Les systèmes de base de données temps réels

Pokrovskaya Natalia, Kabbali Nadia

Année académique 2008 - 2009

Table des matières

Chapitre 1

Introduction

De nos jours, il y a de plus en plus de domaines, dans lesquels les données d'environnement sont gérées par des ordinateurs, comme, par exemple, les ventes boursières, la surveillance des patients dans des hôpitaux, les productions automatisées, le contrôle militaire et aérien. On voit une énorme augmentation du nombre d'applications dites temps réel, c'est-à-dire des applications qui doivent non seulement respecter les contraintes logiques et de cohérence imposées par le système, mais aussi les contraintes temporelles, liées à la vitesse d'exécution des opérations, ainsi qu'au respect de leur échéances. De plus, ces applications doivent pouvoir manipuler des quantités importantes d'informations nécessaires pour leur bon fonctionnement. Ainsi, l'utilisation des bases de données s'avère nécessaire et indispensable pour ce type de systèmes.

Mais, contrairement à des bases de données classiques, les bases de données temps réel, détaillées dans cet article, doivent pouvoir aussi répondre à des contraintes temporelles introduites par les systèmes temps réels, tout en assurant les contraintes d'intégrité et de cohérence, la possibilité de partager des données, la reprises après les pannes, etc., fournis par les systèmes de gestion de bases de données classiques (SGBD). Donc, en quelque sorte, les bases de données temps réel représentent les bases de données classiques auxquelles sont rajoutées des contraintes temporelles propres aux systèmes temps réels.

Chapitre 2

Système de gestion de bases de données classiques

Ce sont des systèmes qui permettent d'interagir avec les bases de données. Leur but principal est de pouvoir définir les bases de données, fournir les moyens pour les consulter et les mettre à jour. Ce type de système permet de gérer de manière efficace des quantités importantes de données. On désigne par transaction une ou plusieurs opérations sur les données de la base. Ces transactions doivent posséder les propriétés ACID (Atomicité-Cohérence-Isolation-Durabilité)([CONT])

- Atomicité : cette propriété signifie que chaque transaction est considérée comme une unité atomique d'exécution (pas importe le nombre d'opérations dont elle est composée). C'est-à-dire que pour qu'une transaction soit exécutée, toutes les opérations doivent être exécutées. Si une opération ne s'est pas exécutée correctement, toute la transaction est annulé.
- Cohérence - étant donnée une base de données cohérente, l'exécution d'une transaction fait passer la base dans un autre état cohérent
- Isolation - les transactions sont dites isolées les unes des autres, c'est-à-dire que les modifications effectuées par l'une d'elles ne deviennent visibles et exploitables par les autres transactions qu'après sa phase de validation (Commit).
- Durabilité - cette propriété indique qu'une fois la transaction validée (Commit), les modifications qu'elle a effectuées sur la base pendant son exécution deviennent permanentes.

Un des plus grands problèmes pour assurer la cohérence d'une base vient du fait qu'il doit être possible d'avoir les accès simultanés aux mêmes ressources d'une base. Pour cela, il existe des protocoles de contrôle de concurrence. La plupart de ces protocoles utilisent l'idée de transactions sérialisables. La sérialisabilité de deux transactions peut être définie de la manière suivante (Bernstein et al., 1987) : *L'exécution de deux transactions est dite sérialisable si l'exécution entrelacée de leurs opérations fournit les mêmes résultats que leur exécution en série (l'une après l'autre)*. Et donc, pour assurer l'accès concurrent aux données, les protocoles de contrôle de concurrence veillent à ce que l'exécution de transactions concurrentes soit sérialisable.

L'objectif de performances des systèmes de gestion de bases de données classiques est de garantir l'exécution d'une transaction, sans se soucier du temps d'exécution de chaque opération composant

la transaction. La seule garantie temporelle qu'un système peut fournir est le temps **moyen** d'exécution d'une transaction.

Chapitre 3

Systèmes temps réel

L'objectif des systèmes temps réel est la gestion efficace des données tout en respectant les contraintes temporelles qui leur sont imposées ([IDOU]). Les systèmes temps-réel fournissent à cet égard des algorithmes efficaces pour ordonnancer les tâches temps réel de manière à respecter leurs échéances.

La raison d'être des systèmes temps réel est que la fiabilité du système dépend non seulement de la l'exactitude des traitements effectués, mais également du temps dans lequel ces résultats sont fournis. La limite de ce type de système est son incapacité de pouvoir manipuler des grandes quantités d'informations (d'où l'intérêt de les coupler avec les bases de données).

Chapitre 4

SGBD temps réel

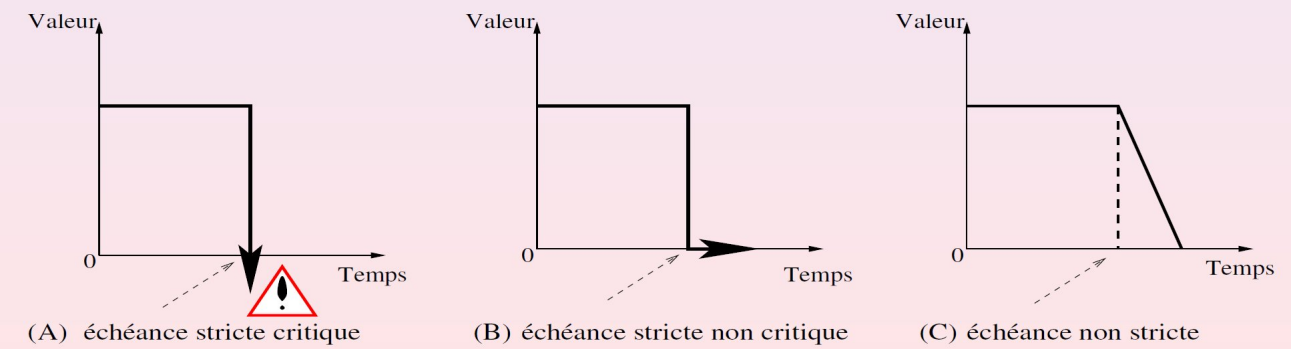
Vu les obstacles rencontrés par les systèmes temps réel et les bases de données classiques afin de gérer de grande quantité de données temps réels, les systèmes de gestion de bases de données temps réel voient leur naissance. On définit un système de gestion de bases de données temps réel comme étant un système de gestion de bases de données dans lequel on associe des contraintes temporelles à chaque transaction et chaque donnée possède un intervalle de temps pendant lequel elle est valide. Ce type de système permet de gérer la double contrainte : la cohérence logique de données (qu'ils héritent de bases de données classiques) et la cohérence temporelle de données (venue de systèmes temps réel). Les données sont collectées dans des intervalles de temps discrets, ce qui représente en quelque sorte une approximation de la réalité. Les contraintes temporelles dans un SGBDTR peuvent prendre des formes différentes (par exemple périodes de leur invocations, ou bien échéances absolues ou relatives, etc.) et peuvent s'appliquer à divers composants d'une application :

– 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)
– à des événements (par exemple, tel événement doit apparaître x unités de temps après tel autre événement) ([IDOU]).

Comme on a des contraintes temporelles liées au temps d'exécution des transactions, ces dernières peuvent être divisées en trois catégories, selon les conséquences engendrées lorsque ces transactions ratent leur échéance.

1. Transactions à échéances strictes critiques (hard deadline transactions) : ce sont des transactions dont le non respect de leurs échéances peut mener à des conséquences graves sur le système ou l'environnement contrôlé (Par exemple, dans le cas de système de surveillance de patients dans des hôpitaux, le non respect d'échéance pour la mise à jour d'information sur le rythme cardiaque peut mener à des conséquences graves, voire même à la mort)
2. Transactions à échéances strictes non critiques (firm deadline transactions) : si une transaction manque son échéance, elle devient inutile pour le système. Elle est donc abandonnée dès qu'elle rate son échéance.
3. Transactions à échéances non strictes (soft deadline transactions) : si une transaction manque son échéance, le système peut ne pas l'abandonner immédiatement. En effet, l'utilité de la

donnée ne fait que décroitre. Les applications multimédia sont également un très bon exemple de système manipulant des transactions de type soft.



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

Les données enregistrées dans une base de données doivent pouvoir être accédées simultanément par plusieurs transactions, de telle sorte que ces dernières puissent s'exécuter de manière concurrente. Les techniques de transactions sérialisables, utilisées pour des bases de données classiques, ne sont pas applicables pour les bases de données temps réel, car les transactions qui ne respectent pas leurs échéances doivent être rejetées, car on doit pas seulement respecter les contraintes logiques, mais aussi les contraintes temporelles.

Donc, pour pouvoir répondre aux deux contraintes au même temps, on va utiliser les techniques de contrôle de concurrence de bases de données traditionnelles d'une part et d'autre part, on utilisera les techniques d'ordonnancement de tâches des systèmes temps réel (pour respecter les contraintes temporelles).

Toutes les techniques de contrôle de concurrence de transactions dans les SGBD temps réel sont principalement basées sur deux types de politiques ([DUVA]) :

- Politiques de contrôle de concurrence optimistes - 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 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.
- Politiques de contrôle de concurrence pessimistes - 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.

4.2. Idées fausses

On va maintenant présenter les idées fausses que l'on a sur des bases de données temps réel ([RTDS]).

- **Les avancées dans la technologie matérielle des ordinateurs vont satisfaire les besoins des bases de données temps réel** : Le fait d'avoir fait plus vite n'implique pas d'avoir fait dans le temps. Le matériel tout seul ne peut pas garantir un bon ordonnancement de transactions de telle sorte qu'elles respectent leurs échéances. Par exemple, si une transaction utilise des données obsolètes plus vite, elle ne va pas pour autant devenir correcte.
- **Les avancées dans les méthodes de conception et développement des bases de données vont satisfaire les besoins des bases de données temps réel** : Des protocoles de validation plus rapides, des nouvelles techniques de transfert de requêtes permettent d'"accélérer" le traitement des transactions dans une bases de données mais elles ne peuvent pas garantir la rencontre de toutes les échéances du système
- **Le calcul temps réel est équivalent à du calcul rapide** : L'objective du calcul rapide est seulement minimiser le temps moyen de réponse d'un ensemble de transactions, tandis que l'objective des bases de données temps réel est de rencontrer les échéances de chaque transaction en particulier et mettre à jour la base de données à des périodes constantes.
- **Une base de données temps réel doit résider entièrement en mémoire centrale** : la raison principale pour mettre les données directement dans la mémoire centrale est d'éviter les délais introduits par l'accès disque. Ceci est fait habituellement par des algorithmes d'ordonnancement d'accès disque comme FIFO (First-In-First-Out), SSTF(Shortest-Seek-Time-First) ou SCAN. D'un l'autre côté, on a des transactions qui consistent en une séquence d'opérations de lecture, puis à des traitements des données collectées et leur enregistrement ensuite dans la base de données. Comme chacune de ces transactions a une échéance et un niveau d'importance, qui ne sont pas pris en compte par les algorithmes d'ordonnancement d'accès disque et dès lors, les échéances peuvent être quand même compromises.
- **Une base de données temporelle est une base de données temps réel** : bien que ces deux types de bases de données utilisent la notion du temps, les aspects du temps ne sont pas les mêmes. Les bases de données temporelles se basent sur les notions de temps associé aux informations contenues dans la base de données, tandis que les bases de données temps réel mettent l'accent sur les contraintes temporelles associées aux opérations appliquées à cette base de données
- **Il est impossible de garantir en même temps le respect des contraintes logiques et temporelles dans un SGBD temps réel** : la complexité de ce concept vient du fait que dans les bases de données temps réel, il y a un nombre de sources imprévisibles, comme la dépendance d'exécution de transactions de la quantité d'informations qu'elle utilise, les conflits de données et de ressources, les délais I/O, l'abandon de transactions dû aux redémarrages. On a déjà mentionné les recours possibles pour des problèmes I/O, comme l'utilisation de la mémoire centrale pour sauvegarder les données. Dû à des problèmes de conflits, le pire temps de réponse d'une transaction peut augmenter et mener, dans le pire cas, à un nombre de redémarrages non borné et dès lors compro-

mettre la rencontre des échéances d'autres tâches. La solution à ce problème réside directement dans la notion de systèmes temps réel, où l'ensemble des transactions est a priori bien connu à l'avance et dès lors peut être pré-analysé et borné dans le temps.

Chapitre 5

RTSQL

RTSQL est une extension du langage standard de requêtes SQL, cette extension a été conçue pour la gestion de base de données temps réel et spécifie des contraintes de cohérence temporelle sur les données, des contraintes sur le temps d'exécution, des limites sur l'utilisation de ressources système pour la prédictibilité et enfin des structures de transactions flexibles qui relâchent les propriétés traditionnelles des transactions ACID afin de respecter les exigences temps réel.

5.1. RTSORAC

RTSORAC (Real Time Semantic Object Relationship ans Constraints) est un modèle développé pour les bases de données temps réel, ce modèle est légèrement basé sur le modèle relationnel (ER Entity Relationship). RTSORAC offre des capacités dérivées du modèle ER comme la modélisation des données ainsi que de leurs relations mais aussi de nouvelles capacités afin de supporter les exigences temps réel. Pour plus de détails sur ce modèle, se référer au chapitre 3 de [Pri95].

5.2. Extensions RTSQL

RTSQL étend SQL2 sur trois champs :
- Les contraintes sont étendues pour les données et leur exécutions afin de prendre en compte les aspects temporels.
- Une nouvelle construction appelée *directive* est ajoutée.
- La structure des transactions est flexible et relâche les exigences ACID, cette structure n'est pas discutée dans ce document, toute personne intéressée peut se référer à [Pri95].

5.2.1 Les contraintes

Dans SQL2, les contraintes sont des mécanismes spécifiant seulement les exigences de cohérence logique des données, RTSQL étend cette notion de deux manières. Dans un premier temps, les exi-

gences de cohérence temporelle doivent être spécifiées et dans un second, les contraintes sur les temps d'exécution des transactions et déclarations doivent être spécifiées.

Cohérence temporelle

Les données dans les bases de données temporelles ne doivent pas seulement avoir une cohérence logique mais ils doivent lêtre au niveau temporel également. Cela est dû au fait que les applications critiques doivent avoir des données reflétant létat courant de lenvironnement. Ces données ne sont collectées quà des intervalles discrets et ne sont, de ce fait, quune approximation de la réalité, ainsi plus le temps passe, plus cette approximation est de moins en moins précise jusquà ce que celle-ci deviennent complètement non représentative de lenvironnement et, à ce moment là, les données sont dites pas cohérentes temporellement. Deux types de cohérence temporelle se distinguent, la cohérence temporelle absolue et la cohérence temporelle relative.

La cohérence temporelle absolue consiste à ce que l'âge d'une donnée ne dépasse pas son échéance, par exemple une donnée de vitesse devant être mise à jour toutes les 5 secondes est dites cohérente temporellement dans l'absolu tant que l'âge de cette donnée ne dépasse pas 5 secondes. Cette exigence nécessite d'indiquer l'âge maximal acceptable pour une donnée et de sauvegarder la date à laquelle la donnée a été déterminée. Afin de déterminer la date de création d'une donnée, il faut lui spécifier un timestamp :

<data timestamp clause> : :=[WITH TIMESTAMP <datetime type>]

Il est à noter que le type datetime provient de SQL2. Afin d'accéder à ce timestamp, il existe une fonction TIMESTAMP. Il existe aussi une fonction dans SQL2 qui retourne l'instant présent qui est CURRENT_TIMESTAMP. SQL2 offre une spécification pour la syntaxe des contraintes qui peut être utilisé pour spécifier les contraintes de cohérence temporelles à l'aide des deux fonctions présentées ci-dessus. Voici un exemple de contrainte définissant la contrainte citée précédemment sur la vitesse :

```
CONSTRAINT speed_avi
   CHECK (CURRENT_TIMESTAMP -TIMESTAMP(speed)) DAY to SECOND
      < INTERVAL '5' SECOND
```

Cette clause CHECK, comme en SQL2, est une expression booléenne à la différence près que dans SQL2, il n'est pas permis qu'une clause CHECK contienne une fonction qui retourne la date. Il est évident que dans RTSQL, cette restriction doit être relâchée.

Un ensemble de données est relativement cohérent temporellement lorsque cet ensemble est utilisé pour un calcul, chaque donnée de cet ensemble doit de ce fait représenter une même image de

l'environnement, leurs âges doivent donc être dans un intervalle spécifié. Par exemple un système de suivi calculant la position d'un contact doit utiliser des données de vitesse et d'orientation avec des âges relativement proches, de l'ordre de 2 secondes. Voici l'expression de cette contrainte :

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

Il peut aussi être indiqué à quel moment une contrainte est valide et cela lorsque la contrainte est suivie d'une clause AFTER ou BEFORE appliquée à une expression de type datetime. Dans l'exemple ci-dessus, il peut être indiqué que la clause est valide seulement après une date CONTACT_MADE de type datetime de la manière suivante :

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

La violation de contrainte temporelle doit être détectée et récupérée. La détection peut être effectuée selon deux approches : l'approche passive et l'approche active. L'approche passive consiste à attendre jusqu'à ce que la valeur soit lue et vérifier la contrainte à ce moment. L'approche active consiste vérifier la contrainte périodiquement. Dans RTSQL, c'est l'approche passive qui est utilisée. Comme dans SQL2, la récupération d'erreur passe par une exception, par exemple, par une tentative de mise à jour de la donnée ou encore par une alerte seulement à l'utilisateur.

Contraintes de temps sur les exécutions

Du aux contraintes de cohérence temporelle de données, une nouvelle contrainte est créée, la contrainte de temps sur l'exécution. En effet, dû aux contraintes d'âge des données, il faut que les opérations de mise à jour par exemple, s'effectuent dans des temps bornés inférieurs à l'âge. Les systèmes de base de données traditionnels n'offrent pas de mécanismes pour placer des contraintes de temps sur les opérations ou transactions, RTSQL doit permettre de tels contraintes et pour cela spécifie des clauses de contraintes de temps qui peuvent être placée sur une déclaration ou sur un bloc de déclarations :

```
<timing constraint clause> : :=
   [START BEFORE <datetime value expression>]
   [START AFTER <datetime value expression>]
   [COMPLETE BEFORE <datetime value expression>]
```

[COMPLETE AFTER <datetime value expression>]
[PERIOD <interval value expression>
[START AT <time expression>]
[UNTIL <boolean expression>]]

Ces différentes clauses permettent d'indiquer les instants auxquels les déclarations doivent commencer ou terminer leurs exécutions. Dans RTSQL, toutes les fonctions ayant comme valeur un datetime et étant incluses dans une expression, doivent être évaluées au même instant, avant l'exécution de l'expression.
Exemple :

X :BEGIN
SELECT price FROM stocks WHERE name="Acme"
COMPLETE BEFORE CURRENT_TIMESTAMP + INTERVAL '30' SECOND ;
– other computations
END X COMPLETE BEFORE CURRENT_TIMESTAMP + INTERVAL '1' MINUTE ;

Dans cet exemple, la valeur de CURRENT_TIMESTAMP est la même pour les deux occurrences et l'exemple indique que l'exécution de la clause SELECT doit être terminée en 30 seconde et le reste des calculs en 1 minute.

La détection de violation de contrainte de temps peut être faite à l'aide de timers définis pour chaque contrainte de temps. Pour chaque contrainte de temps non respectée, il est possible de spécifier une action à effectuer, le temps d'exécution de cette action doit aussi respecter les contraintes de temps du bloc dans lequel la contrainte de temps a été violée. Plus de détails au chapitre 4.1 de [Pri95]

5.2.2 Les directives

Afin de respecter les exigences de prédictibilité, RTSQL introduit un concept appelé *directive*. Une directive est une information supplémentaire donnée au système permettant de mieux respecter les contraintes et la prédictibilité. Un exemple serait une contrainte sur l'acquisition d'une donnée de l'ordre de quelques millisecondes, une directive pour que cette contrainte respecte son échéance est de demander à ce que cette donnée soit toujours sauvegardée en mémoire centrale du fait que cet accès est plus rapide et plus prédictible qu'un accès sur le disque. Il existe un certain nombre de directives proposées dans RTSQL, celles-ci sont présentées dans les sous-sections qui suivent.

Stockage de données

L'une des caractéristiques ayant une influence sur le temps d'exécution de transaction est l'endroit où sont stockées les données accédées. Dans ce but, des directives de stockage sont utilisées et permettent au programmeur de préciser où et comment une table doit être stockée. Les clauses suivantes sont utilisées lors de la création d'une table et permettent de définir les exigences de stockage :

<storage clause> : :=[STORE IN <storage type> [AT <location>]]
<table size clause> : :=[SIZE UPPER LIMIT <integer>]

La première clause permet de spécifier l'endroit de stockage comme par exemple STORE IN main_memory, il peut aussi être précisé l'adresse de stockage en mémoire grâce à la clause AT <location>. La seconde clause permet d'avoir une borne lors d'une recherche dans la table et cela en limitant la taille de cette table.

Niveau d'importance relatif

Une directive peut permettre de donner des importances relatives à des actions, cela permet, par la suite, à des algorithme d'ordonnancement d'utiliser cette information afin de déterminer les priorités des tâches. La sémantique du niveau d'importance peut varier d'un système à un autre, pour cette raison, la portabilité de cette directive est limitée.

<importance clause> : :=IMPORTANCE LEVEL <importance level>

Exécutions asynchrone

Dans des systèmes temps réel, il est utile de connaitre les actions qui peuvent être faites de manière asynchrone (en parallèle) surtout lorsque plusieurs processeurs sont disponibles. Ce concept a été proposé dans SQL3 et la déclaration de telles exécutions est faite par la clause suivante :

[ASYNC (< async statement identifier >)]

Si cette déclaration n'est pas spécifié, cela veut dire que l'exécution est synchrone. Il est aussi possible de vérifier la terminaison de traitements asynchrones comme suit :

< test completion statement > : :=
    TEST | WAIT
    ALL | ANY | < async statement identifier list > COMPLETION

Cette expression permet donc soit de tester (TEST) la terminaison de traitements asynchrones, si ce n'est pas le cas, une exception est traitée, soit d'attendre (WAIT) cette terminaison. Les traitements asynchrones considérés sont soit tous les traitements d'une transaction (ALL), soit n'importe lequel (ANY), soit un ensemble listé.

Pire temps d'exécution (WCET)

RTSQL permet de préciser le pire temps d'exécution d'un traitement, ce temps est généralement obtenu par le concepteur du système par des analyses[PM94]. Cette valeur est utile pour étudier la faisabilité d'un système de tâches [LL73] ou encore pour déterminer si une action pourra ce terminer avant son échéance et de ce fait prédire une future violation. Le WCET est spécifié de la manière suivante :

[WCET (<time literal>)]

Chapitre 6

Conclusion

Nous avons présenté dans ce document les raisons d'être des systèmes de base de données temps réels, leurs applications et leurs contraintes. Nous avons ensuite présenté RTSQL et ces extensions par rapport à SQL2. A l'heure actuelle, il n'existe aucun système de base de données commerciale permettant de satisfaire toutes les exigences des systèmes temps réel, certains se disent temps réel comme extremeDB, mais les spécifications de ceux-ci insistent plus sur la rapidité que sur des temps de réponse bornés. RTSQL était une tentative de standardisation de langage de requêtes pour les bases de données temps réelles mais étant donné le manque de documentation récente, nous pensant que ce projet a été abandonné.

Bibliographie

[IDOU] Vers une méthode de conception des bases de données temps réel - Nizar Idouli, Claude Duvallet, Bruno Sadeg, Faiez Gargouri

[RTDS] Real-Time Database Systems - Architecture and Techniques (Springer), Kam-Yiu Lam, Tei-Wei Kuo, 2002

[CONT] Contributions à la gestion des transactions dans les SGBD temps réel - Bruno Sadeg

[DUVA] Les SGBD temps réel - Claude Duvallet, Zoubir Mammeri, Bruno Sadeg

[Pri95] RTSQL :Extending the SQL2 standard to support real-time databases-JANET JAMIE PRICHARD,UNIVERSITY OF RHODE ISLAND,1995

[LL73] L. Liu, W. Layland. Scheduling algorithms for multi-programming in a hard-real-time environment. Journal of the ACM, 20(1) :46 61, 1973.

[PM94] W. Pugh, T. Marlow, Proceedings of the ACM SIGPLAN workshop on language, compiler and tool support for real-time systems. Juin 1994.