et EDF-CR (Conditional Restart) [ABB 88], 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 imminente. Des performances meilleures sont obtenues avec l'algorithme EDF-CR car il utilise une information supplémentaire : une estimation du temps d'exécution des transactions. Il permet ainsi à des transactions non prioritaires de se terminer à condition qu'elles n'obligent pas les autres transactions en conflit ayant une plus haute priorité à rater leur échéance.

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. L'algorithme estime si T_d peut terminer son exécution sans remettre en cause le respect de l'échéance de T_r, auquel cas il l'autorise à poursuivre son exécution évitant ainsi un abandon (peut-être) inutile, sinon il l'abandonne.

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 pré-analyse 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 pré-analyse consiste à effectuer une analyse syntaxique des transactions afin de construire leur arbre de décision. Les figures 4 et 5 illustrent un exemple de pré-analyse de deux transactions concurrente *A* et *B* et la construction de leur arbre de décision. Ces deux arbres permettent de connaître la structure des transactions et pour chaque branche de ces arbres, les données qui seront accédées.

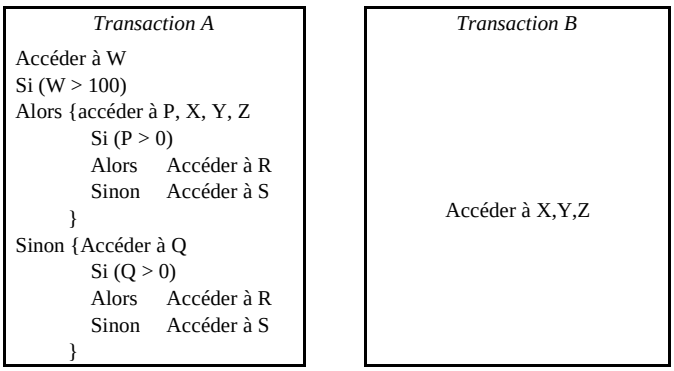

Figure 4. *Exemple de transactions susceptibles d'être concurrentes*

La pré-analyse de transactions concurrentes effectuée précédemment permet de mettre en évidence l'absence de conflit dans certains cas d'exécution (feuilles *Ae* et *Af* de l'arbre de décision de la transaction *A*) alors que potentiellement au début de l'exécution, un conflit sur les données *X*, *Y* et *Z* existait (nœud *A*). L'algorithme CCA peut alors utiliser les informations obtenues par la pré-analyse lors de l'exécution concurrente des transactions *A* et *B* afin de déterminer les conflits qui surviennent réellement en cours d'exécution. Cet aspect d'évaluation dynamique de l'exécution des transactions permet de diminuer les blocages.

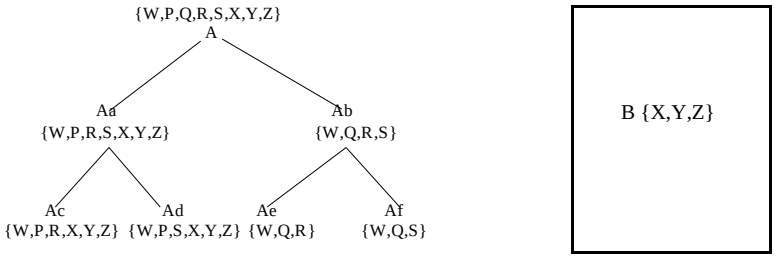

Figure 5. *Arbres de décision des transactions A et B*

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 [CHA 98]. 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 [CHA 98] 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 pré-analyse des transactions peut s'avérer négative ; néanmoins puisqu'il s'agit d'une pré-analyse 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. Cependant, les simulations effectuées considèrent que les verrous sur les données sont exclusifs et que toutes les transactions ont le même niveau de criticité. L'utilisation de verrous partagés et des niveaux multiples de criticité des transactions pourrait affecter les résultats obtenus.

4.3. *Techniques pessimistes*

Dans les SGBD traditionnels, l'algorithme de verrouillage à deux phases, 2PL (Two Phase Locking) [ESW 76] est le plus répandu parmi les techniques pessimistes. Son principe repose sur le fait qu'une transaction qui a besoin d'un objet devra d'abord acquérir un verrou sur cet objet. L'algorithme se déroule en deux étapes : (1) acquisition des verrous et exécution des actions composant la transaction et (2) libération des objets verrouillés.

A chaque fois qu'une transaction T_d demande un verrou sur un objet détenu par une autre transaction T_a dans un mode conflictuel (lecture/écriture ou écriture/écriture), alors T_d est bloquée jusqu'à la libération du verrou par la transaction T_a. L'adaptation de cet algorithme au contexte temps réel a donné naissance à deux principales familles d'algorithmes : *l'abandon par priorité* et *l'héritage de priorité.*

4.3.1. *L'abandon par priorité*

L'algorithme d'abandon par priorité proposé dans [HAR 92] consiste à abandonner la transaction de plus basse priorité quand le phénomène d'*inversion de priorité*[5] 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 de l'objet considéré, alors cette transaction doit attendre que l'objet soit libéré, comme dans l'algorithme 2PL traditionnel. De plus, une nouvelle transaction de lecture ne peut rejoindre l'ensemble des transactions de lecture qui détiennent un verrou que si elle a une priorité supérieure à toute transaction d'écriture en attente du verrou. Ce protocole assure que les 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.

4.3.2. *L'héritage de priorité*

L'algorithme d'héritage de priorité [HAR 92] 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. A chaque fois que le phénomène d'inversion de priorité survient, la transaction de plus faible priorité voit sa priorité augmentée et ramenée au niveau de celle de la transaction demandeuse du verrou, jusqu'à ce qu'elle se termine et libère le verrou. De cette manière les transactions détentrices

[5] Situation où une transaction *T* doit attendre qu'une autre transaction de plus faible priorité libère une ressource dont *T* a besoin pour poursuivre son exécution.

de verrous pourraient se terminer plus tôt, réduisant ainsi les temps d'attente des transactions de haute priorité. Les inconvénients de ce protocole sont notamment :

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

4.3.3. *L'héritage conditionnel de priorités*

Il s'agit d'une variante du protocole précédent [HUA 91]. 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 faible priorité auxquelles il reste beaucoup de temps pour se terminer). Les simulations effectuées par Huang et al. [HUA 96] 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.

4.3.4. *Le protocole de plafonnement de priorités (Priority Ceiling Protocol, PCP)*

Il s'agit d'une extension du protocole de l'héritage de priorité qui élimine les interblocages et limite les durées des inversions de priorités. Il est conçu à l'origine pour l'ordonnancement des tâches dans les STR [SHA 90], 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 91] 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. [NAK 93] qui ont introduit le protocole CCP (Convex Ceiling Protocol) qui 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 les inconvénients suivants :

- ils ne donnent 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.

Récemment, Lam et al. ont proposé une nouvelle variante du protocole PCP qui améliore les deux précédentes, en évitant des blocages inutiles des transactions [LAM 97].

4.4. *Autres techniques*

4.4.1. *Protocole hybride des intervalles d'estampilles*

Ce protocole proposé dans [SON 92a] combine les techniques optimistes et l'ordre des estampilles[6]. Il améliore les algorithmes optimistes où se posent des problèmes de gestion efficace de 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 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 phase de lecture [HAR 90]). 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.

De plus, le protocole hybride des intervalles d'estampilles utilise l'allocation dynamique d'estampilles [BAY 82], 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 [BOK 87]. A chaque transaction est affecté un intervalle d'estampille (correspondant initialement à tout l'espace des estampilles). Ensuite, à chaque lecture ou écriture d'un objet par la transaction, son intervalle d'estampille est ajusté pour préserver l'ordre de sérialisation induit par la validation d'autres transactions.

4.4.2. *Le protocole de Contrôle de Concurrence Multiversions (MCC)*

[6] Une estampille de transaction est une valeur numérique unique affectée à la transaction dès son arrivée.

Dans ce protocole [BER 83], 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 [BER 87].

Dans [KIM 91], Kim et Srivastava ont adapté cet algorithme 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 tâches qui écrivent les valeurs issues des capteurs 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 est un critère important.

Dans [SON 95b], Son et al. présentent un protocole complexe qui combine les caractéristiques des techniques optimistes et celles des estampilles. Un autre protocole basé sur la combinaison des caractéristiques des protocoles optimistes et des techniques de contrôle de concurrence multiversions est présenté par Son et Lin dans [SON 92a]. Les auteurs affirment, sur la base d'une analyse qualitative, que les algorithmes hybrides présentent des performances meilleures que ceux basés 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.

4.4.3. *Les protocoles SCC (Speculative Concurrency Control)*

SCC est une nouvelle classe de protocoles de contrôle de concurrence conçue spécialement pour les SGBD temps réel [BES 94, BRA 95b] fondés sur la *redondance* des ressources. L'objectif est de trouver le plus tôt possible des ordonnancements sérialisables et de les mettre en œuvre, 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, qui est 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.

Dans un algorithme optimiste de type OCC classique, l'exécution d'une transaction comporte trois phases : exécution, certification, validation. Le sort d'une transaction n'est déterminé qu'à la phase de 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 visible la modification). Dans le 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 (cf. §4.2.1.). Il est illustré dans les figures 2 et 3.

Le premier protocole conçu suivant le protocole SCC est SCC-OB (Speculative Concurrency Control-Order Based). Il est basé sur l'ordre de sérialisation des transactions (actions de lecture et d'écriture composant les transactions). 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 s'effectue sans qu'aucune transaction ne rate son échéance.

Pour illustrer le fonctionnement du protocole SCC-OB, considérons les figures 6 et 7 où deux transactions T_1 et T_2 accèdent à une donnée x : elles concernent deux scénarios 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.

Dans le cas de la figure 6, la transaction T_2 arrive à son étape de certification avant celle de T_1. Elle réalise donc sa validation sans conflit avec T_1. Cet ordonnancement est donc sérialisable avec T_2 qui précède T_1. Une fois que T_2 a effectué sa validation, sa copie T'_2 est abandonnée.

Dans le cas de la figure 7, c'est la transaction T_1 qui atteint en premier sa phase de certification. La transaction T_2 ne peut donc continuer à s'exécuter à cause du conflit sur la valeur de x (visible maintenant). Elle doit être abandonnée. Mais au lieu de redémarrer T_2, sa copie T'_2 qui a été créée au moment de l'apparition du conflit avec T_1, est en cours d'exécution et pourra lire x. La transaction T_2 d'origine est abandonnée et la transaction T_2' peut encore se terminer correctement.

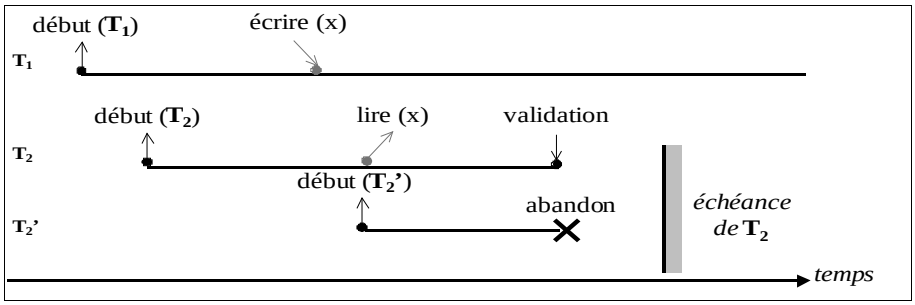

Figure 6. *Ordonnancement avec SCC-OB, en absence de conflit potentiel*

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-kS [BRA 95b].

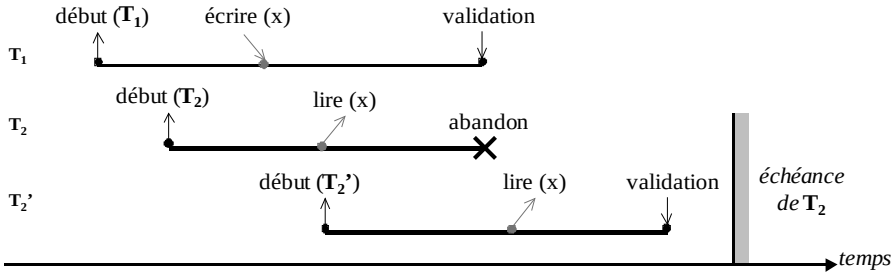

Figure 7. *Ordonnancement avec SCC-OB, en présence de conflit*

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.

4.5. *Récapitulatif des algorithmes d'ordonnancement de transactions temps réel*

A l'aide du tableau 2, nous reprenons l'ensemble des algorithmes précédemment décrits afin d'en faire une analyse comparative.

Techniques	Protocoles	Avantages	Inconvénients

Optimistes	OCC-BC (Optimistic Concurrency Control-Broadcast Commit)	La transaction en phase de validation est privilégiée et elle garde les ressources acquises.	Abandons très nombreux. Verrouillage des ressources liées à la transaction en attente, donc augmentation du risque de conflits et baisse des performances du système.
	Wait-50	Ressources bloquées moins longtemps que pour OCC-BC . Nombre d'abandons réduit.	Les mêmes que OCC-BC mais de façon plus modérée.
	CCA (Cost Conscious Approach)	Nombre d'abandons réduit. Tient compte du coût des transactions abandonnées, et intègre les contraintes à échéances non strictes.	Ne tient pas compte du coût de la pré-analyse des transactions (construction d'arbres de décision).
Pessimistes	Abandon par priorité	Les transactions de plus haute priorité terminent plus facilement dans les temps. Nombre d'abandons réduit.	Certaines transactions sont abandonnées au profit de transactions qui ne terminent pas toujours dans les temps, d'où perte de temps pour le système.
	Héritage de priorité	Evite l'inversion de priorités.	Durée de blocage imprévisible. Les transactions de plus faible priorité concurrencent les autres sur l'accès aux ressources.
	Héritage conditionnel de priorité	Moins de perte de travail. Délais d'attente réduits pour les transactions de haute priorité. Meilleures performances que le protocole précédent.	Nombre d'abandons important.
	PCP (Priority Ceiling Protocol)	Élimine les interblocages. limite la durée de l'inversion de priorité	Nécessité d'une connaissance a priori sur les ressources. Assignation statique des priorités aux transactions. Blocages inutiles de transactions
Autres	Protocole hybride des intervalles d'estampilles	Améliore la détection et la résolution des conflits, diminue les abandons inutiles.	Relativement complexe à mettre en œuvre.
	MCC (Contrôle de Concurrence Multiversions)	Réduit le taux de transactions abandonnées.	L'exploitation d'anciennes versions des données peut compromettre la cohérence temporelle des bases de données temps réel.
	SCC (Speculative Concurrency Control)	Augmente la probabilité que les transactions satisfassent leurs échéances.	Nécessite beaucoup de ressources. Nombre d'abandons important.

Tableau 2. *Comparaison d'algorithmes d'ordonnancement des transactions avec contraintes temporelles strictes non critiques*

5. Autres aspects dans les SGBD temps réel

5.1. *Ordonnancement des entrées/sorties*

Dans un SGBD, en plus du problème de la sérialisation des transactions, se posent les problèmes d'allocation et de libération de tampons qui leurs sont dédiés. Dans [ABB 88], des simulations de plusieurs algorithmes d'ordonnancement d'E/S sur disque et de gestion de tampons ont été effectuées en prenant en compte les contraintes de temps induites par l'aspect temps réel (échéances des transactions, priorités, …). Les auteurs en ont déduit que la prise en compte des contraintes temporelles par les algorithmes d'ordonnancement des E/S améliore sensiblement le respect des échéances des transactions. La stratégie la plus répandue pour les transactions d'écriture dans une BD consiste à différer celles-ci jusqu'à ce que la transaction demande à effectuer sa validation. L'un des avantages est qu'aucun travail n'est à défaire en cas d'échec de la transaction. Dans [ABB 90], Abbot et Garcia-Molina affirment que même dans un contexte temps réel, les écritures peuvent être différées et proposent une nouvelle architecture pour les E/S qui traite de manière asymétrique les lectures et les écritures.

En général, très peu de travaux ont été effectués sur les entrées/sorties qui, rappelons-le, représentent un goulot d'étranglement pour tout système informatique. Par conséquent, il est difficile sinon impossible dans certaines applications temps réel, pour les transactions de respecter leurs contraintes de temps. Certains travaux explorent donc un autre type de bases de données où ce problème des E/S ne se pose pas : les bases de données résidant en mémoire principale (appelées en anglais MMDB, Main-Memory DataBases). Le système a donc plus de chances de respecter les échéances des transactions.

5.2. *Bases de données en mémoire principale*

Tous les systèmes informatiques aussi rapides soient-ils sont confrontés au problème des entrées/sorties qui retardent considérablement les traitements et l'élaboration de résultats. Ce retard peut être toléré quand il s'agit de systèmes informatiques traditionnels ou de SGBD temps réel à contraintes non strictes, où seule la qualité du service offert est altérée. Pour les SGBD temps réel à contraintes strictes et non critiques, le problème devient plus significatif, puisque les transactions qui ratent leurs échéances sont abandonnées, engendrant ainsi du travail inutile pour le système. Dans le cas des SGBD temps réel à contraintes strictes et critiques, la gestion des E/S devient un des problèmes cruciaux à résoudre pour pouvoir espérer respecter les contraintes temporelles des applications. Pour éviter ce problème, dans beaucoup de travaux sur les SGBD temps réel, il est supposé que les bases de données à gérer résident entièrement (ou en grande partie) en mémoire principale [GRA 92]. Ces travaux concernent la réalisation et l'implantation de prototypes, la conception de modèles pour la simulation d'algorithmes d'ordonnancement et de CC des transactions temps réel [HON 93, HUA 96, TSE 96, ULU 96], etc. Par exemple :

- Tseng et al. [TSE 96] ont étudié un ordonnancement basé sur les valeurs des priorités affectées aux transactions ayant terminé leur exécution avant

échéance. Des comparaisons entre certains algorithmes d'assignation de priorités appliqués aux bases de données résidentes en mémoire principale ont été effectuées et des études de coûts relatifs au fait de maintenir tout ou partie de la base en mémoire principale ont aussi été effectuées. Les résultats des expérimentations effectuées montrent notamment comment on peut optimiser la taille de la mémoire principale à affecter à la base de données en fonction des types de transactions.

– Huang et al [HUA 96] ont travaillé sur les techniques de pose de points de contrôle dans les bases de données résidentes en mémoire principale. Les objectifs de la technique proposée sont de réduire le temps de reprise du système suite à une défaillance et d'augmenter le pourcentage de transactions qui respectent leurs échéances. L'algorithme proposé repose sur la partition des données selon leur nature : partition des données persistantes selon leur fréquence de mise à jour et partition des données temps réel selon leurs durées de validité. Les partitions des données temps réel ayant de courts intervalles de validité sont mises à jour plus souvent que les autres partitions leur donnant ainsi davantage d'opportunités d'être transférées en mémoire secondaire. Cependant, la pose de points de contrôle ne s'effectue pas sur les partitions des données temps réel ayant des durées de validité inférieures à un seuil fixé, diminuant ainsi la surcharge du système et augmentant la rapidité du processus de reprise. Des simulations ont montré [HUA 96] que cette technique de pose des points de contrôle améliore la technique proposée par [HAG 86] pour la pose des points de contrôle dans les MMDB (Main-Memory DataBases).

5.3. *SGBD temps réel et systèmes d'exploitation*

Un SGBD 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 94, LEH 95]. 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 [RAM 94]. Certains sont des SE dédiés, d'autres sont des extensions de SE multi-tâches existants. Parmi ces systèmes, on peut citer : RT-Mach [TOK 90] qui a servi de noyau au prototype de SGBD temps réel StarBase [KIM 97], ARTS [TOK 89] qui a servi de noyau aux prototypes de SGBD temps réel DeeDs [AND 96] et RTDB [SON 93], et Chorus –

un SE réparti enrichi par des fonctions temps réel [GIE 90, CHO 92] – qui a servi comme noyau du prototype de SGBD temps réel REACH [ULU 96].

D'autres SE temps réel peuvent également servir de support à des SGBD temps réel. Par exemple, la réalisation de SGBD temps réel répartis pourrait être facilitée par l'utilisation de plates-formes de type CORBA, étendues par des fonctions temps réel où des travaux commencent à voir le jour [WOL 97a, 97b].

Certains auteurs [GRA 92] 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 [WOL 97c] qui intègre les fonctions d'un noyau de SE temps réel à la norme POSIX.

5.4. *Offre industrielle et état de la normalisation*

Les applications temps réel sont souvent liées à des domaines critiques (avionique, procédés industriels, surveillance de malades, contrôle de la circulation urbaine, etc.) où la défaillance du système informatique peut avoir des conséquences très graves. Par conséquent, les utilisateurs de ces applications sont souvent très réservés face à de nouveaux produits à intégrer dans leurs applications. Il faut qu'un produit soit expérimenté avec succès, sur plusieurs années, avant de l'accepter dans les systèmes temps réel. C'est là une des raisons qui font que les SGBD temps réel ne sont pas encore une pratique courante dans le domaine du temps réel. Encore, plus important, devant le manque d'investissement des industriels dans ce nouvel outil, les produits offerts sont rares et la normalisation est à un stade embryonnaire.

On remarque cependant quelques tentatives d'offres commerciales de SGBD dits temps réel, notamment aux Etats-Unis. Comme la littérature est quasi-inexistante sur de tels produits, nous avons exploité les nouvelles possibilités de communication des entreprises, à savoir l'Internet[7]. Nous avons recensé plusieurs produits : *IndustrialSQL Server* [www.syntronix.com/html/insql2.htm], *Tachys* [www.probita.com/tachys2.html], *Polyhedra* [www.polyhedra.com/], *RapideBase* [www.vtt.fi/tte/projects/rapid/], *Portfolio* [www.tbroker.co.uk/offers/phrt1.html], *TimesTen MMDB Manager* [www.timesten.com/main.html]. Notre étude de ces produits montre qu'il s'agit surtout de systèmes garantissant de meilleurs temps de réponse que les SGBD traditionnels. Les performances en termes de temps de réponse sont obtenues grâce au stockage des données en mémoire centrale.

Le deuxième aspect important pour les utilisateurs est celui de la normalisation. À notre connaissance, une seule proposition a été faite concernant des extensions temps réel au langage SQL : il s'agit du langage RT-SQL [FOR 94, PRI 97]. Cette proposition est le résultat de travaux coordonnés entre différents organismes américains tels que le groupe de travail sur les interfaces standards pour les bases de

[7] C'est la raison pour laquelle nous donnons des adresses de sites Web de présentation des produits.

données du département de la défense, le groupe de travail de gestion prédictible d'informations temps réel de l'ANSI (American National Standard Institute), l'université de Rhode Island, etc. Cependant le projet de cette norme n'a pas encore été adopté, probablement parce qu'il n'existe pas encore de consensus sur la manière de concevoir et d'implanter un SGBD temps réel ; le problème des langages et d'autres interfaces avec l'environnement est repoussé à plus tard.

6. Conclusion

La manipulation et la gestion d'importantes quantités de données nécessitent des mécanismes adéquats, en particulier pour assurer l'intégrité de ces données lorsqu'elles sont utilisées par plusieurs activités concurrentes dans des applications de grande taille, comme les applications temps réel. En effet, l'accès concurrent et avec des contraintes de temps ne peut utiliser des données correctes et les laisser correctes que si des mécanismes adéquats sont mis en œuvre. Comme nous l'avons souligné dans les sections précédentes, les SGBD temps réel constituent une alternative pour répondre à ces besoins.

Bien que des travaux sur les SGBD temps réel soient de plus en plus nombreux, ils ne sont pas encore suffisamment avancés pour susciter l'intérêt des industriels et les inciter à investir dans ce nouveau domaine et encourager la normalisation de ces systèmes. C'est la conclusion à laquelle ont abouti plusieurs participants à la première rencontre internationale sur les SGBD temps réel [BES 96] qui ont préconisé de travailler sur la réalisation de nombreux prototypes afin de tester la validité des algorithmes de contrôle de concurrence de transactions temps réel dans des conditions réelles de charges [AND 96, ARA 96]. Ces prototypes doivent montrer la valeur ajoutée des SGBD temps réel. D'autre part, excepté le domaine de contrôle de concurrence des transactions, étudié abondamment, beaucoup de problèmes relatifs à la conception et réalisation de SGBD temps réel, ainsi que la conception des applications qui vont faire appel aux services de tels SGBD, ne sont pas abordés ou commencent à peine à l'être et constituent donc des sujets qui devraient intéresser la communauté de recherche en informatique. Il s'agit notamment des problèmes suivants :

- Spécification des contraintes temporelles imposées aux données et aux transactions par des approches formelles et validation de ces contraintes. La seule étude, à notre connaissance, qui essaie d'appliquer les techniques de spécification de STR aux SGBD temps réel est celle développée par Xiong et Ramamritham [XIO 97a] qui proposent un ensemble d'axiomes sur les différents événements et actions pour spécifier des transactions dans un SGBD actif et temps réel. Leur approche s'apparente à la logique RTL [JOH 86]. Mais, comme le soulignent bien les auteurs de cette étude, on est encore loin d'une approche complète de spécification et de validation des contraintes temporelles dans les SGBD temps réel.

- Approche de conception d'une application temps réel fondée sur l'utilisation d'une base de données, en s'inspirant à la fois des approches de conception d'applications fondées sur les bases de données classiques (introduction de données et leurs attributs, introduction de traitements) et des approches de conception d'applications temps réel (introduction de tâches et événements).
- Langages standards pour la description des données et des traitements.
- Outils d'estimation des temps d'exécution des transactions temps réel de manière à prédire si elles peuvent ou non respecter leurs échéances.
- Analyse quantitative et qualitatives des protocoles de contrôle de concurrence.
- Restauration d'un état cohérent de la base suite à une défaillance. L'approche classique d'utilisation des copies de sauvegarde sur disque (réplication de données) pour redémarrer le système, n'est pas adéquate pour les SGBD temps réel à cause de la durée limitée de la validité des données temps réel. Les techniques de reconfiguration, de changement de modes de marche et d'utilisation de données redondantes, connues dans le domaine des systèmes temps réel, seront certainement à la base des techniques de restauration de bases de données temps réel.
- Répartition des données temps réel sur plusieurs sites pour accroître la disponibilité du système global et pour avoir davantage de puissance de calcul. Les travaux développés à la fois sur les bases de données réparties et sur les systèmes temps réel répartis devraient être conjugués pour développer des SGBD temps réel et répartis. Il reste beaucoup de choses à faire si l'on juge de l'état d'avancement des travaux sur les systèmes temps réel et répartis qui sont encore à un stade de balbutiement.
- Approche de relaxation de certaines des propriétés ACID des transactions pour leur trouver un équivalent adapté aux contraintes du temps réel. Quelques pistes ont commencé à être explorées : la similarité des données [XIO 96b], le principe de retard forcé [XIO 96a] et celui de la delta-sérialisabilité [KUO 96].
- Stockage des données temps réel : mémoire principale et/ou mémoire secondaire. Une démarche systématique est nécessaire pour savoir quels types de données et pour quels types de systèmes et de charge du système on peut considérer que le stockage en mémoire est la solution efficace. Par ailleurs, la mémoire principale étant volatile, il n'est pas concevable que la base de données y réside indéfiniment. En effet, il existe un risque non négligeable de perdre toutes les données de la base. Par conséquent, davantage de travaux sur la pose des points de contrôle sont nécessaires de manière à sauvegarder sur disque une version (même non fraîche) de la base, en évitant de perturber le processus de gestion normale des transactions.
- Gestion des tampons d'entrées/sorties et de l'ordonnancement des opérations d'entrées/sortie pour les bases de données sur disque.

Enfin, pour ce qui concerne nos travaux futurs, nous travaillerons principalement sur deux aspects : analyse d'algorithmes d'ordonnancement de transactions à contraintes temporelles strictes critiques et développement de plate-forme d'expérimentation d'algorithmes de contrôle de concurrence de transactions.

7. Remerciements

Les auteurs tiennent à remercier les lecteurs anonymes pour leurs remarques constructives qui ont permis d'améliorer l'article.

8. Bibliographie

[ABB 88] ABBOT R. and GARCIA-MOLINA H., « Scheduling real-time transactions: a performance evaluation », *14th Int. Conf. on VLDB*, p. 1-12, 1988.

[ABB 90] ABBOT R. and GARCIA-MOLINA H., « Scheduling I/O request with deadlines: a performance evaluation », *11th RTS Symposium*, p. 113-124, 1990.

[ADI 95] ADIBA M., « STORM: a structural and temporal object-oRiented multimedia database system », *Actes des 11ième Journées Bases de Données Avancées*, p. 413-429, Nancy, France, 1995.

[ALU 90] ALUR R. and HENZINGER T.A., « Real-Time logics: complexity and expressiveness », *5th Symp. on Logics in Comp. Science*, Philadelphia, p. 390-401, 1990.

[AND 96] ANDLER S.F., HANSSON J., ERIKSSON J., MELLON J., BERNDTSSON M. and EFTRING B., « DeeDS: towards a distributed and active real-time database system », *ACM SIGMOD Record*, vol. 15, nº 1, p. 38-40, 1996.

[ARA 96] ARANHA R.F.M., GANTI V., NARAYANAN S., MUTHUKRISHNAN C.R., PRASAD S.T.S. and RAMAMRITHAM K,. « Implementation of a real-time database system », *Information Systems, Special Issue on Real-Time Databases*, vol. 21, n° 1, p.55-74, 1996.

[BAY 82] BAYER R., ELHARDT K., HEIGERT J. and RESEIR A., « Dynamic timestamp allocation for transactions in database systems », *2nd Int. Symp. on Distruted Databases*, p. 9-20, 1982.

[BER 83] BERNSTEIN P. and GOODMAN N., « Multiversion concurrency control - theory and algorithms », *ACM Transaction on Database Systems*, vol. 8, n° 4, p. 465-484, 1983.

[BER 84] BERRY G. and COSSERAT L., « The ESTEREL synchronous programming language and its mathematical semantics », *L.N.C.S.* vol. 197, 1984.

[BER 87] BERNSTEIN P., HADZILACOS V. and GOODMAN N., *Concurrency control and recovery in database systems*, Addison-Wesley, 1987.

[BER 91] BERTHOMIEU B., and DIAZ M., « Modeling and verification of time dependant systems using time Petri nets », *IEEE T.S.E.*, vol. 17, n° 3, p. 259-273, 1991.

[BES 94] BESTAVROS A. and BRAOUDAKIS S., « Timeliness via speculation for real-time database », *14th IEEE RTS Symposium*, 1994.

[BES 96] BESTAVROS A., LIN K-J. and SON S., « Report on first international workshop on real-time database systems », *SIGMOD Record*, vol. 25, n° 3, p. 50-52, 1996.

[BOK 87] BOKSENBAUM C., CART M., FERRIE J. and PONS J., « Concurrent certifications by intervals of timestamps in distributed database systems », *IEEE T.S.E.*, vol. 13, n° 4, p. 409-419, 1987.

[BRA 95a] BRANDING B. and BUSHMANN A., « On providing soft and hard real-time capabilities in an active DBMS », *Int. Workshop on ARTDB*, p. 158-169, 1995.

[BRA 95b] BRAOUDAKIS S., « Concurrency control protocols for real-time databases », PhD dissertation, Computer Science Department, Boston University, 1995.

[BUC 95] BUCCI G., CAMPANAI M. and NESI P., « Tools for specifying real-time systems », *J. of Real-time Systems*, vol. 8, p. 117-172, 1995.

[BUR 94] BURNS A. and WELLINGS A.J., « HRT-HOOD: A structured design method for hard real-time systems », *J. of Real-time Systems*, vol. 6, p. 73-114, 1994.

[CHA 98] CHAKRAVARTHY S., HONG D-K. and JOHNSON T., « Real-time transaction scheduling: a framework for synthesizing static and dynamic factors », *J. of RTS*, n° 14, p. 135-170, 1998.

[CHO 92] CHORUS SYSTÈMES, « CHORUS/Kernel V3r4.0 specification and interface », *Technical report, CHORUS Systèmes*, 1992.

[COL 94] COLLET C., COUPAYE T. and SVENSEN T., « NAOS: efficient and modular capabilities in an object-oriented database system », *20th Int. Conf. on VLDB*, p. 132-143, 1994.

[COU 93] COURTIAT J.P., DE CAMARGO M.S. et SAÎDOUNI D.E., « RT-LOTOS : LOTOS temporisé pour la spécification de systèmes temps réel », *Actes de CFIP'93, Montreal*, p. 427-441, Editions Hermès 1993.

[COU 98] COUPAYE T. et COLLET C., « Modèle de comportement des SGBD actifs : caractérisation et comparaison », *TSI*, vol. 17, n° 3, p. 299-328, 1998.

[EME 86] EMERSON E.A. and HALPERN J., « Sometimes and not never revisited: on branching versus linear time temporal logic », *J. of ACM*, vol. 33, n° 1, 1986.

[ESW 76] ESWARAN K.P., GRAY J.N., LORIE R.A. and TRAIGER I.L., « The notion of consistency and predicate locks in a database system », *CACM*, vol. 19, n° 11, p. 624-633, 1976.

[FOR 94] FORTIER P., WOLFE V.F., PRICHARD J.J., « Flexible real-time SQL transactions », *IEEE RTS Symposium*, 1994.

[GHE 90] GHEZZI C., MANDRIOLLI D. and MARZENTI A. « TRIO, a logical language for executable specifications of real-time systems », *J. of System Soft.*, vol. 12, p. 107-123, 1990.

[GIE 90] GIEN M., « Microkernel architecture-key to modern systems design », Unix Review. 1990.

[GOM 86] GOMAA H., « Software development for real-time systems », *Communication of ACM*, vol. 29, n° 7, p. 657-668, 1986.

[GRA 92] GRAHAM M.H., « Issues in real-time data management », *J. of RTS*, vol.4, n° 3, p. 185-202, 1992.

[HAG 86] HAGMANN R.B., « A crash recovery scheme for a memory-resident database system », *IEEE Transactions on Computers*, vol. C-35, n° 9, 1986.

[HAR 87] HAREL D., PNEULI A., SCHMIDT J.P. and SHERMAN R., « On the formal semantics of Statecharts », *2nd IEEE Symposium on Logic in Computer Science Ithaca*, New York, p. 54-64, 1987.

[HAR 90] HARITSA J.R., CAREY M.J. and LIVNY M., « Dynamic real-time optimistic concurrency control », *IEEE RTS Symposium*, Florida, p. 94-103, 1990.

[HAR 92] HARITSA J.R., CAREY M.J. and LIVNY M., « Data access scheduling in firm real-time database systems », *J. of RTS*, vol.4, n° 3, p. 203-241, 1992.

[HON 93] HONG D., JOHNSON T. and CHAKRAVARTHY S., « Real-time transactions scheduling: cost conscious approach », *ACM SIGMOD. Int. Conf. on Management of Data*, p. 197-206, 1993.

[HUA 91] HUANG J., STANKOVIC J.A. and RAMAMRITHAM K., TOWSLEY D., « On using priority inheritance in real-time database », *12th RTS Symposium*, p. 552-561, 1991.

[HUA 96] HUANG J. and LE GRUENWALD., « An update-frequency-valid-interval partition checkpoint technique for real-time main memory databases », *1st Int. Workshop on RTDBS. Issues and Applications*, p. 130-137, California 1996.

[JOH 86] JAHANIAN F. and MOK A.K., « Safety analysis of timing properties in real-time systems », *IEEE T.S.E.*, vol. 12, n° 3, p.890-904, 1986.

[KIM 91] KIM W. and SRIVASTAVA J., « Enhancing real-time DBMS performance with multiversion data and priority based disk scheduling », *12th RTS Symp*, 1991.

[KIM 94] KIM Y-K. and SON S.H., « Predictability and consistency in real-time database systems », *Principles of Real-Time Systems*, Ed. Prentice-Hall, p. 502-524, 1994.

[KIM 97] KIM Y-K. and SON S.H., « Developing a real-time DB: the StarBase experience », *RTDBS. Issues and Applications, edited by Bestavros, Lin and Son*, p. 305-324, 1997.

[KOY 89] KOYMANS R., « Specifying message passing and time critical systems with temporal logic », PhD dissertation, Eindhoven University, The Netherlands, 1989.

[KUN 81] KUNG H.T. and ROBINSON J.T., « On optimistic methods for concurrency control », *ACM Transactions on Database Systems*, vol. 6, n° 2, p. 213-226, 1981.

[KUO 96] KUO T-W. and MOK A.K., « Real-time database - similarity semantics and ressource scheduling », *ACM SIGMOD Record*, vol. 25, n° 1, p. 18-22, 1996.

[LAM 97] LAM K-W., SON S.H. and HUNG S-L., « A priority ceiling protocol with dynamic adjustment of serialization Order », *13th Int. Conf. on Data Engeneering* (ICDE'97), Birminghan, U.K., p. 552-561, 1997.

[LED 93] LEDUC G. and LÉONARD L. « A timed LOTOS supporting a dense time domain and including new timed operators », *Diaz M. and Groz R., eds., Formal description techniques, V* (North-Holland, 1993), p. 87-102, 1993.

[LEH 95] LEHR M.R., KIM Y-K. and SON S.H., « Managing contention and timing constraints in a real-time database system », *16th IEEE RTS Symp.*, Italy, 1995.

[MAM 85] MAMMERI Z., « Conception d'applications réparties en commande de processus : une approche par la structuration des données », Thèse de Doctorat, Institut National Polytechnique de Lorraine, Nancy, 1985.

[MAM 98] MAMMERI Z., « Expression et dérivation des contraintes temporelles dans les applications temps réel », *APII-JESA*, vol. 32, n° 5-6, p. 609-644, 1998.

[MEN 82] MENASCE D. AND NAKANISHI T., « Optimistic versus pessimistic concurrency control mechanisms in database management systems », *Information Systems*, vol. 7, n° 1, 1982.

[MOL 85] MOLLOY K., « Discrete time stochastic Petri Nets », *IEEE T.S.E.*, vol. 11, n° 4, p. 417-423, 1985.

[MOR 94] MORZENTI A. and PIETRO P.S., « Object-orientd logical specification of time critical systems », *ACM T.O.S.E.M.*, vol. 3, n° 1, p. 56-98, 1994.

[NAK 93] NAKAZATO H., « Issues on synchronizing and scheduling tasks in real-time database systems », PhD thesis, University of Illinois at Urbana-Champaign, 1993.

[OST 87] OSTROFF J.S. and WONHAM W., « Modeling and verifying real-time embedded computer systems », *IEEE RTS Symposium*, p. 124-132, 1987.

[OST 89] OSTROFF J., « Temporal logic for real-time systems », *Advanced Software Development Series*, Research Studies Press, Taunton, Somerset, U.K., vol. 1, 1989.

[PRI 97] PRICHARD J.J. and FORTIER P., « Standardizing real-time databases - RTSQL », *Real-Time Database and Information Systems*, Eds Bestavros A., Fay-Wolfe V., Kluwer Academic Publihers, p. 289-310, 1997.

[PUR 94] PURIMETLA B., SIVASANKARAN R.M., RAMAMRITHAM K. and STANKOVIC J.A., « Real-time databases: issues and applications », *Principles of RTS*, ed. Printice Hill, 1994.

[RAM 93a] RAMAMRITHAM K., « Real-time databases », *J. of Distributed and Parallel Databases*, vol. 1, n° 2, p. 199-226, 1993.

[RAM 93b] RAMAMRITHAM K., « Time for real-time temporal databases ? », *Int. Workshop on an Infrastructure for Temporal Databases*, 1993.

[RAM 94] RAMAMRITHAM K. and STANKOVIC J.A., « Scheduling algorithms and operating systems support for real-time systems », *Proc. of the IEEE*, vol. 82, n° 1, p. 55-67, 1994.

[RAM 96a] RAMAMRITHAM K., SIVASANKARAN R.M., STANKOVIC J.A., TOWSLEY D.T., and XIONG M., « Integrating temporal, real-time, and active databases », *ACM SIGMOD Record. Special Issues on RTDBS*, vol. 25, n° 1, p. 8-12, 1996.

[RAM 96b] RAMAMRITHAM K., « Where do time constraints come from and where do they go ? », *Int. J. of Database Management*, vol. 7, n° 2, p. 4-10, 1996.

[RAZ 89] Razouk R.R., and Gorlick M.M., « A real-time interval logic for reasoning about execution of real-time program », *ACM/SIGSOFT'89* (TAV3), 1989.

[SCH 98] Scholl P-C., Fauvet M-C. and Canavaggio J.F., « Un modèle d'historique pour un SGBD temporel », *TSI*, vol. 17, n° 3, p. 379-399, 1998.

[SHA 90] Sha L., Rajkumar R. and Lehocsky J.P., « Priority inheritance protocols: an approach to real-time synchronisation », *IEEE Transactions on Computers*, vol. 39, n° 9, p. 1175-1185, 1990.

[SHA 91] Sha L., Rajkumar R., Son S.H. and Chang C.H., « A real-time locking protocol », *IEEE Transactions on Computers*, vol. 40, n° 7, p. 782-800, 1991.

[SNO 95] Snodgrass R.T., *The TSQL2 language design comittee: The TSQL2 Temporal Query Language*, Kluwer Academic Publishers, 1995.

[SON 88] Son S.(ed.), *ACM SIGMOD Record: Special Issue on Real-Time Databases*, 1988.

[SON 92a] Son S.H., Lee J. and Lin Y., « Hybrid protocols using dynamic adjustment of serialization order for real-time concurrency control », *J. of RTS*, vol. 4, n° 3, p. 269-276, 1992.

[SON 92b] Son S.H. and Liu J.W.S., « How well can data temporal consistency be maintained ? », *IEEE Symp. on Computer-Aided Control System Design*, 1992.

[SON 93] Son S.H., George D.W. and Kim Y-K., « Developing a database system for time-critical applications », *IEEE Int. Conf. on Database and Expert System Applications* (DEXA'93), Prague, Czech, p. 313-324, 1993.

[SON 95a] Son S.H. and Liu J.W.S., « Maintaining temporal consistency: pessimistic vs. optimistic concurrency control », *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, n° 5, 1995.

[SON 95b] Son S.H. and Park S., « A priorty based scheduling algorithm for real-time databases », *J. of Information Science and Engineering*, n° 11, p. 233-248, 1995.

[STA 88] Stankovic J.A. and Ramamritham K., « Hard real-time systems: a tutorial », IEEE Computer Society Press, 1988.

[STA 95] Stankovic J.A., Spuri M., Di Natale M. and Buttazzo G.C., « Implications of classical scheduling results for real-time systems », *IEEE Computer*, vol. 28, n° 8, p. 16-25, 1995.

[TAN 93] Tansel A., Clifford J., Gadia V., Segev A. and Snodgrass R., « Temporal databases: theory, design and implementation », The Benjamen/Cummings publishing Company. Redwood City. California. 1993.

[TOK 89] Tokuda H. and Mercer C., « ARTS: A distributed real-time kernel », *ACM Operating System Review*, vol. 23, n° 3, p. 29-53 , 1989.

[TOK 90] Tokuda H., Nakajima T. and Rao P., « Real-time mach: toward predictable real-time systems », *USENIX Mach Workshop*, 1990.

[TSE 96] Tseng S-M., Chin Y.H. and Yang W-P., « Scheduling value-based transactions in real-time main-memory databases », *1st Int. Workshop on RTDBS, Issues and Application*, California, p. 111-117, 1996.

[UIT 93] Union Internationale des Télécommunications, « SDL : Langage de description et de spécification », *Recommandation UTI-T Z100*, 1993.

[ULU 96] ULUSOY Ö. and BUCHMANN A.P., « Exploiting main memory DBMS features to improve real-time concurrency control protocols », *Special section on RTDBS of ACM SIGMOD Record*, vol. 25, n° 1, p. 23-25, 1996.

[WAR 86] WARD P., « The transformation scheme: an extension of the dataflow diagram to represent control and timing », *IEEE T.S.E.*, vol. 12, n°2, p. 198-210, 1986.

[WID 95] WIDOM J. and CERI S., *Active database systems*, Morgan Kauffman Publishers, 1995.

[WOL 97a] WOLFE V.F., DIPIPPO L.C., GINIS R., SQUADRITO M., WOHLEVER S., ZYKH I. and JOHNSTON R., « Expressing and enforcing timing constraints in a dynamic real-time CORBA system », Technical Report TR97-252, 1997.

[WOL 97b] WOLFE V.F., DIPIPPO L.C., GINIS R., SQUADRITO M., WOHLEVER S., ZYKH I. and JOHNSTON R., « Real-time CORBA », Technical Report TR97-256, 1997.

[WOL 97c] WOLFE V.F., PRICHARD J.J., DIPIPPO L.C. and BLACK J., « Developping a real-time DB: the StarBase experience », *Real-Time Database Systems*, p. 279-303, 1997.

[XIO 96a] XIONG M., STANKOVIC J.A., RAMAMRITHAM K., TOWSLEY D. and SIVASANKARAN R.M., « Maintaining temporal consistency: issues and algorithms », *1st Int. Workshop on RTDBS. Issues and Applications*, p. 1-6, California 1996.

[XIO 96b] XIONG M., SIVASANKARAN R.M., STANKOVIC J.A., RAMAMRITHAM K. and TOWSLEY D., « Scheduling transactions with temporal constraints: exploiting data semantics », *17th IEEE RTS Symposium*, Washington, p. 240-251, 1996.

[XIO 97a] XIONG M. and RAMAMRITHAM K., *Towards the specification and analysis of transactions in real-time active databases*, RTDB and Information Systems, Edited by Bestavros and Fay-Wolfe. Kluwer Academic Publishers, p. 327-354, 1997.

[XIO 97b] XIONG M., SIVASANKARAN R.M., STANKOVIC J.A., RAMAMRITHAM K. and TOWSLEY D., *Scheduling access to temporal data in real-time databases*, RTDBS. Issues and Applications, Edited by Bestavros, Lin and Son, Kluwer Academic Publishers, p. 167-191, 1997.

Claude Duvallet *prépare une thèse sur les SGBD Temps Réel au sein du LIH (Laboratoire d'Informatique du Havre). Ses travaux portent principalement sur le respect des contraintes temporelles des transactions, notamment dans un contexte distribué, ainsi que sur l'adaptation des propriétés ACID pour les transactions temps réel.*

Zoubir Mammeri *est professeur à l'université Paul Sabatier de Toulouse. Il est membre de l'IRIT. Ses activités de recherche incluent les applications et systèmes temps réel, les systèmes répartis, les réseaux, le multimédia et l'ordonnancement temps réel.*

Bruno Sadeg *est maître de conférences à la Faculté des Sciences de l'Université du Havre. Il travaille au sein du LIH sur les SGBD Temps Réel. Ses travaux portent sur les algorithmes de contrôle de concurrence et d'ordonnancement des transactions temps réel*

strictes critiques, ainsi que sur les problèmes de cohérence dans les bases de données temps réel.