Le Modèle des Espaces de Travail : Une Solution Extensible pour la Distribution de Données Persistantes

Eric Abécassis*, Didier Donsez†, Liming Chen†, Pascal Faudemay‡

* APIC Systèmes,
25 rue de Stalingrad,
94742 Arcueil cedex, France
Tél : 33.1.49.69.90.90
Fax : 33.1.49.69.92.93
ea@apic.fr

† Laboratoire HEUDIASYC / UTC
Centre de Recherche de Royallieu,
BP649, 60200 Compiègne cedex , France
Tél : 33.44.23.44.87
{donsez,lchen}@hds.univ-compiegne.fr

‡ Laboratoire MASI-IBP / UPMC,
4 place Jussieu
75252 Paris cedex 05, France
Tél : 33.1.44.27.71.16
faudemay@masi.ibp.fr

Résumé

Nous décrivons dans ce papier le modèle des Espaces de Travail et son implémentation. Il est utilisée comme une solution pour la distribution de données persistantes. Ce modèle est une extension du modèle des transactions imbriquées appliquée à la réalisation d'applications distribuées; il met en correspondance les composants d'une application distribuée avec les noeuds d'une transaction imbriquée. L'élément de base de notre modèle est l'Espace de Travail (ET); ce processus générique se compose d'une vue base de données, d'une transaction principale et de plusieurs sous-transactions. Le code applicatif, exécuté par les transactions, spécialise le rôle du processus ET. Ainsi un ET peut être un serveur bases de données, un client ou un noeud intermédiaire dans une transaction imbriquée. L'abstraction fournie par la vue base de données rend l'application indépendante du graphe de processus ET; elle permet l'extensibilité du modèle depuis un unique processus accédant à une base de données privée à un système hiérarchisé de clients et de serveurs. Nous avons implanté le modèle des ETs en utilisant le multithreading et le memory-mapping. Elle propose une interface C++ semblable à celle de l'ODMG 93. Les résultats du banc d'essai OO7 sont données à la fin de ce papier.

I. Introduction

Nous assistons ces dernières années au recul du modèle centralisé et au développement des architectures distribuées. Les stations de travail, interconnectées par des réseaux, remplacent l'étoile de terminaux autour d'un ordinateur central. Le modèle actuel d'entreprise suit une architecture à trois niveaux : le serveur d'entreprise, les serveurs départementaux, et les postes de travail. Bien que cette architecture se généralise, il reste difficile d'utiliser pleinement la puissance de calcul disponible et de tirer avantage de la distribution de données. Actuellement la distribution de données s'appuie sur un mécanisme de checkin/checkout entre le serveur d'entreprise et les serveurs départementaux; les PCs exécutent principalement les interfaces utilisateurs. Cette situation complique l'accès aux données de l'entreprise et conduit à une sous utilisation des CPU disponibles.

Les principaux travaux sur les bases de données distribuées ont été réalisées dans le cadre de l'approche relationnelle. Celle-ci facilite la distribution des traitements; une requête est interprétée comme un flux de données distribué entre des CPUs ; les données sont généralement traitées par les machines qui les archivent. Cependant le couplage langage de requêtes/L3G constitue un handicap sérieux pour développer des applications générales. Les SGBDOOs résolvent ce problème en adoptant des langages généraux comme langages de manipulation de données. Toutefois, les architectures des SGBDOOs actuels ne proposent qu'une distribution simplifiée des données. Chaque serveur est utilisé comme un serveur de pages et de verrous, et aucun mécanisme n'est prévu pour distribuer les traitements. Par conséquent, même les requêtes très sélectives sont calculées par les applications clientes et toutes les données nécessaires à celles-ci sont transférées du serveur vers le client. Une telle solution génère un trafic très important au niveau du réseau; ceci peut constituer un goulot d'étranglement pour certaines applications.

Notre approche pour la distribution des données consiste à définir un modèle générique, le modèle des Espaces de Travail (ET) [Donsez94]. Ce modèle s'inspire du modèle de transactions imbriquées et le met en oeuvre sur des architectures hétérogènes de systèmes informatiques. Il met en correspondance les composants de telles architectures distribuées avec les noeuds de transactions imbriquées. La base de données est une collection de volumes contenant des objets. Nous définissons le processus d'Espace de Travail comme une vue base de données (vue BD), une transaction principale et plusieurs sous-transactions. Une vue BD se compose de connections à des volumes locaux et de connections distantes à d'autres processus ET. La transaction principale contrôle la vue BD locale.

Grâce à l'architecture générique de ce processus, tout composant d'une application distribuée est une spécialisation d'un ET. Dans l'exemple de l'architecture client-serveur utilisée par les SGBDOO, un serveur bases de données est un ET dont la vue BD est constituée uniquement de connexions à des volumes locaux et dont les sous transactions servent d'autres processus ET clients. Ces derniers constituent leur vue BD à partir de connexions vers les ET serveurs.

La définition de notre modèle a abouti à une implémentation. Il s'agit d'un SGBDOO complet dont l'architecture est très proche de celle qui est définie dans l'ODMG 93. Cette implantation est basée sur le mappage de mémoire (memory mapping) pour l'accès aux données persistantes; elle utilise le "multithreading" afin d'exécuter des transactions concurrentes. Plusieurs applications ont été développées en utilisant ce produit, dont certaines sont en cours de commercialisation.

Le reste du papier est organisé comme suit. Un état de l'art est présenté dans la section 2. Le modèle d'Espace de Travail est décrit dans la section 3. La section 4 discute les différentes possibilités dans l'utilisation du modèle. La section 5 présente quelques éléments de l'implémentation. La section 6 présente les résultats du banc d'essais OO7 [Carey93]. Les conclusions et quelques directions pour le travail futur sont données dans la section 7.

II. Etat de l'Art.

Les bases de données distribuées ont fait l'objet d'études approfondies dans le cadre de l'approche relationnelles. Des projets comme Bubba [Alexander88] ou Gamma [Dewitt90] ont pour objectif de construire des bases de données à hautes performances en utilisant des machines multiprocesseurs. D'autres projets comme Ingres [Stonebraker77] permettent de distribuer des bases de données sur un réseau de machines. Dans ces deux cas, l'objectif principal est de rendre transparent l'exécution distribuée des requêtes. Il s'agit de la philosophie relationnelle, avant tout orientée vers les utilisateurs finaux et non vers les développeurs.

Les SGBDOOs proposent un environnement de développement d'application utilisant des données persistantes. Ceux-ci sont désormais reconnus comme mieux adaptés par rapport aux bases de données relationnelles dans la réalisation d'application de type CAO, SIG, AGL…. En effet, le modèle de données objet permet de capturer la complexité de l'information gérée dans ce type d'application. D'autre part, les traitements nécessaires sont plutôt de type navigationnel qu'ensembliste, ce qui s'adapte bien à l'approche orientée objet.

La structure des SGBDOOs actuels est résolument tournée vers le modèle client-serveur à un niveau. Le serveur ne fait qu'archiver les données. L'ensemble des traitements est réalisé par les processus clients. Certains systèmes proposent quelques extensions à ce modèle. O2 [Deux90] offre une distribution logique à travers des fonctionnalités multibases. Cela permet d'utiliser un seul serveur pour servir plusieurs bases de données indépendantes. ObjectStore [Lamb91] propose aux clients de se connecter à plusieurs bases. Chacune de ces bases peut être gérée par un serveur différent. ObjectStore offre également un mécanisme de "Workspace/Configuration" qui permet de dériver une version alternative d'une base et de la partager entre plusieurs utilisateurs. Orion propose un serveur avec des fonctionnalités de traitement sur les données; une requête peut être exécutée sur l'application cliente ou sur le serveur.

Dans tous ces systèmes, les autres aspects de la distribution sont reportés vers des services externes comme CORBA. SHORE est un des premiers SGBDOOs qui propose une autre approche pour distribuer et pour accéder les données [Carey94]. Cette approche appelée symétrique instancie un processus serveur dans tous les noeuds d'un réseau.. Tous les serveurs sont interconnectés pour échanger les différentes parties de la base. Chaque application cliente se connecte uniquement au serveur local de sa machine pour obtenir l'ensemble des données de la base; le serveur local cache ainsi le réseau à ses clients. L'objectif principal de SHORE est de fournir des mécanismes permettant de distribuer d'une façon simple et transparente les objets sur le réseau. SHORE offre également des Services à Valeur Ajoutée (SVA). Un SVA propose un accès plus structuré aux données que le service primaire d'exportation des données "brutes". Un exemple de SVA est un service de requêtes OQL.

A l'exception de SHORE, peu de systèmes tiennent en compte de la distribution physique des données sur une architecture matérielle sous-jacente. Notre travail s'oriente vers la prise en compte de ces problèmes.

III. Modèle des Espaces de Travail.

Le modèle des espaces de travail est une application du modèle des transactions imbriquées à une hiérarchie de processus. Le but est d'obtenir un modèle d'architecture de processus facilement adaptable aux architectures d'applications distribuées. A mi-chemin entre modèle théorique et implémentation, le modèle des espaces de travail permet la conception d'applications base de données et la projection sur cette conception de contraintes matérielles comme par exemple les machines physiquement disponibles ou encore la localisation des données.

Le modèle se décompose en deux parties, le modèle de transactions et le modèle de processus. Le modèle de transactions se définit comme une extension du modèle des transactions imbriquées de [Moss85]. Les extensions portent sur la prise en compte de transactions distribuées et sur la possibilité pour une transaction fille de valider indépendamment de sa mère. Par ailleurs, le modèle de processus pose un certain nombre de contraintes sur le modèle de transactions afin de définir l'architecture des processus. Ces contraintes nous ont conduit à définir un processus générique, l'Espace de Travail, qui est la brique de base dans la construction d'applications distribuées.

III.1. Présentation informelle du modèle.

III.1.1. Le modèle de transactions imbriquées.

Le modèle de transactions présenté dans ce papier est une extension du modèle de transactions imbriquées de Moss. Dans ce modèle, une transaction imbriquée se définit comme une transaction pouvant contenir un ensemble de sous-transactions. Ces sous-transactions peuvent elles-mêmes être des transactions imbriquées. Une transaction ne contenant pas de sous-transactions est appelée transaction feuille. Une transaction qui n'est pas contenu dans une autre est appelée transaction racine. Une transaction contenant des sous-transactions est appelé transaction parent par rapport à ses sous-transactions. Celles-ci sont appelées transactions filles par rapport à leur transaction parent et transactions soeurs entre elles. L'ensemble des transactions racines sont considérées comme soeurs. La relation ancêtre (resp. descendant) correspond à la fermeture transitive de la relation parent (resp. fille). On peut représenter une transaction imbriquée sous forme d'un arbre de transaction.

Les transactions soeurs sont atomiques entre elles. Elles peuvent indépendamment valider ou abandonner. La validation d'une sous-transaction se fait dans l'espace de donnée de la transaction parent. La validation d'une transaction racine se fait dans la base de données. Lorsqu'une transaction demande l'accès à un objet, celui-ci est cherché récursivement parmi les ancêtres de la transaction. On renvoie la version de l'ancêtre le plus proche. Une transaction imbriquée ne peut valider que lorsque toutes ses sous-transactions ont été validées ou

abandonnées. Les modifications ne deviennent permanentes que lorsque la transaction racine valide.

Le contrôle de concurrence s'appuie sur les techniques de verrouillage. Chaque transaction peut posséder et conserver des verrous sur les objets de la base. Une transaction ne peut accéder et/ou modifier un objet que si elle possède le verrou correspondant dessus. Lorsqu'une transaction demande un verrou, elle ne l'obtient que lorsqu'il est conservé par un de ses ancêtres. Si elle l'obtient, on dit alors qu'elle possède le verrou. Lorsqu'une transaction valide, ses verrous sont hérités par sa transaction parent. La transaction parent conserve alors les verrous hérités.

III.1.2. Le modèle de transactions des espaces de travail.

Le modèle de transactions a pour objectif de définir un ensemble de règles d'accès aux objets de la base de données. Celle-ci se définit comme un ensemble de volumes, les volumes étant formés d'un ensemble d'objets. Pour chaque volume, il existe une transaction racine accédant directement au volume physique. Cette transaction peut être une transaction feuille ou une transaction imbriquée. On obtient donc pour chaque volume de la base de données, un arbre de transactions répondant au modèle des transactions imbriquées.

Les transactions se constituent une vision de la base de donnée en important des volumes. Elles peuvent avoir une vision partielle en important seulement certains volumes. L'importation peut être directe, la transaction est alors transaction racine pour ce volume. Elle peut être indirecte, la transaction devenant fille d'une autre transaction (au sens des transactions imbriquées) pour ce volume. Une transaction peut donc avoir plusieurs transactions parentes servant des parties disjointes de la base de données (i.e. volumes différents). Lors de la phase de validation, elle effectue une validation deux-phases sur l'ensemble des transactions parentes. Le modèle de transactions des espaces de travail peut être représenté par un graphe résultant de l'union des arbres de transactions imbriquées pour chacun des volumes de la base de données (figure 1).

Une transaction peut exporter les volumes qu'elle importe. Dans ce cas, une autre transaction devient une de ses transactions filles dès qu'elle importe un ou plusieurs volumes exportés. L'exportation des volumes par une transaction se fait dans les deux modes : passant ou englobant. Dans le cas d'un export englobant, la validation de ses sous-transactions et l'héritage de verrous obéissent aux règles du modèle des transactions imbriquées de Moss. Les objets modifiés sont validés dans l'espace de la transaction parent et les verrous sont conservés par cette dernière jusqu'à sa propre terminaison. Cependant, le modèle des espaces de travail ne fait pas la différence entre les verrous conservés et les verrous possédés. Dans le cas d'un export passant, les sous-transactions peuvent valider indépendamment de la transaction parent. La validation et l'héritage des verrous sont alors reportés vers les transactions grand-parents. La descente de la validation et l'héritage de verrous peuvent se poursuivre récursivement suivant le mode d'export des transactions ancêtres. Ils s'arrêtent au niveau des volumes physiques ou au niveau d'un ancêtre dont l'export est englobant. L'export passant transforme la transaction en cache de verrous et cache de pages. La section suivante exploite le mode passant pour spécialiser certains processus générique en processus serveur de base de données.

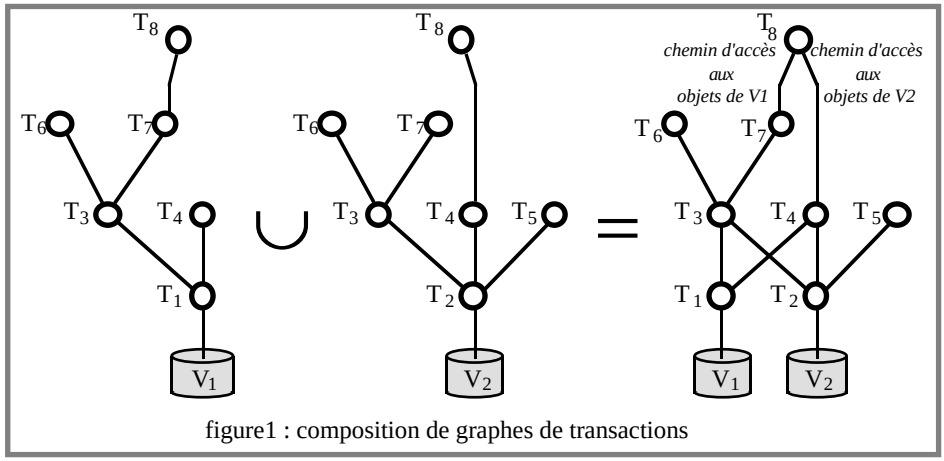

figure1 : composition de graphes de transactions

III.1.3. Le modèle de processus.

Le modèle de processus contraint notre modèle de transactions présenté précédemment afin d'obtenir un modèle d'architecture de processus distribués. L'idée maîtresse est la définition d'un processus générique possédant une couche de contrôle qui permet de le spécialiser suivant le rôle à tenir (ex : serveur de base de données, client simple, serveur coopératif,...).

D'un point de vue théorique, un processus générique se compose d'une transaction principale, d'une vision de la base de données construit par la transaction principale et d'un ensemble de sous-transactions. D'un point de vue pratique, il se compose des couches logicielles permettant l'accès direct ou indirect aux volumes de la base et d'un ensemble d'activités, une activité étant un processus léger (ou thread) exécutant du code dans le cadre d'une transaction. Une activité, dite activité principale, joue le rôle de transaction principale. Les autres activités, dites activités secondaires, exécutent des sous-transactions de la transaction principale.

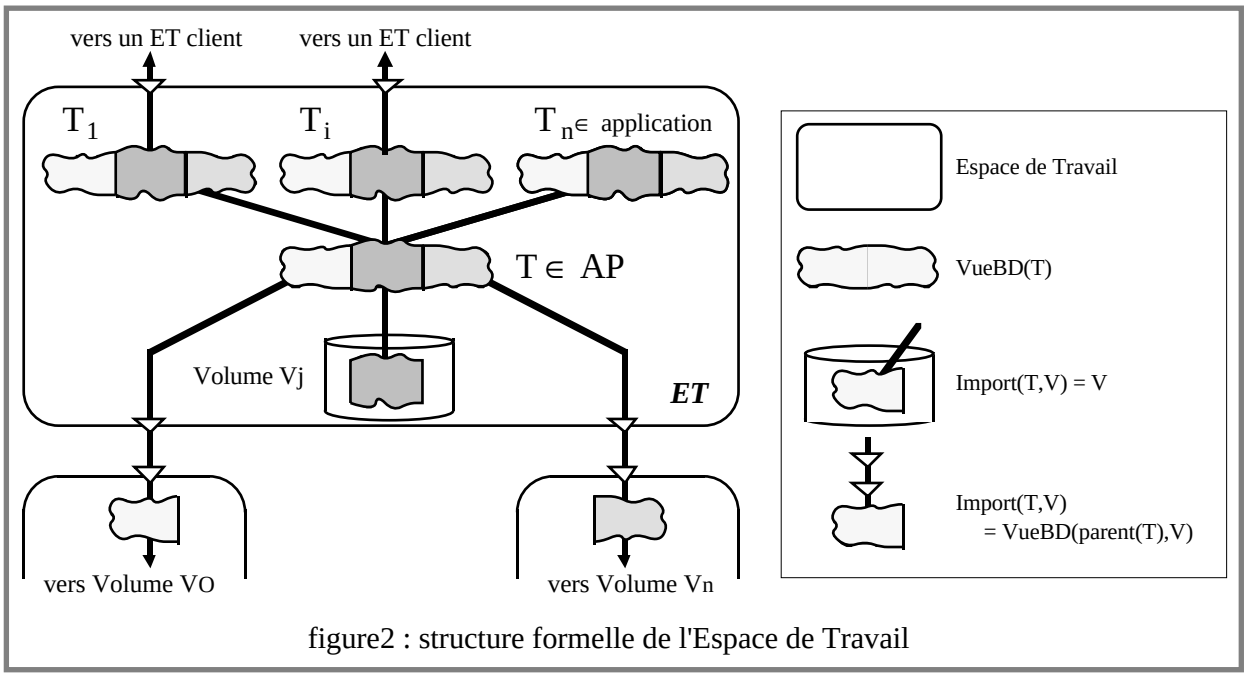

figure2 : structure formelle de l'Espace de Travail

La transaction principale peut décider de s'exporter ou de ne pas s'exporter. Dans le cas où elle ne s'exporte pas qu'il n'y a pas d'activité secondaire, le processus correspond à une application séquentielle. Il peut cependant disposer en local de volumes de la base de données. Dans le cas où la transaction principale s'exporte, elle peut le faire en mode passant ou englobant. Seules les sous-transactions locales peuvent devenir filles d'une transaction principale. Par ailleurs, les sous-transactions locales ne possèdent qu'une seule transaction parente qui est la transaction principale (figure 2).

Les sous-transactions locales peuvent s'exporter ou non. Si elles s'exportent, elles auront une seule transaction fille qui est la transaction principale d'un autre processus. Elles ne peuvent s'exporter qu'en mode passant. Si elles ne s'exportent pas, elles exécutent alors du code dans le cadre de sous-transactions locales. Ce code peut être celui d'une application attachée à un utilisateur ou bien celui d'un Service à Valeur Ajoutée (SVA) proposant à d'autres processus une autre manière d'accéder aux données persistantes.

Lors de la connexion à la base de données, la transaction principale compose sa vue à partir d'import sur chaque volume. Les imports peuvent être locaux (i.e. volumes locaux) ou distants au moyen d'une relation parent-enfant avec une sous-transaction d'un autre processus (i.e. un serveur). Cette approche permet de reconfigurer dynamiquement la topographie de la base et donc de pouvoir l'adapter aux besoins. On peut facilement passer d'une architecture monoprocessus avec tous les volumes en local à un graphe de processus générique, sans changer le code de l'application.

III.2. Formalisation du modèle des espaces de travail.

III.2.1. Le modèle de transactions.

Définissons tout d'abord la base de données. L'élément atomique est l'objet. Un objet est un couple formé d'un identifiant ou OID et d'une valeur. Un volume est un ensemble d'objets. Une base est un ensemble de volumes.

Soit OID l'ensemble des identifiants partitionnés en OidVolume
Soit VAL l'ensemble des valeurs

On note $O = OID \times VAL$

Soit V_i *un volume,* $V_i = \{ (oid,val) \in O / oid \in OidVolume_i \}$
Soit BD une base de données, BD est un sous-ensemble de O composée de volumes

Une transaction est une fonction de O dans O réalisant la modification d'un certain nombre d'objets. On peut donc voir la transaction comme un sous-ensemble de $O \times O$. L'ensemble d'arrivée forme la vue de la base de données pour la transaction. On partitionne cet ensemble par volume.

$T \subset O \times O$

VueBD(T) = { VueVolume(T,V) tq $V \in BD$ *}*

La vue des objets d'un volume pour une transaction est une union entre l'ensemble des objets du volume importés depuis la transaction parent et l'ensemble des objets modifiés par la transaction.

VueVolume(T,Vi) = VueLocale(T,Vi) $\cup$ *VueImportee(T,Vi)*

VueLocale(T,Vi) = { (oid,val) $\in O$ *tq oid* $\in OidVolume_i$ *et*
$(\exists (oid, val') \in O$ *tq* $(oid,val') \times (oid', val) \in T$ *et* $val \neq val') \}$

VueImportée(T,Vi) =
VueVolume(T^{-1}*,Vi) tq* T^{-1} *= Parent(T) au sens des transactions imbriquées*
ou
V_i

Soit le prédicat Export,

Export(T,V) = Vrai ssi la transaction s'exporte pour le volume V

Le mode d'exportation ne varie pas suivant les volumes exportés. Il n'est donc fonction que de la transaction.

ModeExport(T) $\in$ *{passant englobant}*

Considérons maintenant la validation d'une transaction. Elle se décompose en un ensemble de validation sur chaque volume accédé. La validation totale est réalisé à l'aide d'une validation à deux phases.

Validation (T) = {ValidationVolume(VueLocale(T,V_i*),* T_i*) tq* T_i *= Parent(T) pour* V_i*}*

Dans le cas d'une transaction père ayant réalisé un export passant, la validation est répercutée plus bas sur l'arbre de transaction.

Si ModeExort(T_i) = passant

alors ValidationVolume(VueLocale(T, V_i), T_i) $\Leftrightarrow$ ValidationVolume(VueLocale(T, V_i), T_i^{-1})

avec T_i^{-1} = parent(T_i) pour V_i et T_i = Parent(T) pour V_i

Dans le cas d'une transaction père ayant réalisé un export englobant, la validation est une projection de l'espace local de la transaction enfant dans l'espace local de la transaction parente.

Soit Proj (A,B) la projection des éléments de A dans l'ensemble B

On a Proj (A,B) $\Leftrightarrow B = A \cup B$

Si ModeExort(T_i) = englobant

alors ValidationVolume(VueLocale(T, V_i), T_i) $\Leftrightarrow$ Proj(VueLocale(T, V_i), VueLocale(T_i, V_i))

avec T_i = Parent(T) pour V_i

III.2.2. Le modèle de processus.

On définit une activité comme une séquence de transaction

On définit un processus générique comme une activité principale, notée AP et un ensemble d'activité secondaire notée AS_i. L'ensemble d'activités secondaires peut être vide.

$\forall Ti \in AP,\ \forall Tj \in AS_n,\ Ti = parent(Tj)$ *et* $\neg\exists ET$ *tq* $T = parent(Tj)$ *et* $T \neq Ti$

$\forall T \in AS_i,$ *si* $Export(T)$

alors $ModeExport(T) = passant$

et $\exists Ts$ *unique tq* $T = parent(Ts)$.

$\forall T \in AP,\ \forall Ts,\ T = parent(Ts) \Rightarrow Ts \in AS_i$

IV. Variantes d'Architectures Bases de Données.

Le modèle des Espaces de Travail définit une brique générique pour construire une structure du SGBD adaptée à l'architecture matérielle hôte. Dans cette section, nous instançions le modèle afin de construire les architectures adaptées de SGBD telles que les bases de données privées pour PC isolé, les bases client-serveur sur un réseau local ou dans un système d'entreprise à 3 niveaux. L'instanciation consiste à spécialiser certains ET constituant le graphe d'architecture en restreignant la définition de ces ET. Nous terminerons par l'offre de "Services à Valeur Ajoutée" au sien de ces architectures de Bases de Données.

IV.1. Bases de Données Privées.

L'architecture de bases de données la plus simple est la base de données privée : elle correspond en général à une architecture matérielle qui se réduit à une machine isolée (i.e. PC). Le succès de l'informatique personnelle a conduit les grands fournisseurs de systèmes bases de données à porter leurs produits mainframe vers cette informatique en plein essor.

La modélisation d'une telle utilisation des bases de données privées se formalise de la manière suivante dans le modèle de transaction des Espaces de Travail :

D1

- *soit AP, $AS_S \in ET$, $\forall V \in Base$*

a) $\forall T \in AP,\ VueVolume(T,V) \in VueBD(T),\ VueImportée(T,V) = V$

b) $\forall T_S \in AS_S, \neg\exists\ ET\ \ tq\ \ T^{+1} \in ET,\ T_S = Parent(T^{+1})\ \ et\ \ Export(T_S,V) = faux$

L'application est un Espace de Travail dont la vue de la transaction principale n'est composée que de Volumes Locaux qui contiennent les objets de la base privée (figure 3.a). L'application est exécutée par la transaction principale ou par ses sous-transactions. Cette dernière possibilité introduit le parallélisme dans l'exécution de l'application et offre à son développeur un moyen implicite de synchroniser les activités concurrentes qui composent l'application.

Les bases de données privées ne s'appliquent pas seulement au contexte des machines isolées (personnelles ou embarquées); il peut également s'appliquer à la phase de développement d'une application bases de données. Dans ce cas, le développeur met au point l'application, prévue pour une base distribuée, sur une base de test privée. Cette approche permet de séparer le travail de développement du travail d'exploitation.

IV.2. Bases de Données Client-Serveur.

L'architecture Client-Serveur des Gérants d'Objets répond à un besoin de partage et de sûreté de fonctionnement : la gestion de la persistance des données est isolée de l'exécution de l'application. Le client est la tâche qui exécute l'application et le serveur centralise les données et en maintient une archive consistante. Les tâches client et serveur peuvent être sur la même machine physique : les applications ne peuvent plus provoquer l'arrêt du serveur de données ou corrompre l'intégrité de la base. D'autre part dans un environnement distribué, les communications sont les seuls moyens d'échange entre deux machines distinctes et donc de partage de données entre des applications distribuées.

Dans un contexte mono-serveur, la base n'est servie que par un seul serveur. Ceci se modélise en spécialisant la définition des ET. Dans l'ET serveur, l'activité principale accède directement à l'ensemble de Volumes et ses sous-transactions exportent leur VueBD. Les ET clients se connectent à l'ET serveur pour importer la VueBD. L'ET client exécute une application au moyen de la transaction principale ou bien de ses sous-transactions concurrentes.

L'exportation de vue des sous-transactions de l'ET serveur est de nature passante; les sous-transactions du client valident leurs modifications dans les volumes de la base de données.

D2C

- *soit AP, $AS_S \in ET$ client,*

a) $\forall V_i, V_j \in Base,\ T \in AP, VueVolume(T,V) \in VueBD(T),$

si $VueImportée(T,V_i) = VueVolume(T_i^{-1},V_i)\ \ tq\ \ T_i^{-1} = parent(T)$

et $VueImportée(T,V_j) = VueVolume(T_j^{-1},V_j)\ \ tq\ \ T_j^{-1} = parent(T)$

alors $T_i^{-1} = T_j^{-1}$

b) $\forall V \in Base$, *identique à* **D1. b**

D2S

- *soit AP, $AS_S \in ET$ serveur,* $\forall V \in Base$

a) *identique à* **D1. a**

b) $\forall T_S \in AS_S,\ VueVolume(T_S,V) \in VueBD(T_S),$

$Export(T_S,V) = vrai\ \ et\ \ ModeExport(T_S) = passant$

Dans un contexte multi-serveurs, les volumes de la base sont répartis entre plusieurs serveurs (figure 3.b). Le client doit se connecter à chacun de ces serveurs pour constituer une image complète de la base. Ceci entraîne une nouvelle définition des ET serveurs et des ET clients. C'est en fait une approche multi-base car les serveurs n'ont a priori aucune relation entre eux. Dans ce type d'approche, le client pilote la validation "2-phases" des transactions qu'il exécute,

auprès des différents serveurs (munis de moniteurs transactionnels TP/XA [Xopen91]). Ceci peut être une faille dans le maintien de l'intégrité de la base distribuée si le calcul des règles d'intégrité nécessitent la valeur de données validées auprès des différents serveurs.

D3C

- *soit AP, AS$_S$ ∈ ET client, ∀V ∈ Base*

a) $\forall T \in AP,\ VueVolume(T,V) \in VueBD(T),\ VueImportée(T,V) \neq V$

b) *identique à* **D1. b**

IV.3. Architecture Symétrique de Serveurs.

Une autre modélisation d'une base multi-serveurs est l'architecture Symétrique Pair-vers-Pair (peer-to-peer symetric) qui propose de ne mettre un client en contact qu'avec un seul serveur (figure 3.c). Dans cette architecture, chaque serveur ne détient qu'une partie des volumes de la base et il exporte ceux-ci comme une vue "partielle" de la base vers les autres serveurs, ses pairs. Il importe également les vues "partielles" proposées par ses pairs et fusionne celles-ci avec la vue sur les volumes locaux afin d'exporter une vue "complète" de la base vers les clients. Un client ne se connecte qu'à un seul serveur; ce serveur lui masque ainsi la distribution de la base et coordonne la validation 2-phases de ses transactions.

Cette approche a été choisie par les concepteurs de SHORE [Carey94]. SHORE équipe chaque machine du réseau d'un processus serveur sur lequel les applications clientes de la station se connecte pour obtenir la vue complète de la base. Le serveur local joue ici le rôle de cache de données pour les applications de la machine. Cependant, ce placement systématique des processus serveurs sur les machines du réseau pose des problèmes de sécurité car il suppose que les machines ne sont administrées par leurs utilisateurs; ce qui est rarement le cas avec des stations de travail ou des PCs. Le modèle de ETs est plus flexible car il permet à l'application cliente et le processus serveur de demeurer sur des machines séparées.

D4C ___*identique à* **D2C**_____

D4S

- *soit AP, AS$_S$ ∈ ET serveur, ∀V ∈ Base,*

a) $T \in AP,\ VueObjetVolume(T,V) \in VueBD(T),$

$$VueImportée(T,V) = V \text{ ou } VueVolume(T^{-1},V)$$

$$tq\ T^{-1} = Parent(T),\ \exists T^{-2} = Parent(T^{-1})\ tq\ VueImportée(T^{-2},V) = V$$

b) identique à **D2S. b**

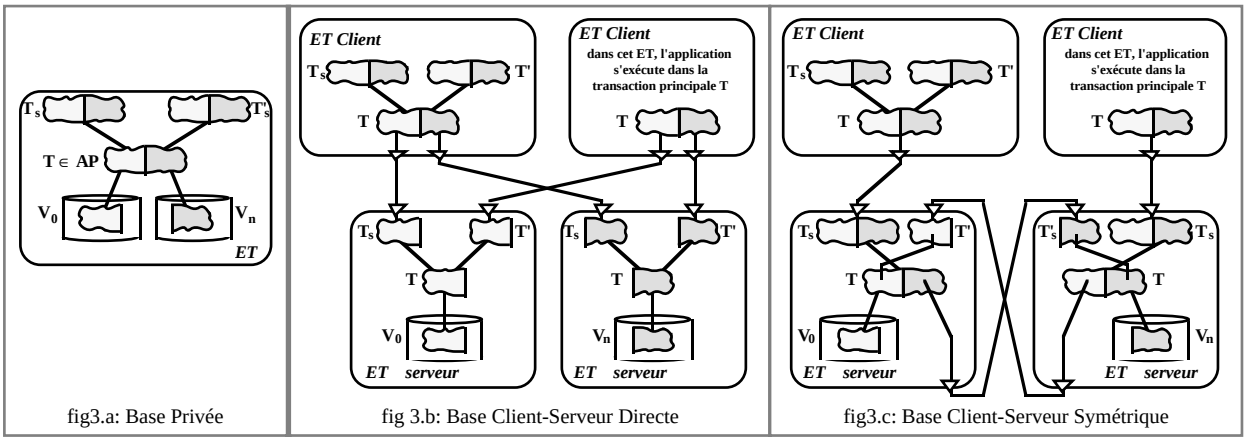

fig3.a: Base Privée — fig 3.b: Base Client-Serveur Directe — fig3.c: Base Client-Serveur Symétrique

IV.4. Frontal de Réseau Longue Distance.

Le système informatique de l'entreprise suit en général le modèle à 3 niveaux. Il se constitue de centres de traitement qui regroupent sur un réseau local haut débit plusieurs stations de travail et

un serveur départemental archivant l'information locale au centre de traitement. L'information globale est accessible en dialoguant avec le serveur central de l'entreprise ou bien avec les serveurs départementaux des autres centres de traitement au travers d'un réseau longue distance (figure 4.a).

Dans cette architecture, le serveur départemental joue le rôle de frontal réseau longue distance. En effet, ces échanges longue distance représentent une part importante du coût de l'exploitation du système et un frein aux performances. Dans ces conditions, il convient d'éviter que deux postes de travail, dans le même centre, engagent deux communications séparées pour rapatrier la même donnée "outre-centre". Le frontal réseau concentre ainsi les demandes des postes vers les serveurs externes au réseau local et cache les données rapportées pour l'ensemble des postes du centre.

La modélisation de cette architecture à 3 niveaux est réalisée en introduisant un Espace de Travail frontal jouant le rôle d'intermédiaire dans l'architecture client-serveur précédente (fig4.b). La formalisation de cette nouvelle architecture conserve la définition de l'ET client de l'architecture symétrique (D2C) et celle de l'ET serveur de l'architecture classique Client/Serveur (D2S). D5I définit de l'ET intermédiaire.

D5C	*identique à* **D2C**
D5I	• *soit AP, AS$_S$ ∈ ET frontal, ∀V ∈ Base,* *a) identique à* **D3C. a** *b) identique à* **D2S. b**
D5S	*identique à* **D2S**

Dans cette approche, l'information globale peut être également chargé en bloc sur le frontal (serveur départemental) pour une certaine durée afin d'optimiser les échanges entre les postes et les serveurs centraux. Ce mécanisme de "check in/check out" est modélisé au moyen de l'exportation englobante des ET frontaux. Les modifications, apportées par les transactions des postes clients, sont conservées localement sur l'ET frontal; elles ne sont pas retournées immédiatement aux serveur de données pour limiter les échanges longue distance. Les modifications des transactions validées sur l'ET frontal (i.e. check-in) sont définitivement archivées sur les serveurs lorsque l'activité principale AP de l'ET frontal valide sa transaction courante.

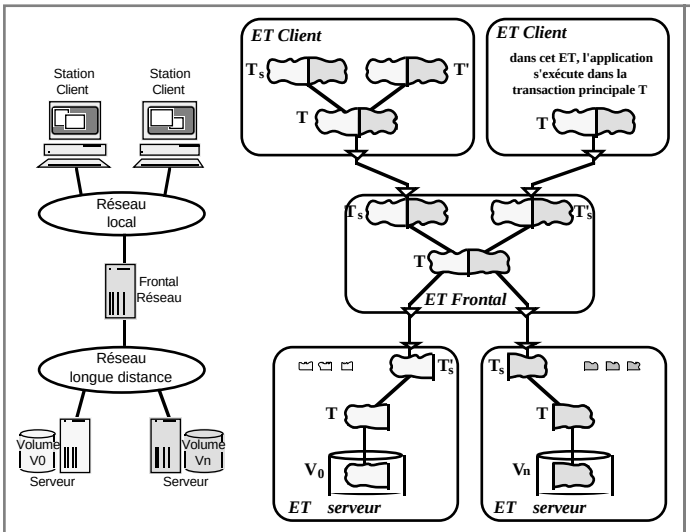

fig4.a: Architecture à 3 niveaux fig4.b: Base Client-Serveur avec Frontal

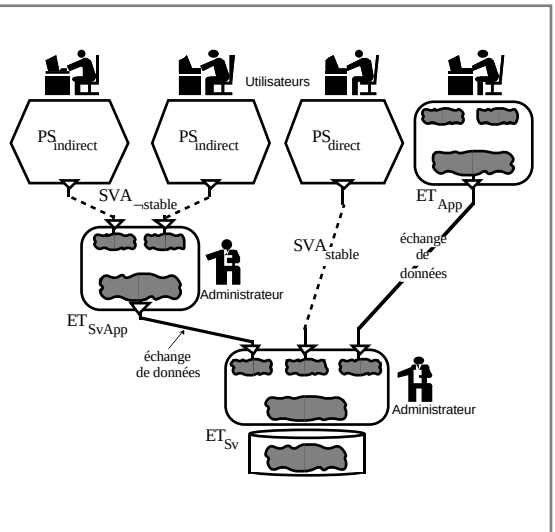

fig4.c: Serveurs à Valeur Ajoutée

IV.5. "Serveurs" à Valeur Ajoutée.

Nous avons supposé dans les variantes d'architectures précédentes que chaque sous-transaction d'un ET client exécute une partie du code d'une application. Cette application est un processus attaché à un utilisateur de la base. Du point de vue de l'ET, la notion d'application a un sens plus large; elle peut être également un "Service à Valeur Ajoutée" (SVA) offert à d'autres processus utilisateurs.

Dans les architectures courantes de SGBDOOs, ces services sont implantés par des processus clients du serveur de données (i.e. ET_{SvApp}). Ces clients sont "serveurs applicatifs" dont le code est celui de services à valeur ajoutée. Cette séparation entre le serveur applicatif et le serveur de données est rassurante du point de sécurité cependant elle conduit à une perte d'efficacité lorsque le serveur applicatif peut être considérer comme fiable.

Ce problème, qui est aussi connu sous le nom de "problème du serveur au niveau client", se résout en autorisant le serveur de données à fournir également des SVA. La généricité des ETs permet de définir des serveurs de données qui proposent aussi directement des SVA. Dans la figure 4.c, l'espace de travail ET_{Sv} exporte une vue de la base aux espaces de travail client ET_{SvApp} et ET_{App}; il fournit en même temps un SVA fiable au processus PS_{direct}. La figure comporte aussi un SVA en court de mise au point proposé par le client ET_{SvApp}.

Les services à valeur ajoutée du type OQL modifient la nature du trafic entre le client et du serveur : le client et le serveur n'échangent plus de données brutes (i.e. Data-Shipping) mais des demandes de calcul et leurs résultats (i.e. Query-Shipping). Ce mode d'interaction est indispensable aux environnements sécurisés dans lesquels la base ne peut être dévoilée au client. Il est également préférable pour des opérations peu sélectives telle qu'un balayage complet de la base. Cependant, le Query-Shipping conduit l'application à retourner vers un système centralisé (i.e. un serveur puissant et des CPU clientes faiblement utilisées). Nous avons défini un SVA permettant à un client de combiner les avantages des deux modes d'interaction. Le client peut importer des données pour y effectuer ses calculs; néanmoins il peut également demander au serveur de réaliser un calcul sur les données comme une opération privilégiée (réalisable uniquement par le serveur sécurisé) ou bien un filtrage.

V. Implantation des Espaces de Travail.

Notre implantation des Espaces de Travail tire profit des systèmes d'exploitation récents. La structure des processus ET est "multithreadée" afin d'exécuter des transactions simultanées mais également pour faciliter les échanges entre les noeuds d'un graphe d'Espace de Travail. L'accès aux données par les transactions utilise le Memory-Mapping; cette approche nous a conduit à choisir la page comme granularité de communication, de verrouillage et de réplication dans les caches des ETs. Les paragraphes suivants décrivent la structure multithreadée des Espaces de Travail, et la gestion des objets à l'intérieur d'un ET

V.1. Structure des Espaces de Travail

La structure des ETs est orientée vers l'exploitation du parallélisme des machines actuelles et vers la nécessité pour les ET serveurs (primaires ou intermédiaires) de traiter les demandes des clients de manière asynchrone. Les machines modernes offrent deux formes de parallélisme : la plus connue est le parallélisme de calcul des multiprocesseurs mais la forme la plus répandue est le parallélisme des entrées/sorties (disques, réseaux) qui est particulièrement recherchée pour les serveurs bases de données. Les systèmes d'exploitation modernes comme Solaris2 [Kleiman92] ou Mach offrent un moyen efficace, le MultiThreading (MT), pour exploiter ces deux formes de parallélisme. Le MT considère le processus comme un ensemble de ressources (espace d'adressage, descripteurs de fichiers, ports de communication, ...) partagées par des activités simultanées, les "threads".

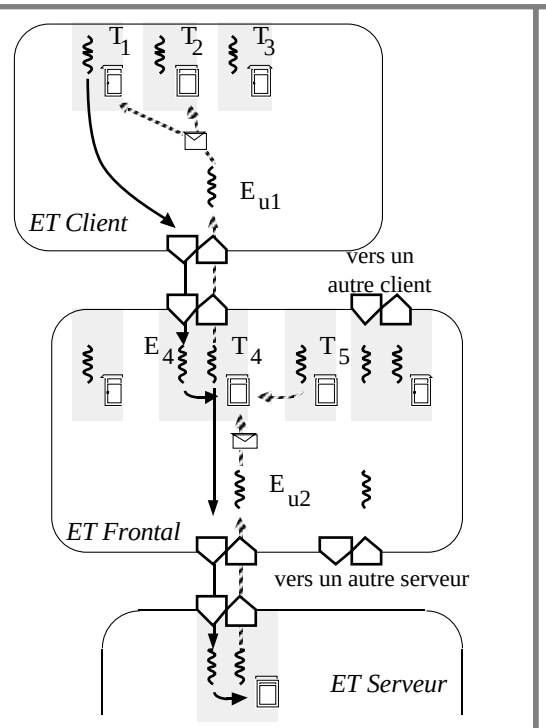

Fig5.a: Structure MT sur noyau réentrant

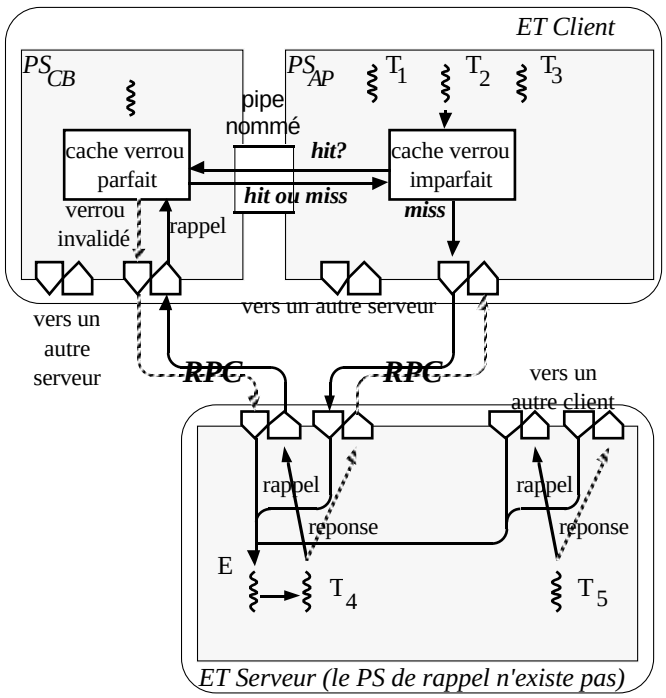

fig5.b: Structure MT sur noyau non réentrant

V.1.1. Threads transactionnelles.

Notre implantation transpose directement ces notions à celles définies par le modèle des Espaces de Travail : l'Espace de Travail est représenté par un processus dont les threads exécutent les transactions (nous verrons en V.2 que cette transposition existe également entre l'espace d'adressage et les VueBDs). Dans le cadre d'un ET serveur, les threads permettent de répondre simultanément à plusieurs clients. Un ET client sur une machine multiprocesseur peut de son coté exécuter plusieurs sous-transactions afin de paralléliser son traitement.

Les threads sont également utilisés afin de rendre asynchrones les dialogues à l'intérieur du graphe de ETs. Le client envoie des requêtes sans attendre la réponse des précédentes; le serveur peut répondre à celles-ci dans un ordre différent que celui des émissions. Cette propriété rend indépendante les threads à l'intérieur d'un processus ET. Ce mode de communication asynchrone permet également à un ET serveur de rappeler les verrous cachés sur les ETs clients dans le cadre du "verrouillage par rappel" (Callback Locking [Wang91]). Ce rappel se propage récursivement dans la hiérarchie de clients et de serveurs.

V.1.2. Structures de l'Espace de Travail.

La structure de l'ET s'articule autour des threads transactionnelles (TT), des threads d'écoute (TE) et d'une messagerie interne. Nous introduisons dans cette structure des threads d'écoute pour pouvoir recevoir des messages de la part d'un client, d'un serveur ou bien d'une autre TT. Les TT reçoivent tous ces messages par une boîte à lettre de la messagerie interne; les TE sont chargées de rediriger les messages externes vers les boites à lettre des TT[1]. La figure 5.a schématise le déroulement d'une demande. Dans l'ET client, la thread transactionnelle T_1 réclame directement une page non cachée au serveur par le port de communication et se place en attente de réponse. T_2 qui demande également cette page, évite la redondance de demande et se place aussi en attente de la réponse. Le serveur reçoit les messages par E_4 associée au port de

[1] Les threads ne peuvent à la fois être bloqués sur des entrées/sorties et sur des synchronisations inter-threads des "boites à lettre".

réception. T_4 reçoit cette demande par sa boite à lettre; elle répond immédiatement si la page est disponible et ne répond pas sinon. T_4 demande donc la page au serveur de niveau inférieur ou la récupère dans sa vue locale; elle se place ensuite en attente d'un message quelconque. Si T_4 obtient la réponse (i.e. la page ou le verrou), elle remonte celle-ci vers son client; E_{U1} reçoit la page (ou le verrou), l'installe puis finalement poste un message de présence à T_1 et à T_2.

Nous proposons également une alternative (figure 5.b) à cette structure multithreadée qui permet aux ETs de s'exécuter sur la génération précédente de systèmes d'exploitation (ex: Solaris1). Comme ces systèmes sérialisent les appels au noyau, l'ET est décomposé en deux processus : le processus PS_{AP} exécute les pseudo-threads transactionnelles et le processus PS_{CB} gère le Callback Locking en cachant les verrous de l'ET. Dans $PS_{AP,}$ les threads demandent directement les pages non cachées au serveur; le serveur répond immédiatement soit en retournant la page et son verrou, soit en demandant au client (à sa thread) de tenter de réclamer de nouveau la page ultérieurement. A la terminaison de la transaction, PS_{AP} communique ses verrous obtenus au PS_{CB}.

V.2. Gestion d'objets dans un processus ET

L'architecture des ET s'appuie sur un gestionnaire d'objets qui offre un accès homogène à l'ensemble de la base de données. Il permet de se connecter à un ensemble d'ETs serveurs tout en cachant la distribution. Le gestionnaire d'objets se compose d'un serveur virtuel de volume qui effectue les connections avec les ET serveurs, d'une gestion de mémoire virtuelle et d'un service d'allocation d'objet.

V.2.1. Accès aux objets

Le système d'accès utilise sur des identifiants physiques d'objet avec un décodage logiciel. Les objets sont contenus dans des pages. Un identifiant de page se compose du numéro de volume et du numéro de pages dans le volume. Un identifiant d'objet se compose d'un identifiant de page et du numéro d'objet dans la page. Dans le cas où la page dans laquelle se trouve l'objet n'est pas en mémoire, on en fait la demande auprès du serveur virtuel de volume. Celui-ci retrouve le processus serveur responsable du volume et la lui demande. La page est alors chargée en mémoire et le système crée une "poignée" dessus. Dans le cas où la page est en mémoire, on utilise une technique de hachage pour retrouver la poignée. La structure de hachage est un arbre de profondeur variable dont les branches sont désignées par des bits extraits de l'identifiant de page.

Le choix d'identifiant physique se justifie par le besoin de rendre autonome volumes et pages que l'on distribue au travers du graphe d'ET. Nous avons préféré un décodage logiciel à une technique de swizzling [White92-94] dans un but de simplification de l'architecture. Ce choix a été conforté par les résultats du banc d'essais présentés dans la section VI.

V.2.2. Mémoire virtuelle

La mémoire virtuelle utilise les techniques de mappage mémoire (memory mapping). On utilise un fichier de taille fixe. Chaque fois que l'on charge une page, on alloue une page libre dans le fichier. On alloue aussi une adresse mémoire virtuelle que l'on fait correspondre à la page allouée dans le fichier. S'il n'y a pas de page libre dans le fichier, on recherche une page éjectable (page lue mais non modifiée) et l'on protège l'adresse mémoire virtuelle correspondante contre tout accès. Lors du prochain accès à cette adresse, on rechargera la page depuis le serveur. Dans le cas où il n'y a ni page libre, ni page éjectable, on double la taille du fichier. Cette technique permet une gestion très simple et transparente de la mémoire virtuelle.

V.2.3. Interface C++

La mémoire virtuelle, la gestion de l'allocation et le système de décodage fournissent la vision d'une mémoire d'objets persistants. Les objets sont créés, consultés, modifiés et détruits dans le cadre des transactions. Comme ces objets sont en fait des contenants persistants sans

signification, notre système propose une vision d'instances C++ de la base d'objets : Les instances des classes C++ définies par le développeur sont allouées dans ces contenants sans signification. Cette interface suit les recommandations de l'ODMG 93 avec quelques différences syntaxiques qui disparaîtront dans une prochaine version.

VI. Mesures des Performances.

Nous avons validés les choix techniques de notre implémentation appelée Yooda au moyen du banc d'essai OO7 de l'université du Wisconsin [Carey93]. OO7 se présente comme un compromis entre complexité de mise en oeuvre et couverture de test, et il s'est rapidement imposé comme le test d'évaluation des principaux SGBDOOs existants.

Le schéma de la base de données OO7 exprime les modèles que l'on trouve traditionnellement dans les applications de conception assistée par ordinateur ou dans les applications de génie logiciel. L'élément de base est la pièce composée. Elle regroupe quelques attributs comme la date de création, un identifiant et un ensemble de pièces atomiques la formant. Chaque pièce composée dispose d'un objet document composé d'un texte long. Les pièces atomiques forment un graphe dont les interconnections sont réalisées par l'intermédiaire d'objet connexion possédant quelques attributs comme un type de connexion et une longueur. Le degré d'interconnexions entre les pièces atomique constitue un des paramètres du banc d'essai. Les pièces d'assemblage se composent de pièces composées ou d'autres pièces d'assemblage; elles constituent ainsi un arbre d'accès aux pièces composées.

Le tableau ci-dessous présente quelques résultats du banc d'essai pour deux tailles de base. Dans la base S3, chaque pièce composée est formée de 20 pièces atomiques, chacune étant relié à trois autres pièces atomiques. Dans la base M3, une pièce composée est formée de 200 pièces atomiques, chacune étant relié à trois autres pièces atomiques. Pour la base S3, les résultats "à chaud" permet de mesurer l'efficacité du cache client.

Performances comparées de SGBDOOs avec Yooda sur les bancs d'essai OO7

Essais (en seconde)	**Exodus**	**Ontos**	**Objectivity**	**Objectstore**	**Yooda**
t1 froid S3	34,8	28,9	38,5	22,7	19,9
t1 chaud S3	10,6	8,1	17,9	6,2	5,1
t1 M3	734,5	1064,6	548,7	372,5	322,3
t2b froid S3	40,5	39,8	60,8	35,9	41,7
t2b M3	963,3	901,7	1329,4	519,6	527,3
t9 froid S3	0,2	1,3	8,3	1,2	0,03
t9 chaud S3	0,002	0,002	0,02	0,01	0,001
t9 M3	0,2	4,9	11,1	1,1	0,88
q2 froid S3	2,0	4,8	10,8	11,0	2,3
q2 chaud S3	0,008	0,01	0,06	0,04	0,002
q2 M3	18,0	34,8	33	52,1	19,2
Taille BD S3 (en Mo)	11,5	4,2	5,7	4,4	7,1
Taille BD M3 (en Mo)	103,8	51,7	54,6	37,5	55,3

Le banc d'essai a été mis en oeuvre à l'aide de deux Sparc IPX (21.8 SPEC92Int) sous SunOS 4.1.3. La machine serveur disposait de 32 Mo de mémoire centrale et la machine cliente de 24 Mo de mémoire. Le banc d'essai publié dans [Carey93] utilise une machine légère moins puissante Sparc ELC (18.2 SPEC92Int) en client: nous avons réajusté les temps de la colonne Yooda afin de permettre une comparaison à puissance égale bien que pénalise Yooda pour les bancs d'essai à froid dans lequel le coût réseau est majoritaire. Remarquons qu'ODI a procédé au même réajustement [ODI93].

- t1 est un parcours de l'arbre d'assemblages. Pour chaque pièce composée atteinte, on visite toute ses pièces atomiques. Cette opération permet de mesurer la vitesse de traversé de pointeurs. Les temps de Yooda sont comparables à ceux d'ObjectStore bien que ce dernier

utilise des techniques de swizzling de pointeurs. Cela confirme le choix de Yooda de réaliser l'accès aux objets par décodage logique et non par swizzling.

- t2b est une variation de t1 dans laquelle on effectue des mises à jour dans chaque pièce atomique rencontrée. La mise à jour dans Yooda se fait par rapatriement des pages modifiées sur le serveur. Bien que Exodus ne transfère vers le serveur que les delta d'objets ayant été modifiés, on observe qu'une architecture simplifiée basée sur un transfert de pages (très proche des mécanismes systèmes) permet d'obtenir de meilleure performances.

- t9 est une opération de manipulations sur les objets longs. Elle compare le premier et le dernier octet de l'objet document. Cela permet de voir si le système peut paginer correctement les objets longs. La gestion par pages des objets longs, utilisée dans Yooda, permet d'obtenir de bonnes performances à froid comme à chaud sur ce type d'opérations.

- q2 exprime une sélection des pièces atomiques sur un attribut non indexé (sélectivité = 1%). On teste la qualité des index. Les résultats de Yooda confirme une certaine efficacité dans l'implémentation des index mais doit être relativisé car la requête q1 est exprimé en C++ et non dans un langage de requêtes comme c'est le cas pour certains systèmes.

VII. Conclusion.

L'architecture client-serveur des systèmes de bases de données distribuées n'est pas adaptée à une distribution transparente des objets sur des réseaux hétérogènes. Un premier pas en direction de la transparence de l'accès à des serveurs multiples a été obtenu par des architectures de mémoire virtuelle distribuée symétrique comme celle de Shore. Chaque client est associé à un serveur, qui cache à son client les accès aux autres serveurs ainsi que les procédures de connexion et de déconnexion.

Toutefois cette architecture n'adresse pas le problème des réseaux hétérogènes. Ceux-ci sont caractérisés par des réseaux locaux connectés entre eux et à des mainframes par l'intermédiaire de réseaux distants. Des serveurs départementaux servent de frontal entre le réseau local et la machine distante. Une hiérarchie de serveurs devient donc nécessaire.

Dans cette article, nous proposons de réaliser cette extension de l'architecture client-serveur par un mappage entre les transactions du modèle des transactions imbriquées de Moss, et des processus communiquants qui gèrent des vues sur la base de données. Le processus communiquant est la brique générique de cette architecture. Chacun d'eux est à la fois client et serveur.

Un processus communiquant, appelé Espace de Travail, gère une vue sur les données. Cette vue comporte des objets locaux et des objets importés. Le processus est composé de plusieurs activités ou threads. Chaque activité est une thread qui exécute une séquence des transactions. Dans un Espace de Travail, il existe une transaction principale, qui gère la vue locale et les importations, et des transactions filles, qui gèrent les exportations vers les Espaces de Travail clients. Nous présentons un modèle formel de cette architecture. Nous décrivons aussi son application pour générer différentes architectures effectives, depuis la machine isolée jusqu'à un réseau de machines à plusieurs niveaux.

Cette architecture est bien adaptée à des réseaux étendus d'entreprises. Il en existe une implémentation industrielle, disponible par ftp anonyme (ftp.uu.net, /pub/database/yooda). Cette implémentation présente de bonnes performances dans l'exécution du banc d'essais OO7.

Dans le futur, nous prévoyons de poursuivre le développement de cette version domaine public. D'autres axes de recherche que nous envisageons actuellement sont les suivants :

- adaptation à la gestion d'objets distribuée sur des très grands réseaux, notamment sur l'Internet, où les contraintes de débit sont différentes.

- étude du placement des données et des requêtes, notamment pour favoriser la recherche d'informations distribuées sur le réseau.
- adaptation à la gestion de flots de données composites et synchrones, de type multimédia.

L'aspect que nous souhaitons adresser est de gérer dans le modèle des Espaces de Travail des flots de données produits par des serveurs multimédia en cours de développement par d'autres équipes.

Références

[Abécassis 95] E. Abécassis, "YOODA: Handling Distribution Through OODBMS", soumis à publication

[Alexander 88] W. Alexander, G. Copeland,"Process And Dataflow Control In Distributed Data-Intensive Systems",Proc. of the 1988 SIGMOD Conf, Chicago, Illinois, Juin 1988.

[Carey 93] M.J. Carey, D.J. DeWitt, J.F. Naughton, "The OO7 Benchmark", Proc. of the 1993 SIGMOD Conference.

[Carey 94] M.J. Carey, D.J. DeWitt, M.J. Franklin, et al, "Shoring Up Persistent Applications", Proc. of the 1994 SIGMOD Conf., Minneapolis, Minnesota, Mai 1994.

[Deux 90] O. Deux et al, "The Story of O2", IEEE transactions on knowledge and data engineering vol.2 n°1 1990

[DeWitt 90] D.J. DeWitt et al, "The Gamma Database Machine Project",IEEE Transactions on Knowledge ans Data Engineering, Vol 2, No 1, Mars 1990, p44-62

[Donsez 94] D. Donsez, P. Homond, P. Faudemay, "WEA, A Distributed Object Manager based on a Workspace Hierarchy", Proc. of IFIP Conf. on Applications in Parallel and Distributed Computing, Caracas, Venezuela, Avril 1994, pp247-256.

[Kleiman 92] S. Kleiman et al, "Symmetric Multiprocessing in Solaris 2.0", COMPCON, p181, San Francisco, Californie, Printemps 1992.

[Lamb 91] Charles Lamb, Gordon Landis, Jack Orenstein, Don Weinreb, "The ObjectStore Database System", Communication of the ACM, Octobre 1991, Vol. 34, No. 10, pp50-63

[Moss 85] J.E.B. Moss, "Nested Transactions: An Approach to Reliable Distributed Computing", Boston, MIT Press, 1985.

[Odi 93] Object Design Inc, "Understanding the OO7 Research Project", Version 1.0, April 4, 1993.

[Stonebraker77] M. Stonebraker, E. Neuhold, "A Distributed Database Version of INGRES", proc of 5th BerkeleyWorkshop on Distributed Data Management and Computer Networks, ACM/IEEE, 1981, pp143-153.

[Wang 91] Y.Wang, L.A. Rowe, "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture", Proc. of the 1991 SIGMOD Conference, pp367-376.

[White 92] S.J. White, D.J. DeWitt, "A Performance Study of Alternative Object Faulting and Pointer Swizzling Strategies", Proc; of the 18th VLDB Conference, British Columbia, Canada 1992.

[White 94] S.J. White, D.J. DeWitt, "QuickStore: A High Performance Mapped Object Store", Proc. of the 1994 SIGMOD Conf., Minneapolis, Minnesota, Mai 1994, pp395-406.

[Xopen 91] X/Open Company, "X/Open Snapshot SQL Remote Database Access", 1991.