Notes de synthèse :
Petite histoire de la gestion des transactions

Alain Bernardeau-Moreau

Janvier 1997

Rapport CERMICS N^o 97-90

Notes de synthèse : Petite histoire de la gestion des transactions

Alain Bernardeau-Moreau
Équipe « Base de données » - CERMICS
INRIA, 2004 route des Lucioles 06902 Sophia Antipolis
http://www.inria.fr/cermics/dbteam/

Résumé

Ce document présente une petite synthèse sur la gestion de transaction dans les systèmes de gestion de bases de données (SGBD). Dans un premier temps sont décrites les techniques permettant le contrôle de l'exécution concurrente des transactions, en particulier les techniques de verrouillage, d'estampillage ou encore la gestion de versions. Les modèles transactionnels spécifiques aux SGBD à objets sont ensuite présentés. Sont finalement évoqués les problèmes transactionnels posés par la distribution et l'exploitation de SGBD à travers l'Internet.

Abstract

This document is a synthetic article on the transaction management in database management systems (DBMS). In the first part, the techniques that allow the transaction concurrency are described, in particular locking, stamping, and version management. Then, the transactional models for object DBMS are presented. Finally, the transactional problems due to distribution or to the use of DBMS through Internet are itemized.

Table des matières

1 Contrôle de l'exécution concurrente de transactions

Une base de données est un ensemble de données sur lesquelles des opérations de lecture (R pour Read) et d'écriture (W pour Write) peuvent être effectuées. Les opérations de lecture ne modifient pas les données contrairement aux opérations d'écriture. Une transaction est une séquence d'opérations qui fait passer une base de données d'un état cohérent à un autre.

La terminologie utilisée est conforme au modèle standard de sérialisabilité qui repose sur la notion de conflit [GR93]. On dit que deux opérations sur la même donnée entrent en conflit si elles appartiennent à des transactions différentes et si l'une au moins est une écriture ; donc nous avons les conflits (R/W) et (W/W). Une histoire est constituée de plusieurs transactions qui forment dans leur ensemble une séquence d'opérations de lecture et d'écriture. Une histoire sérialisée est une séquence d'opérations appartenant à des transactions consécutives. Aucun conflit ne peut apparaître dans une histoire sérialisée. Une histoire d'opérations appartenant à plusieurs transactions est dite sérialisable si les résultats qu'elle produit sont équivalents à ceux produit par une histoire sérialisée. Ces équivalences assurent que l'exécution d'une histoire sérialisable est correcte dans le sens où elle laisse la base de données dans un état cohérent. Les mécanismes de contrôle de concurrence qui assurent la sérialisabilité sont des algorithmes qui gèrent l'ordre des opérations de telle façon que seules les histoires sérialisables sont autorisées.

1.1 Techniques de verrouillage

Cette technique de contrôle de concurrence assure que, si des opérations entrent en conflit lors du partage des mêmes données, ces données ne seront accédées que par une seule opération en même temps. Ceci est obtenu par l'association de verrous à certaines parties (ou granules) de la base de données. Nous désignerons un "granule" par le terme "donnée" lorsque la notion de granularité de la donnée n'a pas d'importance. Un verrou est positionné par une transaction sur une donnée avant que celle-ci ne soit accédée ; le verrou est relâché lorsque la donnée n'est plus utilisée. Naturellement, une donnée verrouillée par une opération ne peut être accédée par une opération concurrente, qui est alors mise en attente.

1.1.1 Le verrouillage à deux phases

Ce mécanisme de verrouillage peut être utilisé par un gestionnaire de verrous pour assurer la sérialisabilité des opérations [HG87]. Considérons les notations suivantes pour clarifier notre propos : *rli[x]* (respectivement *wli[x]*) indique que la transaction Ti a obtenu un verrou en lecture sur la donnée x (respectivement, un verrou en écriture). Nous utiliserons les lettres p et q pour représenter une opération de n'importe quel type (une lecture ou une écriture). Ainsi, *pi[x]* est une opération de la transaction Ti sur la donnée x. Nous dirons que les verrous *pli[x]* et *qlj[y]* rentrent en conflit si x=y, i≠j et les opérations p et q rentrent en conflit.

Voici les règles que suit le gestionnaire de verrous pour gérer un verrouillage à deux phases :

1. Quand il reçoit une opération *pi[x]* du gestionnaire de transactions, le gestionnaire de verrous teste si *pli[x]* rentrent en conflit avec quelque *qlj[x]* déjà positionné. Si tel est le cas, alors *pi[x]* est retardé, mettant en attente la transaction Ti qui ne peut recevoir le verrou dont elle a besoin. Sinon, le gestionnaire de verrous positionne *pli[x]* et envoie *pi[x]* au gestionnaire de données.

2. Une fois que le gestionnaire de verrous a positionné un verrou pour Ti, disons *pli[x]*, il ne doit pas relâcher le verrou avant que le gestionnaire de données ait confirmé le traitement de l'opération *pi[x]*.

3. Une fois que le gestionnaire de verrous a relâché un verrou pour une transaction, celle-ci ne doit plus obtenir aucun verrou sur aucune donnée.

La règle (3), appelée règle des deux phases est à l'origine du nom, verrouillage à deux phases. Ainsi, chaque transaction peut être divisée en deux phases : la phase de développement (growing phase) au cours de laquelle la transaction acquiert des verrous et une phase de réduction (shrinking phase) qui correspond au relâchement des verrous.

Généralement, toutes les applications gérant le verrouillage à deux phases implantent cette méthode en s'assurant que tous les verrous sont relâchés lorsque la transaction se termine. Pratiquement, cela signifie que les verrous en lecture peuvent être relâchés lorsque la transaction se termine (i.e. lorsque le gestionnaire de verrous reçoit le message de validation ou d'annulation de la transaction), tandis que les verrous en écriture doivent être maintenus jusqu'à ce que la transaction soit validée ou annulée (i.e. après que le gestionnaire de données ait traité la validation ou l'annulation).

1.1.2 Le verrouillage hiérarchique

Cette technique de verrouillage s'appuie sur une organisation hiérarchique des structures de la base de données : par exemple, on considère qu'une **base de données** est composée de sites ; chaque **site** organise ses informations dans des **fichiers** qui regroupent plusieurs **enregistrements**. Dans les bases de données relationnelles, les fichiers correspondent aux relations et les enregistrements sont des n-uplets. Dans les bases de données objet, les fichiers correspondent aux classes et les enregistrements aux objets. La granularité d'une donnée correspond à la taille relative de cette donnée. Ainsi, la granularité d'un fichier est plus élevée que celle d'un enregistrement.

Le verrouillage hiérarchique consiste à tirer partie des relations hiérarchiques entre des verrous positionnés sur des granules différents [HG87]. On représente ces relations par un graphe de types verrouillables ("lock type graph", cf. figure 1). Un ensemble de données structurées conformément à ce graphe est appelé un graphe d'instances verrouillables. Pour simplifier le modèle,

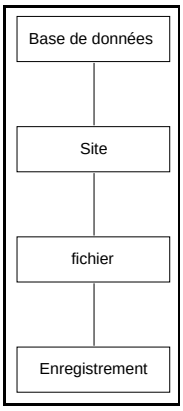

FIG. 1 – *hiérarchie de types verrouillables*

nous supposerons que ce graphe est un arbre (cf. figure 2). L'idée principale du verrouillage hiérarchique est la suivante : la pose d'un verrou sur une donnée x verrouille explicitement x et verrouille implicitement tous les descendants de x qui sont de granularité plus fine. Il faut aussi assurer la propagation ascendante de la pose de verrous. Par exemple, avant de verrouiller x, le gestionnaire de verrous doit s'assurer qu'il n'y a pas de verrous sur les ancêtres de x qui verrouillent implicitement x dans un mode conflictuel. A cet effet, à chaque type de verrou (*rl* et *wl*), on associe un verrou d'intention (respectivement *irl* et *iwl*). Ces verrous d'intentions sont posés sur tous les ancêtres de x en partant du sommet de l'arbre (i.e. base, site, fichier...). Il est utile également, d'introduire un cinquième type de verrou : *riwl*. Celui-ci est utilisé lorsqu'une transaction veut lire tous les enregistrements d'un fichier mais ne veut en modifier que quelques uns. Ce type de verrou est équivalent à la pose simultanée des deux verrous *rl* et *iwl*. On définit une table de compatibilités entre deux opérations concurrentes ; celles-ci entrent en conflit lorsqu'elles apparaissent incompatibles dans la table. Voici la table de compatibilité dans le cas du verrouillage hiérarchique :

compatible ?	*rl*	*wl*	*irl*	*iwl*	*riwl*
rl	y	n	y	n	n
wl	n	n	n	n	n
irl	y	n	y	y	y
iwl	n	n	y	y	n
riwl	n	n	y	n	n

Étant donné un arbre d'instances verrouillables A, voici le protocole que suit le gestionnaire de verrous pour implanter le verrouillage hiérarchique :

1. Si x n'est pas la racine de A, alors, pour obtenir un verrou *rli[x]* ou *irli[x]* sur x, Ti doit

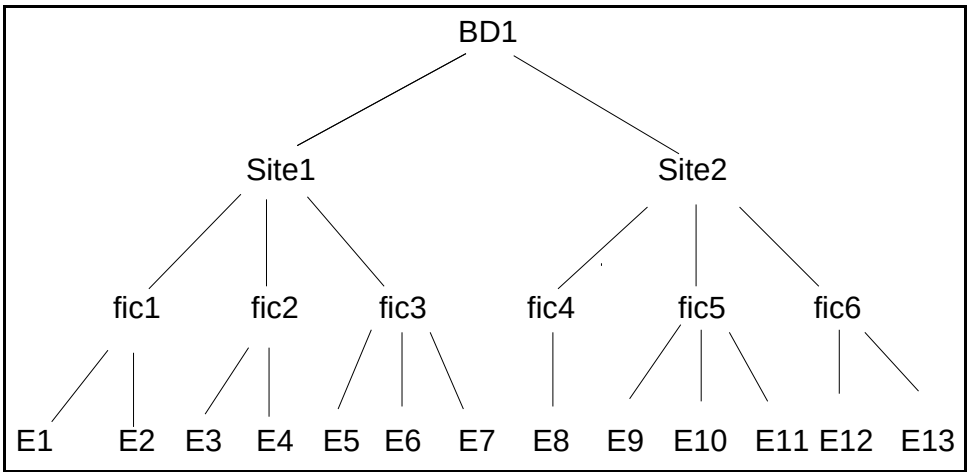

FIG. 2 – *arbre d'instances verrouillables*

avoir un verrou *ir* ou *iw* sur le parent de x. Si aucun verrou n'est présent, on pose *ir* sur le parent de x.

2. Si x n'est pas la racine de A, alors, pour obtenir un verrou *wli[x]* ou *iwli[x]* sur x, Ti doit avoir un verrou *iw* sur le parent de x. Si aucun verrou n'est présent, on pose *iw* sur le parent de x.

3. Pour lire (respectivement écrire) x, Ti doit posséder un verrou de lecture ou d'écriture (respectivement d'écriture) sur un des ancêtres de x. Un verrou sur x lui-même est un verrou explicite ; un verrou sur un des ancêtres de x définit un verrou implicite sur x.

4. Une transaction ne peut relâcher un verrou implicite sur x si elle possède encore un verrou sur un fils de x.

Différencier la granularité des données est important pour les performances de la gestion de verrous. Utiliser des verrous grossiers, c'est-à-dire des verrous sur des données de forte granularité, introduit un faible surcoût de travail puisqu'il y a peu de verrous à gérer. Par contre, cela réduit la concurrence du fait que des opérations ont plus de risques de rentrer en conflit. D'un autre côté, utiliser des verrous fins, c'est-à-dire des verrous sur des données de fine granularité, améliore la concurrence car cela permet aux transactions de ne verrouiller que les données qu'elles utilisent.

L'avantage de ce modèle, c'est que chaque transaction peut individuellement choisir la granularité des données qu'elle souhaite verrouiller.La pose de verrous grossiers permet de verrouiller une grande quantité de données à faible coût. Toutefois, contrairement à la pose de verrous

fins, cela limite la concurrence des transactions. En particulier, certaines données peuvent être verrouillées par une transaction sans que celle-ci ne les utilise, ce qui peut bloquer sans raisons d'autres transactions. La pose de verrous fins permet plus de concurrence mais induit un coût de gestion des verrous d'autant plus important que le nombre de données accédées est élevé. Dans le cadre du verrouillage hiérarchique, il faut trouver un compromis entre la finesse des verrous et le coût induit par la gestion de ceux-ci pour atteindre un optimum de disponibilité des données de la base.

1.2 Techniques d'estampillage

Cette technique consiste à associer à chaque transaction une unique estampille notée *ts[Ti]*. Le gestionnaire de transactions associe cette estampille à chacune des opérations de la transaction Ti. On peut ainsi parler de l'estampille d'une opération. Le gestionnaire d'estampilles ordonne les opérations en conflit en utilisant leur estampille respective. Plus précisément, il respecte la règle suivante appelée règle d'ordonnancement des estampilles ("Timestamp Ordering Rule") :

- Si pi[x] et qj[x] sont des opérations conflictuelles alors le gestionnaire de données traitent pi[x] avant qj[x] si et seulement si $ts[Ti] < ts[Tj]$.

En respectant cette règle, on s'assure que deux opérations qui rentrent en conflit seront exécutées dans l'ordre de leurs estampilles. Ainsi, l'exécution selon l'ordre des estampilles est équivalente à une exécution en série dans laquelle les transactions apparaissent dans l'ordre de leurs estampilles. Nous ne développerons pas ici les moyens permettant de respecter la règle d'ordonnancement des estampilles du fait que cette technique n'est pas couramment utilisée.

1.3 Gestion de la concurrence utilisant des versions de données

1.3.1 Présentation générale du modèle

Dans ce modèle de gestion de la concurrence, chaque écriture sur une donnée x entraîne la création d'une nouvelle copie (ou version) de x. Le gestionnaire de données qui gère x doit conserver une liste des versions de x. Pour chaque lecture de x, le gestionnaire de verrous doit décider quand envoyer l'opération de lecture et doit indiquer au gestionnaire de données sur quelle version de x effectuer l'opération [HG87].

L'avantage de cette approche est que cela permet au gestionnaire de concurrence d'éviter de mettre en attente des opérations qui arrivent trop tard, par exemple une opération de lecture sur une donnée qui a déjà été modifiée par une autre transaction. Le fait de conserver plusieurs versions d'une même donnée peut également servir dans l'algorithme de recouvrement des fautes. En effet, les "images avant" d'une donnée correspondent exactement à la liste des anciennes versions de cette donnée. Par contre, conserver plusieurs versions d'une même donnée est coûteux

en espace de stockage. C'est pourquoi, les versions doivent être effacées périodiquement ce qui constitue un coût en terme de performance dans l'algorithme de gestion de la concurrence.

On suppose que, si une transaction est annulée, alors toutes les versions qu'elle a produites sont effacées. Ainsi, le terme version correspond à l'état d'une donnée produit par une transaction en cours ou bien validée. On peut considérer qu'une version d'une donnée est la propriété exclusive d'une transaction ; une autre approche consiste à permettre à des transactions d'accéder en lecture à la version la plus récente d'une donnée. Dans ce cas, si une transaction Ti provoque une lecture sur une version produite par une transaction active Tj, alors la validation de Ti doit être retardée jusqu'à ce que Tj ait été validée. Si Tj est annulée, alors la version produite est effacée et la transaction Ti doit être annulée elle aussi. Ceci peut entrainer des annulations en cascade peu souhaitables.

L'existence de plusieurs versions des mêmes données est seulement visible par le gestionnaire de concurrence et le gestionnaire de données. Au niveau de l'utilisateur et de chaque transaction, tout se passe comme s'il n'y avait qu'une seule version de chaque donnée.

1.3.2 Verrouillage à deux phases utilisant les versions multiples de données

Dans le verrouillage à deux phases classique, un verrou en écriture sur une donnée x empêche à d'autres transactions d'obtenir un verrou en lecture sur x. On peut éviter ce conflit en utilisant deux versions de x. Quand une transaction Ti accède x en écriture, une version xi de x est créée. Elle obtient alors un verrou sur x, mettant l'accès en lecture sur xi et l'accès en écriture sur une autre version de x. Toutefois, d'autres transactions sont autorisées à accéder en lecture sur l'ancienne version de x.

Ce modèle utilise le verrouillage à deux phases pour gérer deux opérations d'écriture entrant en conflit et utilise la sélection de versions pour gérer les conflits de lecture-écriture. Trois types de verrous sont utilisés : le verrou en lecture, le verrou en écriture et le verrou de certification. Ces verrous sont soumis à la table de compatibilités suivante :

compatible ?	Read	Write	Certify
Read	y	y	n
Write	y	n	n
Certify	n	n	n

Le gestionnaire de verrous positionne les verrous de lecture et d'écriture comme d'habitude en fonction des opérations à traiter. Quand une transaction est sur le point de valider, elle tente de convertir tous ses verrous d'écriture en verrous de certification.

Quand le gestionnaire de verrous reçoit une opération d'écriture *wi[x]*, il essaie de positionner un verrou *wli[x]*. Conformément à la table de compatibilité, l'opération est retardée si une autre

transaction a un verrou d'écriture ou de certification sur x. Sinon, il positionne *wli[x]*, convertit *wi[x]* en *wi[xi]* et transmet l'opération au gestionnaire de données.

Quand le gestionnaire de verrous reçoit une opération de lecture *ri[x]*, il essaie de positionner un verrou *rli[x]*. Les verrous de lecture n'entrent en conflit qu'avec les verrous de certification. Si Ti possède déjà un verrou *wli[x]* et a modifié x en xi, alors l'opération *ri[x]* est transformée en *ri[xi]* et envoyée au gestionnaire de données. Dans les autres cas, l'opération est conditionné par l'obtention du verrou en lecture ; une fois le verrou de lecture acquis, l'opération *ri[x]* (où x est la donnée x stockée dans la base) est transmise au gestionnaire de données. Seules les versions validées des données peuvent être lues (sauf au sein d'une même transaction qui veut relire une donnée qu'elle a elle-même modifiée) ; ceci permet d'éviter les annulations en cascade des transactions.

La table de compatibilité assure qu'une transaction qui a faite au moins une modification de la base sur une donnée x ne peut être validée que lorsqu'aucune autre transaction ne lit ni n'écrit cette même donnée x.

L'avantage de ce modèle sur le modèle de verrouillage à deux phases classique est qu'il retarde les opérations de lecture moins "longtemps" ; en effet, le temps de validation d'une transaction est généralement plus court que le temps d'exécution de cette même transaction (notamment lors de transactions longues). Or dans ce modèle, les opérations de lecture sont retardées pendant les phases de certification et non pendant les phases de modification. Ceci améliore la disponibilité des données en lecture.

2 Les modèles transactionnels dans les SGBDO

La technologie des SGBDO est sensée permettre de concevoir des applications complexes, ce qui a des répercussions sur les modèles transactionnels nécessaires. Un réel besoin de modèles transactionnels plus généraux et plus puissants se fait sentir. Les extensions les plus importantes qui ont été identifiées dans le domaine des transactions sont les suivantes [Özs94] :

- Les systèmes de gestion de transactions doivent prendre en compte des opérations plus abstraites que des lectures ou des écritures de données. Il est possible d'améliorer la concurrence des transactions en utilisant la sémantiques des objets et de leurs opérations abstraites.
- Le partage des données doit être plus coopératif que compétitif : par exemple, deux utilisateurs peuvent travailler simultanément sur des parties différentes du même document.
- Les accès transactionnels à des objets composites ou complexes doivent être synchronisés : permettre le partage d'accès à ces objets suppose qu'on gère le partage d'accès aux objets qui les composent, et ainsi de suite.

– Les applications conçues autour des SGBDO peuvent entreprendre de longues activités nécessitant la gestion de transactions longues. Dès lors, le modèle transactionnel doit permettre le partage des résultats intermédiaires. De plus, l'échec de tâches essentielles entraînant l'échec de l'activité toute entière est inacceptable pour une longue transaction. Il faut pouvoir évaluer l'importance des tâches dans l'accomplissement de la transaction et pouvoir proposer des alternatives lorsque les tâches initiales échouent.

Actuellement, les SGBDO intègrent des techniques de verrouillage éprouvées, héritées des SGBD relationnels. EXODUS et OBJECTSTORE emploient des techniques de verrouillage sur un granule de page. VERSANT et POET effectuent des verrouillages sur un granule d'objet. O2, quant à lui, utilise la technique du verrouillage hiérarchique. Enfin, ORION reprend la technique du verrouillage hiérarchique en l'adaptant aux hiérarchies de classes et d'objets complexes.

Plusieurs études de modèles transactionnels ont été proposés dans la littérature, mais n'ont pas encore été intégrés dans les SGBDO du commerce. Beaucoup d'efforts ont été consacrés à l'exploitation de la sémantique des méthodes des objets. Ceci tend à améliorer la gestion de la concurrence par rapport à celle obtenue en ne considérant que les modes d'accès en lecture et écriture pour déterminer les conflits. L'idée est d'utiliser les propriétés de commutativité qui peuvent exister entre les méthodes constituant l'interface des objets. Le gestionnaire de verrous peut autoriser deux transactions à exécuter sur un même objet des méthodes incompatibles en termes d'accès en lecture et écriture si ces méthodes sont commutatives.

3 Les systèmes transactionnels distribués et l'Internet

La rapide évolution des réseaux de réseaux suscite un grand intérêt pour l'adaptation des systèmes distribués existants à ces vastes réseaux, comme Internet. Ainsi, des études sont menées pour étudier les performances du traitement distribué de transactions sur l'Internet [ZG94] ; et notamment l'impact de l'Internet sur les algorithmes de gestion de la concurrence et sur les algorithmes de maintien de l'atomicité. Les conclusions auxquelles ces études ont conduit sont les suivantes :

3.1 Amélioration du module de communication

Le gestionnaire de communication est un des éléments les plus importants dans les systèmes de traitement distribué des transactions. Les gestionnaires de communication fournis actuellement dans les systèmes d'exploitation ne sont pas adaptés pour le traitement des transactions en ligne. Actuellement, la plupart des systèmes transactionnels distribués utilisent le protocole UDP/IP comme protocole de communication. Ce protocole ne garantit pas l'acheminement des messages. Sur un réseau local, le taux d'erreur est suffisamment faible pour répondre aux besoins en performance et en temps réel des applications. Cependant, l'exécution d'une transaction peut

générer un trafic important de données sur le réseau (une requête sur une grande quantité d'objets, par exemple). Le taux d'erreur sur certaines parties du réseau peut être si élevé que même la retransmission des messages avec UDP/IP s'avère inadéquat. Une méthode radicale serait d'utiliser le protocole TCP/IP qui garantit l'acheminement des messages au détriment du temps d'établissement de la connexion. Un compromis consiste à définir un protocole permettant de paramétrer le degré de fiabilité des communications au niveau de chaque application.

3.2 Adaptation des algorithmes et protocoles utilisés dans le traitement distribué des transactions

Voici quelques conjectures sur l'impact d'un réseau comme Internet sur la performance des algorithmes et protocoles utilisés dans les systèmes transactionnels distribués :

- *impact sur la gestion de la concurrence* : Les temps de réponse lors des communications sur un réseau comme l'Internet passent des quelques millisecondes à quelques centaines de millisecondes. Ceci implique qu'une transaction reste plus longtemps dans le système, ce qui augmente la probabilité de conflit. Ainsi, une gestion de la concurrence par des techniques classiques de verrouillage bloquerait les transactions plus longtemps, augmenterait les risques d'interblocage (ou "deadlock") ce qui conduirait à un taux d'annulation de transaction élevé. Une solution consiste à relâcher les contraintes exigées par une gestion stricte de la concurrence. Ainsi, une gestion de la concurrence basée sur la notion "d'epsilon sérialisabilité" autorise un certain degré d'incohérence des données de la base pendant une courte durée "epsilon" [YP94].

- *impact sur le maintien de l'atomicité* : Le protocole de validation à deux phases n'est pas résistant à la panne d'un site. Le protocole à trois phase, quant à lui, nécessite une phase requête-réponse supplémentaire pour assurer l'atomicité des transactions. Sur un réseau local, la validation à trois phases fournit une plus grande disponibilité des données pour un temps de réponse légèrement plus élevé. Sur un réseau comme Internet, la validation peut représenter jusqu'à 80% du temps global d'exécution d'une transaction, comparé aux 30% sur un réseau local. Dès lors, la validation à trois phases doit être utilisée avec précaution, seulement lorsqu'aucun compromis ne peut être fait sur la disponibilité des données. Cela encourage à préférer des protocoles de maintien de l'atomicité ayant été optimisés pour diminuer le nombre de messages et par la même, le nombre de communications.

Références

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufman, 1993.

[HG87] P.A. Bernstein V. Hadzilacos and N. Goodman. *Concurrency control and recovery in database systems.* Addison-Wesley, 1987.

[YP94] Kun-Lung Wu Philip S. Yu and Calton Pu. Divergence control algorithms for epsilon serializability. August 1994.

[ZG94] Bharat Bhargava Yongguang Zhang and Shalab Goel. A study of distrbuted transaction processing in an internetwork. Technical report, Purdue University, 1994.

[Özs94] M.T. Özsu. Transaction models and transaction management in object-oriented database management systems. In A. Biliris A. Dogac, M.T. Özsu and T. Sellis, editors, *Advances in Object-Oriented Database Systems*, pages 147–184. Springer-Verlag, 1994.

Liste des derniers rapports de recherche du CERMICS

List of previous CERMICS research reports

96-77	E. Cancès, C.Le Bris	*On the perturbation methods for some nonlinear quantum chemistry models.* 45 pages, novembre 1996, Noisy-Le-Grand
96-78	S. Depeyre	*Une méthode couplée pour la simulation d'écoulements disphasiques dispersés.* 57 pages, octobre 1996, Sophia-Antipolis
96-79	N.Glinsky-Olivier J.F.Gerbeau B.Larrouturou	*Semi-implicit Roe-type fluxes for low-mach number flows.* ...pages, octobre 1996, Sophia-Antipolis
96-80	*B. Jourdain*	*Propagation du chaos trajectorielle pour les lois de conservation scalaire.* 14 pages, décembre 1996, Noisy-Le-Grand
96-81	*B. Jourdain*	*Diffusions with a nonlinear drift coefficient and probabilistic interpretation of generalized Burger's equations.* 16 pages, décembre 1996, Noisy-Le-Grand
96-82	*D. Hirschkoff*	*Up-to context proofs for the π-calculus in the coq system.* 18 pages, janvier 1997, Noisy-Le-Grand
96-83	*E. V Abrarova,* *A. V. Karapetyan*	*Sur la stabilité et les bifurcations des mouvements stationnaires d'un corps solide dans un champ de gravitation central.* 22 pages, janvier 1997, Noisy-Le-Grand

96-84	*E. V Abrarova,*	*Sur les mouvements stationnaires en orbite d'un système de deux corps avec liaison élastique.* 20 pages, janvier 1997, Noisy-Le-Grand
97-85	J.P. Cioni	*Parallelisation of Maxwell and 3D simulations in electromagnetism using clusters of workstations.* 18 pages, janvier 1997, Sophia-Antipolis
97-86	B. Neveu G. Trombettoni	*Computational complexity of Multi-way, Dataflow constraint problems.* 13 pages, janvier 1997, Sophia-Antipolis
97-87	F. Lebastard	*Driver II (Lisp) : Manuel de référence.* ... pages, janvier 1997, Sophia-Antipolis
97-88	E. Abrarova	*Non-integrability of the restricted three-body problem and geometrical obstacles to integrability.* 12 pages, janvier 1997, Noisy-Le-Grand
97-89	O. Jautzy	*Direct : un C++ dynamique et réflexif pour les bases de données.* 12 pages, janvier 1997, Sophia-Antipolis
97-90	A. Bernardeau-Moreau	*Notes de synthèse : Petite histoire de la gestion des transactions.* 18 pages, janvier 1997, Sophia-Antipolis

Ces rapports peuvent être obtenus en s'adressant au secrétariat du CERMICS :

The reports can be asked from:

Imane HAMADE
ENPC-CERMICS
6 et 8 Avenue Blaise Pascal
Cité Descartes Champs-sur-Marne
77455-Marne-La-Vallée CEDEX 2
Tél : (33) 01 - 64-15-35-71
email: `hamade@newaphro.enpc.fr`