REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université El Hadj Lakhder – Batna

Faculté des sciences **Institut d'informatique**

N° d'ordre :…….
Série :……..

Mémoire présenté en vue de l'obtention du diplôme de
Magistère en informatique

Option : *Informatique industrielle*

Sujet du mémoire :

Perspectives d'utilisation des SDDS pour l'implémentation du niveau physique d'une base de données relationnelle

Présenté et soutenu le : ……....

Par : **BENBELGACEM SONIA**

Devant le jury composé de :

Mr. Bilami Azzedine	**Président**	*(Professeur à l'université de Batna)*
Mr. Belattar Brahim	**Rapporteur**	*(Maître de conférence à l'université de Batna)*
Mr. Kazar Okba	**Examinateur**	*(Maître de conférence à l'université de Biskra)*
Mr. Zidani Abdelmadjid	**Examinateur**	*(Maître de conférence à l'université de Batna)*

Perspectives d'utilisation des SDDS Pour l'implémentation du niveau physique d'une base de données relationnelle

Je dédie ce travail à ceux qui ont pensé à moi et à ma réussite,

Ma famille & mes ami(e)s.

Remerciement

Tous mes remerciements à mon créateur

Ensuite, avec l'aide et la grâce de dieu,

Je tiens à remercier tout d'abord monsieur BELATTAR BRAHIM pour avoir accepté de m'encadrer pédagogiquement.

Je tiens aussi à remercier messieurs les membres de jury qui sont les honorables rapporteurs de ce mémoire, pour leur lecture attentive, ainsi que pour leurs critiques et suggestions constructives.

A toutes les personnes qui m'ont aidé, encouragé et soutenu afin que ce travail puisse être accomplie, je leur exprime ma gratitude.

Un remerciement particulier aux employés des départements (informatique & génie civil) de l'université de Batna.

Table des matières

Chapitre 2 : Multiordinateurs et réseaux P2P

Chapitre 3 : Les SDDS (Structures de données distribuées et scalables)

Chapitre 4 : SDDS et BD

Chapitre 5 : Utilisation des SDDS dans le niveau physique d'une BDR

Liste des figures

Liste des tableaux

Liste des algorithmes

Liste des acronymes

BD	base de données.
BD P2P	base de données peer to peer.
BDR	base de données relationnelle.
DHT	table de hachage distribuée.
E/S	entrée / sortie.
IAM	message d'ajustement d'image.
LAN	réseau local.
MAN	réseau métropolitain.
P2P	peer to peer.
PAN	réseau personnel.
SDDS	structure de données distribuées et scalables.
SGBD	système de gestion de bases de données.
SGBDR	système de gestion de bases de données relationnel.
SGFD	système de gestion de fichiers distribués.
WAN	réseau étendu.

Résumé

La technologie informatique a connaît une évolution importante dans le domaine des bases de données, surtout avec l'apparition des réseaux informatiques qui ont permis d'élargir et de faciliter la communication, le traitement et également le stockage des données.

En effet, un nouveau concept est né, celui de multiordinateur, permettant d'offrir des capacités gigantesques de stockage et de calcul parallèle. Les applications actuelles n'arrivent pas à tirer profit de ces capacités. Pour cela, une nouvelle structure de données dénommée Structure de Données Distribuées et Scalables (SDDS) est apparue. Il s'agit de fichiers résidants en mémoire vive distribuée permettant un accès beaucoup plus rapide que les structures de données classiques sur des disques. Ainsi elle permet d'assurer la disponibilité des données, leur distribution et leur scalabilité (maintien des performances en cas d'accroissement du volume de données stockées).

Les SDDS ont pratiquement des applications plus ou moins variées, leur utilisation par les SGBD semble très importante du fait que cette structure supporte le traitement parallèle et permet le partitionnement dynamique des données de la base. Cette structure a marqué aussi son importance pour le stockage et le traitement des requêtes.

Ainsi, le but de ce mémoire est de démontrer la possibilité et les perspectives d'utilisation de ces nouvelles structures de données dans le niveau physique d'une base de données relationnelle distribuée.

Mots clés : multiordinateur, architecture peer to peer, structures de données distribuées et scalables (SDDS), bases de données relationnelles (BDR), niveau physique d'une BDR, système de gestion de fichiers (SGF), système de gestion de bases de données (SGBD).

Introduction générale

Ces dernières années sont marquées par une avancée évidente de la technologie informatique, l'apparition des réseaux a favorisé ces progrès, en effet, pour exploiter au maximum les ressources cumulées de stockage et de traitement, une nouvelle configuration a été proposée : les multiordinateurs, et afin de réaliser le but souhaité, une nouvelle structure de données dénommée SDDS (structure de données distribuées et scalables) a été définie pour cette configuration.

La nouvelle structure été proposée la première fois en **1993** par Dr. Witold Litwin, il s'agit de fichiers résidents en mémoire centrale (RAM) distribuée.

Cette structure supporte le traitement parallèle, assure une capacité de stockage illimitée et un temps d'accès aux données beaucoup plus rapide que celui des fichiers classiques stockés sur disque. Un fichier SDDS s'adapte dynamiquement à l'accroissement du volume de données, il grandi rapidement en gardant les mêmes performances et c'est ce qui est appelé « la scalabilité », en d'autre terme : la monté en échelle.

Cette structure créée initialement pour les fichiers distribués et introduite récemment dans le domaine des bases de données a montré son intérêt dans ce domaine. Pour une base de données relationnelle, les données sont représentées par des tables implémentées à leur tour sous forme de fichiers logés sur un support de stockage d'où un fichier est divisé en pages contenant chacune un ensemble d'enregistrements et d'informations nécessaires à la localisation de ces données, cette représentation est nommée « le niveau physique de la BD relationnelle ».

Ainsi, ce mémoire est accompli afin d'étudier les possibilités de l'utilisation des SDDS dans une configuration multiordinateur pour implémenter le niveau physique d'une base de données relationnelle distribuée, et la répercussion du choix de cette

structure sur l'architecture fonctionnelle du système de gestion des bases de données relationnel, notamment l'exécution des requêtes SQL.

Organisation du mémoire

Ce mémoire sera organisé comme suit :

Un premier chapitre

Qui explique les notions importantes dans le domaine des bases de données, celle du système de gestion des bases de données, du système de gestion des fichiers, ainsi que le modèle relationnel.

Le niveau physique des BD est aussi évoqué dans ce chapitre, notamment les organisations et les méthodes d'accès aux données.

Un second chapitre

Qui évoque la configuration multiordinateur, appelée aussi la technologie peer to peer (p2p), tout en indiquant les différentes applications de ces technologies.

Un troisième chapitre

Qui introduit la nouvelle structure de données appelée (SDDS), et qui explique le principe de ses deux catégories distinguées : les SDDS LH* et les SDDS RP*.

Un quatrième chapitre

Qui englobe un état d'art sur les travaux réalisés dans le domaine des BD p2p et l'utilisation des SDDS avec les SGBD.

Un cinquième chapitre

Qui résume l'étude de la possibilité et les perspectives de l'utilisation des SDDS pour implémenter le niveau physique d'une base de données relationnelle distribuée sur une configuration multiordinateur et indique la répercussion du choix de cette structure sur l'architecture fonctionnelle du système de gestion des bases de données relationnel.

Et enfin une ***conclusion*** pour achever et entrevoir les perspectives futures de ce travail.

Chapitre 1

BD & SGBD Relationnels

Introduction

Les bases de données ont pris aujourd'hui une place essentielle dans l'informatique, plus particulièrement en gestion. Au cours de ces dernières années, des concepts, des méthodes et des algorithmes ont été développés pour gérer les différents types de données. Ainsi plusieurs systèmes de gestion de bases de données ont été proposés comme par exemple, les Systèmes de Gestion de Bases de Données : hiérarchiques, réseaux, relationnels, objets, objets-relationnels, déductives,….etc.

Généralement un Système de Gestion de Base de Données inclut en son cœur une gestion de fichiers, pour définir la politique de stockage ainsi que le placement des données, dans le modèle relationnel par exemple, la base de données est implantée simplement de la manière suivante : à chaque relation correspond une zone de stockage appelée « *fichier* » et à chaque entité d'une relation correspond un *enregistrement* du fichier. L'avantage de cette méthode de stockage est l'élimination des pointeurs pour stocker les associations entre plusieurs entités de relations différentes, ce qui est sans doute une source d'allègement de la BD, d'une augmentation de son efficacité et de sa sûreté.

Aujourd'hui, des méthodes d'accès de plus en plus performantes ont été élaborées afin d'augmenter les performances des SGBD en terme de traitement et de stockage.

Dans le présent chapitre, nous abordons les BD (particulièrement celles du modèle relationnel), leurs systèmes de Gestion ainsi que leur niveau interne tout en passant par l'explication des notions liées aux fichiers et aux organisations et méthodes d'accès à ces fichiers.

I. Notion d'une base de données

Plusieurs définitions ont été proposées pour désigner une base de données, on trouve par exemple qu'une base de données est :

> « un ensemble de données modélisant les objets d'une partie du monde réel et servant de support à une application informatique » [5].
>
> « Un ensemble d'informations structurées permettant la mise en place d'une série d'applications informatiques destinées à une grande variété d'utilisateurs possible » [46].
>
> « une collection de données inter-reliées, stockées ensemble pour servir une ou plusieurs applications, en parallèle, de façon optimale » [47].

II. Notion d'un système de gestion de base de données (SGBD)

Comme pour la base de données, le système de gestion de base de données a été défini de plusieurs façons, parmi ces définitions on cite les suivantes :

> « un SGBD peut apparaître comme un outil informatique permettant la sauvegarde, l'interrogation, la recherche et la mise à jour en forme de données stockées sur mémoires secondaires » [5].
>
> « Un SGBD peut être vu comme un système informatique (un logiciel) spécialisé dans le traitement de gros volumes d'informations et permettant à différents utilisateurs d'interagir avec la base de données. Il doit permettre de définir la structure de la base et d'y introduire les données correspondantes. De plus, une fois la base créée, il faudra d'une part la mettre à jour et d'autre part l'exploiter ou l'interroger » [46].
>
> « Un SGBD est un ensemble de programmes assurant structuration, stockage, maintenance, mise à jour et recherche des données d'une base de données en plus des interfaces nécessaires aux différentes formes d'utilisation de la base » [47].

II.1. Objectifs d'un SGBD

Des objectifs principaux ont été fixés aux SGBD dès l'origine de ceux-ci et ce, afin de résoudre les problèmes causés par la démarche classique. Ces objectifs sont les suivants [47, 80] :

Indépendance physique.
Indépendance logique.
Accès aux données.
Administration facilitée des données.
Manipulation des données par des langages non procéduraux.
Non redondance des données.
Cohérence des données.
Partage des données.
Sécurité des données.
Résistance aux pannes.

II.2. Architecture des SGBD

II.2.1. Architecture générale des SGBD

Généralement, un SGBD se compose de trois couches présentées sur Fig.1.1 et qui sont :

1)- un gestionnaire de fichiers (SGF) : pour la gestion des récipients de données sur les mémoires secondaires.

2)- un SGBD interne : pour la gestion des données stockées dans les fichiers, les objets, les liens, les structures d'accès,…etc. Cet SGBD repose sur un modèle de données interne, par exemple, des tables reliées par des pointeurs.

3)- un SGBD externe : assure d'une part l'analyse et l'interprétation des requêtes utilisateurs en primitives internes, d'autre part la transformation des données extraites de la base de données échangées avec le monde extérieur.

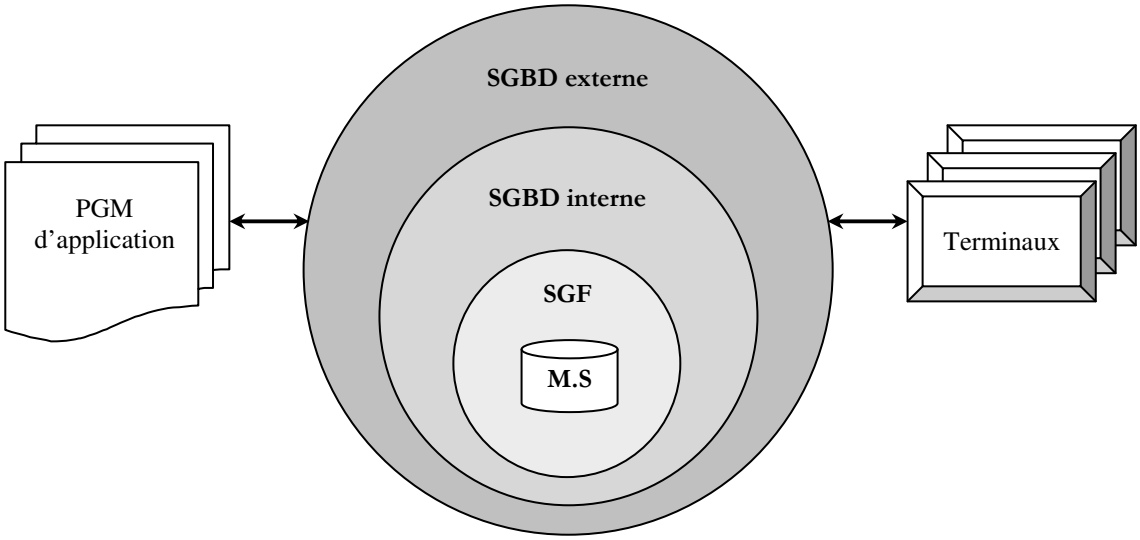


Fig.1.1.Première vue d'un SGBD

II.2.1.1. Les niveaux d'abstraction d'un SGBD

Un objectif majeur des SGBD est d'assurer une abstraction des données stockées sur disques pour simplifier la vision des utilisateurs. Pour cela, trois niveaux de description de données ont été distingués par le groupe ANSI/X3/SPARC [5] :

Niveau conceptuel d'un SGBD

Correspond à la structure canonique (structure sémantique inhérente) des données qui existe dans l'entreprise, le schéma conceptuel permettra de définir :

- Les types de données élémentaires qui définissent les propriétés élémentaires des objets de l'entreprise.
- Les types de données composées qui permettent de regrouper les attributs afin de décrire les objets du monde réel ou les relations entre objets ou bien encore les associations du monde réel.
- Les règles que devront suivre les données au cours de leur vie dans l'entreprise.

Dans le cas des SGBD relationnels, il s'agit d'une vision tabulaire où la sémantique de l'information est exprimée en utilisant les concepts de relation, d'attributs et de contraintes d'intégrité. On appelle cette description *le schéma conceptuel.*

Niveau externe d'un SGBD

Correspond à la perception de tout ou partie de la base par un groupe donné d'utilisateurs, indépendamment des autres. Au niveau externe, chaque groupe de travail possède une description des données perçues, appelée *schéma externe*. Cette description est effectuée selon la manière dont le groupe voit la base dans ses programmes d'application. Le niveau externe assure l'analyse et l'interprétation des requêtes en primitives de plus bas niveau et se charge de convertir les données brutes, issues de la réponse à la requête, dans un format désiré par l'utilisateur.

Niveau interne ou physique d'un SGBD

S'appuie sur un système de gestion de fichiers pour définir la politique de stockage ainsi que le placement des données. Donc le niveau physique est responsable du choix de l'organisation physique des fichiers et de l'utilisation d'une méthode d'accès précise en fonction de la requête. La description correspondante à ce niveau est appelée *schéma interne*.

Il faut noter que les définitions proposées par ce niveau ne doivent pas affecter les applications sauf sur le plan des performances d'accès à la BD afin de garantir l'indépendance entre données et programmes qui représente un objectif essentiel du SGBD.

Le souci du schéma physique est donc de pouvoir changer l'organisation physique des données sans modifier le schéma conceptuel ni le schéma externe.

Par exemple, pour augmenter les performances d'accès à la base de données, on peut :

Changer l'organisation d'un fichier (passer d'une organisation séquentielle à une organisation séquentielle indexée ou directe).

Déplacer physiquement le fichier vers une autre adresse sur le support de stockage.

Modifier les chemins d'accès aux enregistrements (changer d'index, ajouter d'autres indexes, …etc.).

II.2.2. Architecture opérationnelle des SGBD

Il existe un ensemble d'architectures opérationnelles regroupées en catégories, parmi lesquelles on citera très brièvement les suivantes :

Architecture centralisée : le SGBD et les programmes d'application s'exécutent sur un même ordinateur central.

Architecture client–serveur : architecture hiérarchisée mettant en jeu d'une part un serveur de données gérant les données partagées en exécutant le code du SGBD avec d'éventuelles procédures applicatives, d'autre part des clients pouvant être organisés en différents niveaux supportant les applications de la présentation, et dans laquelle les clients dialoguent avec les serveurs via un réseau en utilisant des requêtes de type question-réponse [5].

Architecture client–serveur à deux strates (two-tiered) : architecture client-serveur mettant en jeu d'une part un serveur de données exécutant le SGBD et éventuellement des procédures applicatives, d'autre part des clients.

Architecture client–serveur à trois strates (three-tiered) : architecture client-serveur mettant en jeu un serveur de données exécutant le SGBD et éventuellement des procédures applicatives, un serveur d'applications exécutant le corps des applications et des clients responsables de dialogues et de la présentation des données selon les standards du web.

Architecture répartie : architecture composée de plusieurs serveurs coopérant à la gestion de bases de données composées de plusieurs sous-bases gérées par un seul serveur, mais apparaissant comme des bases uniques centralisées pour l'utilisateur [5].

II.2.3. Architecture fonctionnelle des SGBD [5]

II.2.3.1- Architecture à trois niveaux de l' ANSI/X3/SPARC

Proposée vers la fin des années 70 par le groupe ANSI/X3/SPARC, cette architecture intègre les trois niveaux de schémas (conceptuel, externe et interne), elle est articulée autour du dictionnaire de données et comporte deux parties :

1- Un ensemble de modules (appelés processeurs) permettant d'assurer la description de données et donc la constitution du dictionnaire de données.

2- Une partie permettant d'assurer la manipulation des données, c'est-à-dire l'interrogation et la mise à jour des bases.

Dans chacune des parties, on retrouve les trois niveaux d'abstraction, l'architecture proposée est présentée par la figure Fig.1.2.

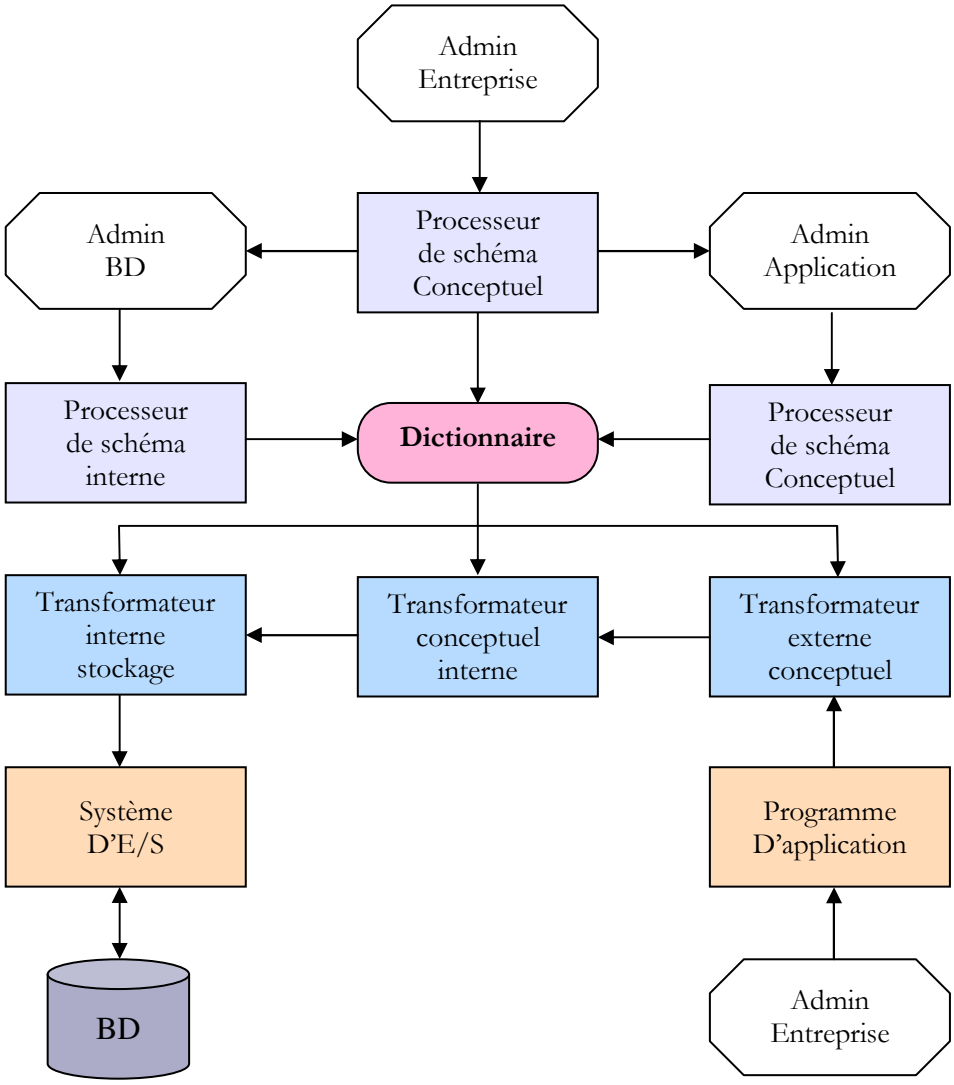


Fig.1.2. Architecture ANSI/X3/SPARC

II.2.3.2- Une Architecture fonctionnelle de référence

La plupart des GSBD actuels utilise une architecture typique, présentée par Fig.1.3, basée sur seulement deux niveaux de schéma : le schéma et les vues.

Le schéma correspond à une intégration des schémas interne et conceptuel, une vue est un schéma externe.

Du point de vue de *la description* de données, un SGBD gère un dictionnaire de données appelé « métabase », car il est souvent organisé sous forme d'une base de données décrivant les autres bases.

Du point de vue de *la manipulation* des données, les requêtes sont prises en compte par un *analyseur de requête* qui est responsable de l'analyse syntaxique et sémantique de la requête, ensuite elle est transférée vers un *contrôleur de requête*, qui a la responsabilité de sa modification, en plus, il se charge du contrôle des droits d'accès et d'intégrité lors de la mise à jour de la base.

Deux autres composants très importants constituant le SGBD sont : *l'optimiseur de requête* qui a le rôle d'élaborer un plan d'accès optimisé pour traiter la requête, et *l'exécuteur de plan* qui se charge d'exécuter ce plan en s'appuyant sur les méthodes d'accès aux fichiers de la base de données.

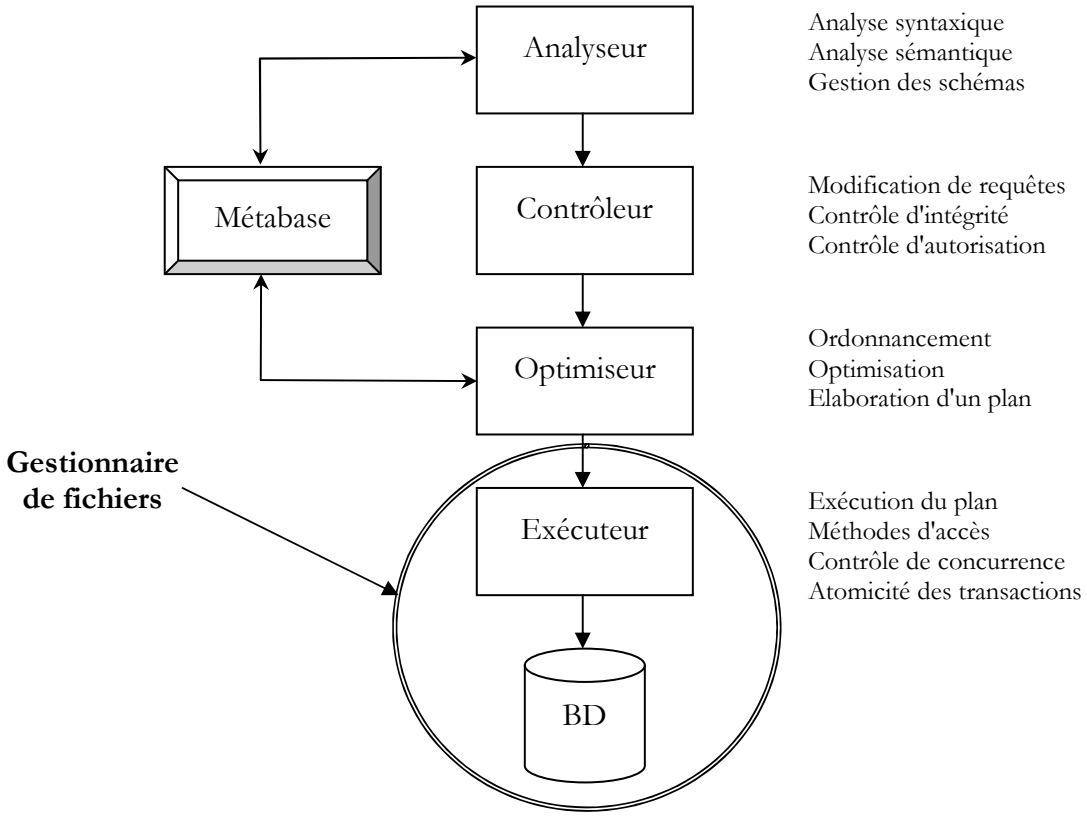


Fig.1.3.Architecture typique d'un SGBD

II.3. Le modèle relationnel

Le modèle relationnel qui est aujourd'hui la base de nombreux systèmes, a été introduit par E.F. Codd [56], il est conçu pour répondre aux objectifs suivants:

Permettre un haut degré d'indépendance des programmes d'application et des activités interactives à la représentation interne des données, en particulier aux choix des ordres d'implantation des données dans les fichiers, des index et plus généralement des chemins d'accès.

fournir une base solide pour traiter les problèmes de cohérence et de redondance des données.

permettre le développement des langages de manipulation de données non procéduraux basés sur des théories solides.

En plus, le modèle relationnel a atteint deux autres objectifs non prévus initialement :

Etre un modèle extensible permettant de modéliser et de manipuler simplement des données tabulaires, mais pouvant être étendu pour modéliser et manipuler des données complexes.

Devenir un standard pour la description et la manipulation des BD.

Ce modèle propose la représentation des entités sous forme de *tables* appelées *relations*, chaque table est composée de plusieurs colonnes, chacune d'elles correspond à un *attribut* prenant ses valeurs dans un *domaine*. On peut aussi dire qu'une relation est un sous-ensemble du produit cartésien de N domaines. Elle peut être identifiée par une *clé* et reliée avec les autres relations par des *contraintes d'intégrité référentielles*.

Le modèle relationnel présente de nombreux avantages dus au fait qu'il soit basé sur la théorie des ensembles : Langage de manipulation des données ensemblistes grâce à l'algèbre relationnelle et grâce à des langages assertionnels qui permettent de spécifier ce que l'on souhaite obtenir sans dire comment l'obtenir. Le SGBD est responsable de la politique d'exécution des requêtes [57].

II.4. Structures physiques d'une BDR

Le modèle relationnel permet une vision logique très simple des relations : des tables, manipulé par un langage facile à utiliser (SQL), qui ne spécifient aucune information physique.

Une vision naïve de l'implantation physique des relations est possible : chaque relation est un fichier simplement séquentiel de données sur disque. Un tel mode de stockage conduit, à l'évidence, à des performances affreux : l'organisation physique des relations sur disque ne peut se contenter d'une simple organisation séquentielle des informations. Les bases de données, utilisent naturellement des organisations physiques plus sophistiquées. Ces méthodes d'accès, efficaces, ont pour but de déterminer, à partir d'une clé de recherche, l'ensemble minimum des pages de disque à lire.

A. Notion de fichier

Dans [5] Gardarin a défini un fichier en tant qu'un Récipient de données identifié par un nom et contenant des informations systèmes ou utilisateurs.

En plus de son nom, un fichier se caractérise par un créateur, une date de création, un ou plusieurs types d'article, un emplacement en mémoire secondaire et une organisation, il peut être exploité par des programmes d'application, qui peuvent lui accéder à l'aide d'une méthode d'accès précise.

B. Structure d'une page

Une page (fig.1.4) est une unité de transfert entre disque et RAM [71], les pages sont généralement utilisées pour adresser les fichiers. La taille de la page est choisie de façon à contenir plusieurs enregistrements physiques, elle dépend parfois de l'organisation du fichier (séquentielle, directe, non-ordonnée, hachée, arbres-B ou autres...).

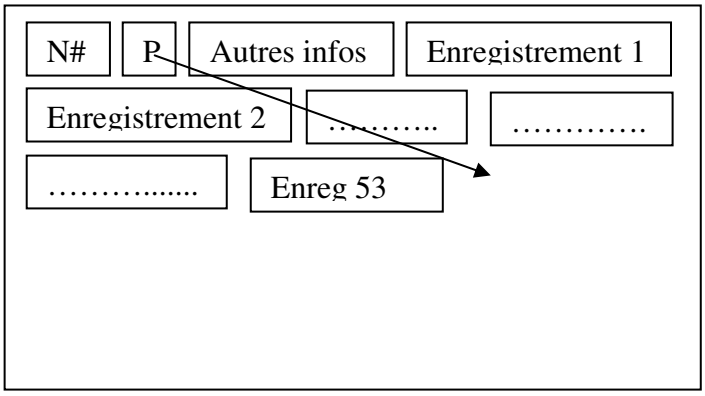


Fig .1.4. Structure d'une page physique

C. L'adressage Relatif d'un fichier

Un fichier étant généralement discontinu sur mémoire secondaire, il est utile de pouvoir adresser son contenu à l'aide d'une adresse continue de 0 à n appelée adresse relative (Fig.1.5), c'est le Numéro d'unité d'adressage dans un fichier c'est-à-dire le déplacement par rapport au début du fichier. On peut dire que le fichier est découpé en pages

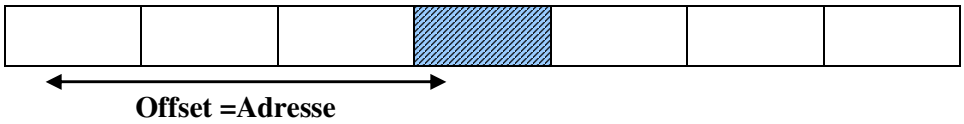


Fig.1.5.Adresse relative

Donc un SGBD inclut en son cœur une gestion de fichiers, on peut dire que c'est son premier niveau. Historiquement, la notion de fichier a été introduite en informatique dans les années 50, afin de simplifier l'utilisation des mémoires secondaires des ordinateurs et de fournir des récipients de données plus manipulables aux programmes. Aujourd'hui, des méthodes d'accès de plus en plus performantes ont été élaborées.

II.5. Architecture d'un SGF

Un gestionnaire de fichiers est composé d'un ensemble de modules ou couches (Fig.1.6). Chaque couche constitue une machine abstraite qui accomplit un certain nombre de fonctions accessibles aux couches supérieures par des primitives constituant l'interface de la couche.

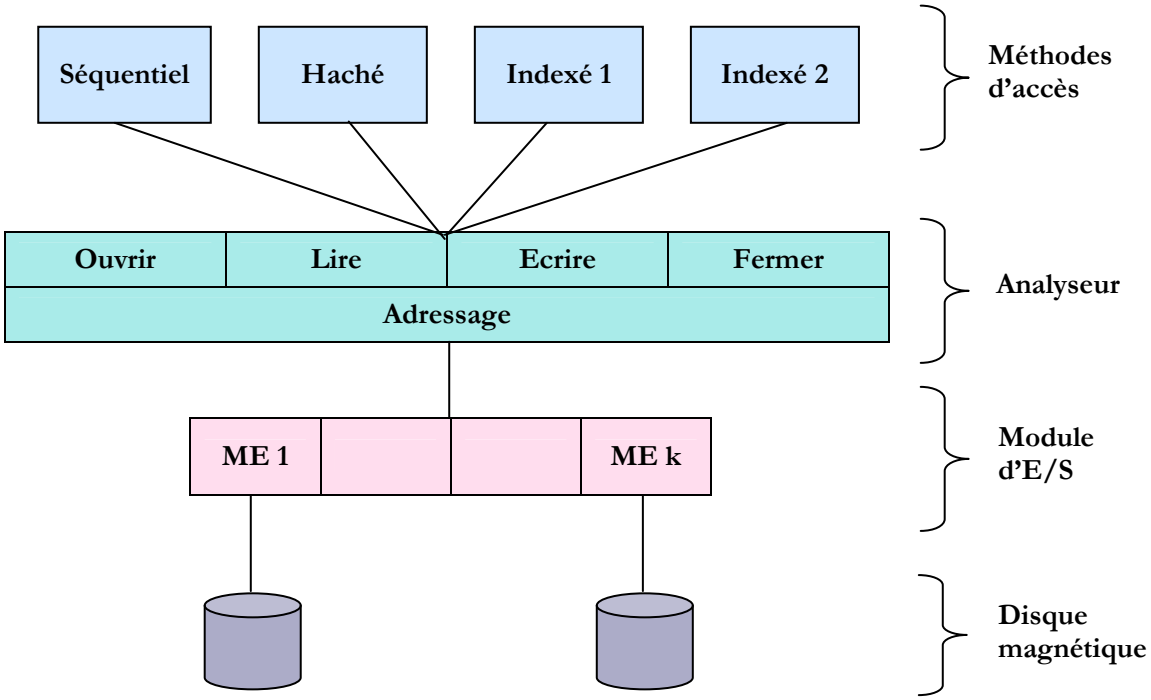


Fig.1.6.Architecture d'un SGF

Le SGF est généralement structuré autour d'un noyau, appelé *analyseur*, qui est chargé d'assurer les différentes fonctions de base de la gestion (manipulation des fichiers, l'adressage relatif, allocation de la place sur mémoire secondaire, localisation des fichiers sur les volumes, classification des fichiers en hiérarchie et le contrôle du fichier).

Les méthodes d'accès sont des modules spécifiques qui constituent une couche externe et qui utilisent les fonctions du noyau.

Le gestionnaire d'entrées/sorties n'appartient pas à proprement dit au SGF, mais au système opératoire, il gère les E/S physiques et leurs fils d'attente, il permet aussi au noyau d'accéder aux mémoires secondaires. En outre, il est responsable de la lecture/écriture des blocs physiques de données sur tous les périphériques et de la gestion des particularités de chacun.

II.6. Organisation et méthodes d'accès à un fichier

On distingue plusieurs organisations et méthodes d'accès à un fichier, elles sont regroupées en catégories, parmi lesquelles on trouve :

les organisations et méthodes d'accès séquentielles : un accès séquentiel signifie qu'il doit être accédé à des éléments dans une séquence préétablie et ordonnée, sa structure de base est généralement représentée par les listes chaînées.

les organisations et méthodes d'accès directes : parmi lesquelles on distingue :

les organisations et méthodes d'accès par hachage.

les organisations et méthodes d'accès indexées.

les organisations et méthodes d'accès multi-attributs.

Dans ce qui suit, nous parlons des organisations et méthodes d'accès par hachage et indexées.

II.6.1. Organisation et méthodes d'accès par hachage

Les organisations et méthodes d'accès par hachage [5] sont basées sur l'utilisation d'une fonction de calcul qui, appliquée à la clé, détermine l'adresse relative d'une zone appelée paquet dans laquelle est placé l'article. On distingue les méthodes d'accès par hachage statique dans lesquelles la taille du fichier est fixée et des méthodes d'accès par hachage dynamique, où le fichier peut grandir.

II.6.1.1. hachage statique

C'est la méthode la plus ancienne et la plus simple, son résultat est un fichier haché statique illustré par Fig.1.7, qui est un fichier de taille fixe dans lequel les articles sont placés dans des paquets (Fig.1.8) dont l'adresse calculée à l'aide d'une fonction de hachage fixe appliquée à la clé.

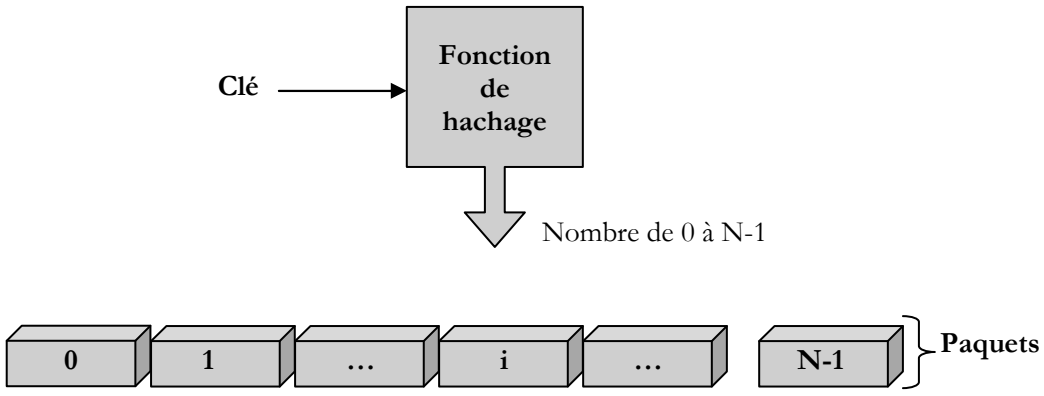


Fig.1.7. Illustration d'un fichier haché statique

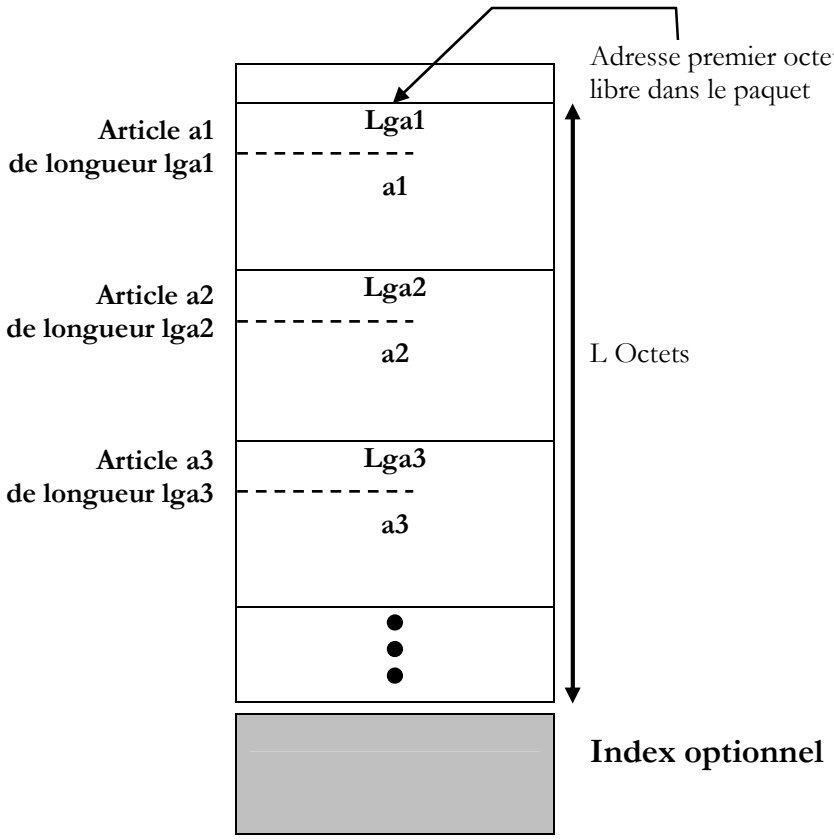


Fig.1.8. Structure interne d'un paquet

Dans un fichier haché statique, le problème de débordement se pose lorsqu'un paquet est plein, et pour gérer ce débordement, plusieurs méthodes ont été proposées [5] parmi lesquelles on trouve :

- L'adressage ouvert : qui consiste à placer l'article qui devrait aller dans un paquet plein dans le premier paquet suivant ayant de la place libre, il faut alors mémoriser tous les paquets dans lequel un paquet plein a débordé.
- Le chaînage : consiste à constituer un paquet logique par chaînage d'un paquet de débordement à un paquet plein.
- Le re-hachage : consiste à appliquer une deuxième fonction de hachage lorsqu'un paquet est plein, cette deuxième fonction conduit généralement à placer les articles dans des paquets de débordement.

II.6.1.2. hachage dynamique

La première organisation hachée dynamique a été proposée pour des tables en mémoire [58]. Puis plusieurs techniques fondées sur le même principe mais différentes ont été proposées pour étendre les possibilités du hachage à des fichiers dynamiques [59, 60, 61, 62, 63]. Le principe de base de ces différentes techniques est la digitalisation progressive de la fonction de hachage : la chaîne de bits résultat de l'application de la fonction de hachage à la clé est exploitée progressivement bit par bit au fur et à mesure des extensions du fichier. On trouve ici plusieurs méthodes comme :

A. le hachage extensible :

C'est une méthodes de hachage dynamique consistant à éclater un paquet plein et à mémoriser l'adresse des paquets dans un répertoire adressé directement par les (M+P) premiers bits de la fonction de hachage, où P est le nombre d'éclatements maximal subi par les paquets.

Un fichier haché extensible est donc structuré en deux niveaux, le répertoire et les paquets comme c'est montré sur la figure suivante :

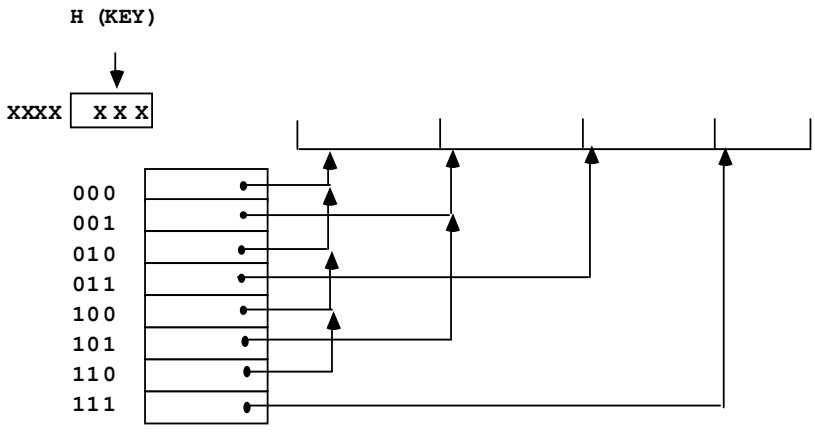


Fig.1.9. Répertoire et paquets d'un fichier haché extensible

.

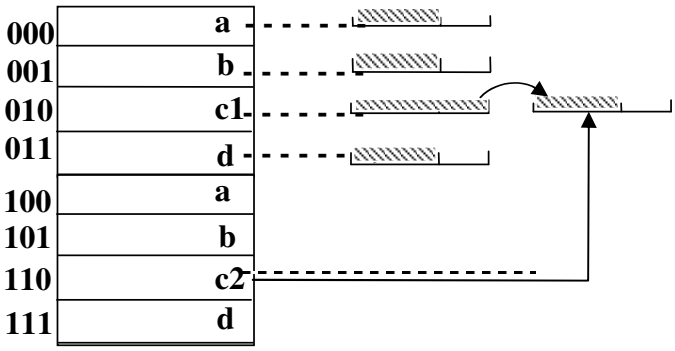


Fig.1.10. Eclatement d'un paquet dans un fichier haché extensible

B. le hachage linéaire :

Proposé par Dr. W.Litwin [63], il nécessite la gestion de débordement et consiste à éclater le paquet pointé par un pointeur courant quand un paquet est plein ; et à mémoriser le niveau d'éclatement du fichier afin de déterminer le nombre de bits de la fonction de hachage à appliquer avant et après le pointeur courant

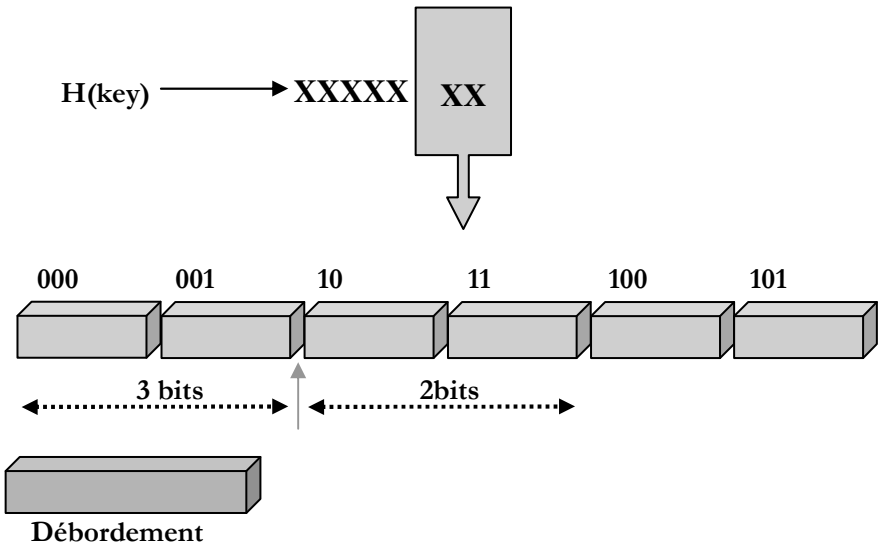


Fig.1.11. Fichier haché linéairement

II.6.1.3. Comparaison entre les méthodes d'accès par hachage

Type de hachage	Ecriture	Lecture	Débordement	Répertoire
Statique	2+d	1+d	oui	non
Extensible	2+r+e	1+r	non	oui
Linéaire	2+d+e	1+d	oui	non

- *d : débordement*
- *r : répertoire*
- *e : entête*

Tableau.1.1. *Tableau de comparaison entre les méthodes de hachage*

En résumé, les méthodes de hachage dynamique sont bien adaptées aux fichiers dynamiques, cependant, pas trop gros pour éviter les saturations de paquets trop nombreuses et les accroissement de la taille du catalogue.

Le grand problème de ces méthodes reste l'absence de possibilités efficaces d'accès ordonné pour un tri total ou pour des questions sur des plages de valeur. Telle est la limite essentielle, qui nécessite l'emploi d'autres méthodes telles que les méthodes d'accès indexés par exemple.

II.6.2. Organisation et méthodes d'accès indexées

A. Notion d'index

C'est une table (ou plusieurs tables) permettant d'associer à une clé d'article l'adresse relative de cet article.

La figure Fig.1.12, illustre cette notion d'index. Le fichier contient un ensemble d'articles, l'index est rangé en fin de fichier comme son dernier article. Il contient une entrée par article indiquant la clé de l'article et son adresse relative dans le fichier.

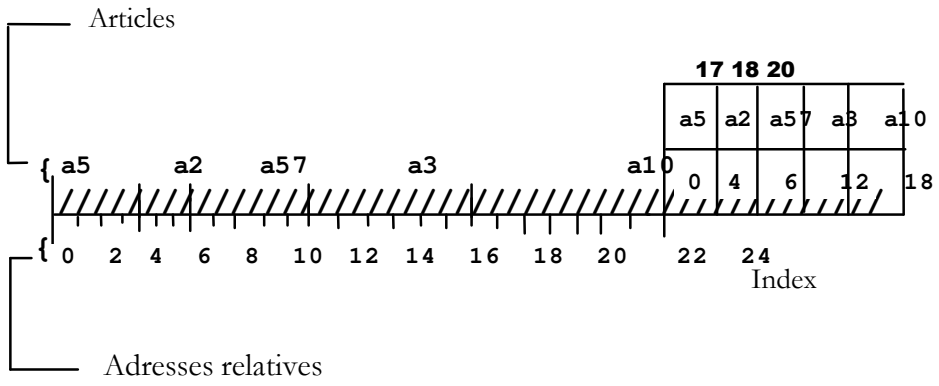


Fig.1.12. Exemple de fichier indexé

B. Types d'index

- **index dense/non dense**

Un index d'un fichier indexé peut contenir toutes les clés (de tous les articles) ou seulement certaines, on parle ici de la densité d'un index qui est le quotient du nombre de clés dans l'index sur le nombre d'articles du fichier, généralement, on distingue deux types d'index selon ce critère, qui sont :

- *Index Dense* : s'il contient toutes les clés (de tous les artides).
- *Index Non Dense* : s'il contient certaines clés seulement Fig.1.13.

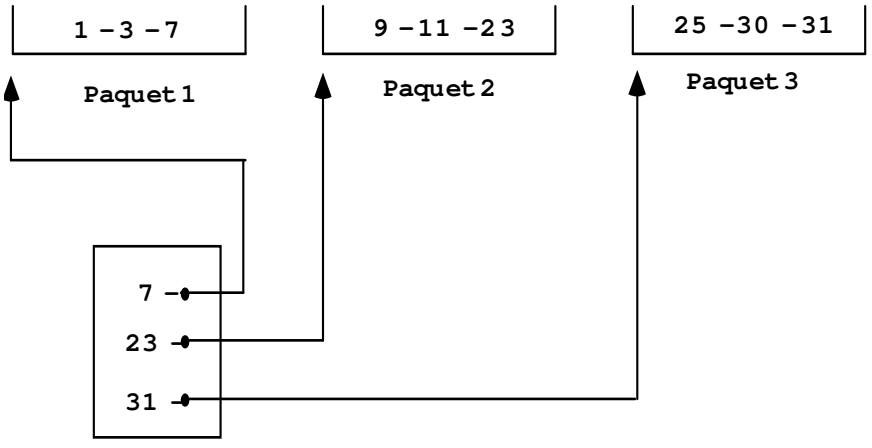


Fig.1.13. Exemple d'index non dense

Comme le fichier peut être trié ou non trié, et l'index dense ou non dense, trié ou non trié, diverses variantes sont théoriquement possibles. Deux méthodes sont

particulièrement intéressantes : le fichier séquentiel nom trié avec index trié dense, historiquement à la base de l'organisation IS3, et le fichier trié avec index non dense trié, sur lequel sont fondées des organisations telles ISAM, VSAM et UFAS, notons qu'il est impossible d'associer un index non dense à un fichier non trié [5, 45].

			Fichier	
			Trié	***Non trié***
INDEX	***Dense***	***Trié***	Possible	IS3
		Non trié		Possible
	Non dense	***Trié***	VSAM, ISAM, UFAS	
		Non trié		

*Tableau.*1.2. *Tableau typique des fichiers indexés*

- **index hiérarchisé**

Un index peut être vu comme un fichier de clés. Si l'index est grand, la recherche d'une clé dans l'index peut devenir très longue. Il est alors souhaitable de créer un index de l'index vu comme un fichier de clés. Un tel index est appelé index hiérarchisé, il est constitué de n niveaux, le niveau k étant un index trié divisé en paquets, possédant lui-même un index de niveau k+1, la clé de chaque entrée de ce dernier étant la plus grande du paquet.

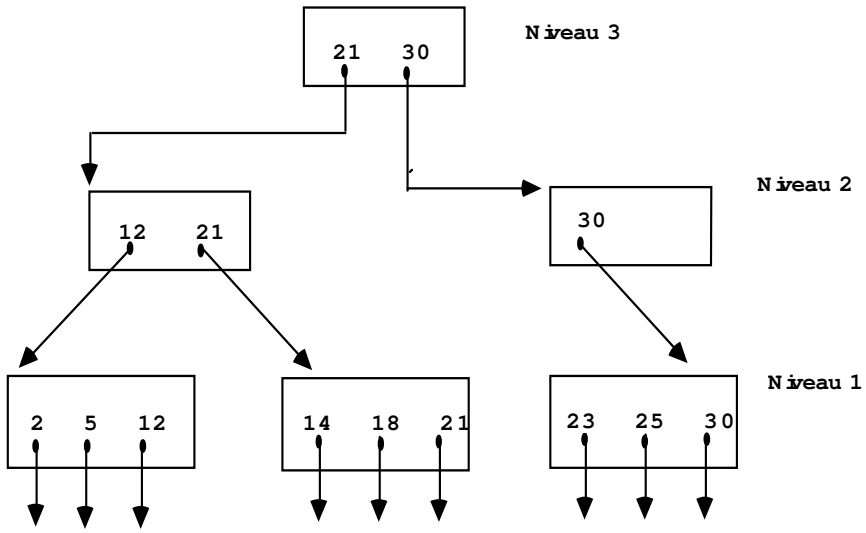


Fig.1.14.Exemple d'index hiérarchisé

II.6.2.1. Arbres B

Un arbre B d'ordre m est un arbre au sens de la théorie des graphes tel que :

1- toutes les feuilles sont au même niveau.

2- tout nœud non feuille a un nombre NF de fils tel que $m+1 \leq NF \leq 2m+1$ sauf la racine, qui a un nombre NFR de fils tel que $0 \leq NFR \leq 2m+1$.

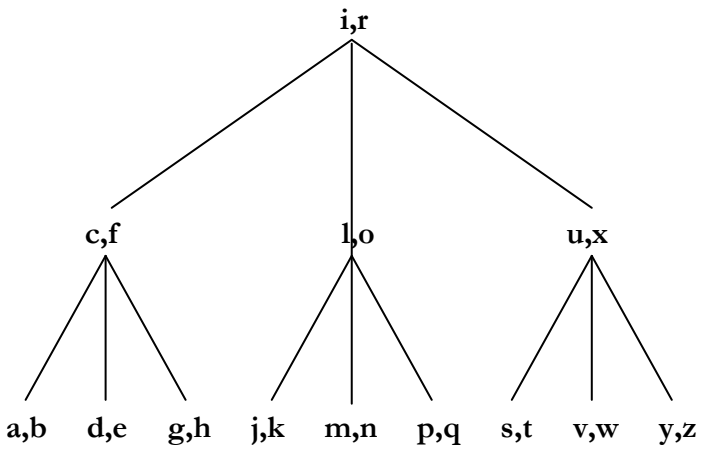


Fig.1.15. Arbres B d'ordre 2

Un arbre B peut être utilisé pour consulter un index hiérarchisé d'un fichier. Dans ce cas, les nœuds représentent des pages de l'index. Il contient des clés triées par ordre croissant et des pointeurs de deux types :

- Internes : désignent des fils et permettent de définir les branches de l'arbre.
- Externes : désignent des pages de données (en général, des adresses relatives d'articles).

La figure suivante précise la structure d'un nœud :

P0	x1 a1 P1	x2 a2 P2	……	xi ai Pi	……	xk ak Pk

- *Pi: Pointeur interne permettant de représenter l'arbre; les feuilles ne contiennent pas de pointeurs Pi;*
- *ai: Pointeur externe sur une page de données;*
- *xi: valeur de clé.*

Fig.1.16. Structure d'un nœud d'un arbre B

Voici un exemple d'index sous forme d'arbre B représenté par Fig.1.17. Cet arbre contient les valeurs de clé de 1 à 25. Les flèches entre les nœuds représentent les pointeurs internes, les autres flèches issues d'une clé représentent les pointeurs externes vers les articles.

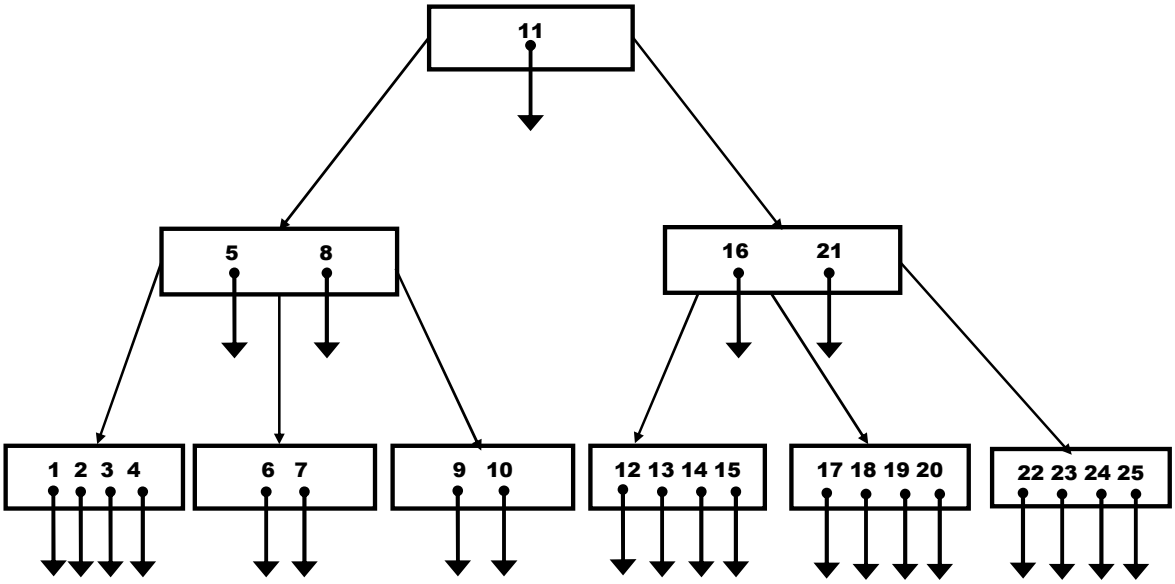


Fig.1.17. Exemple d'index sous forme d'arbre B

II.6.2.2. Arbres B+

Un arbre B+ est un arbre B dans lequel on répète les clés des nœuds :

- ascendants dans chaque nœud et on chaîne les nœuds.
- feuilles pour permettre un accès rapide en séquentiel trié.

Les arbres B+ sont utilisés pour gérer des index hiérarchisés :

en mettant toutes les clés des articles dans un arbre B+ et en pointant sur ces articles par des adresses relatives, ou

en rangeant les articles au plus bas niveau de l'arbre B+.

Voici un exemple d'index en arbre B+, il correspond à l'exemple précédent (Fig.1.17) d'index en arbre -B :

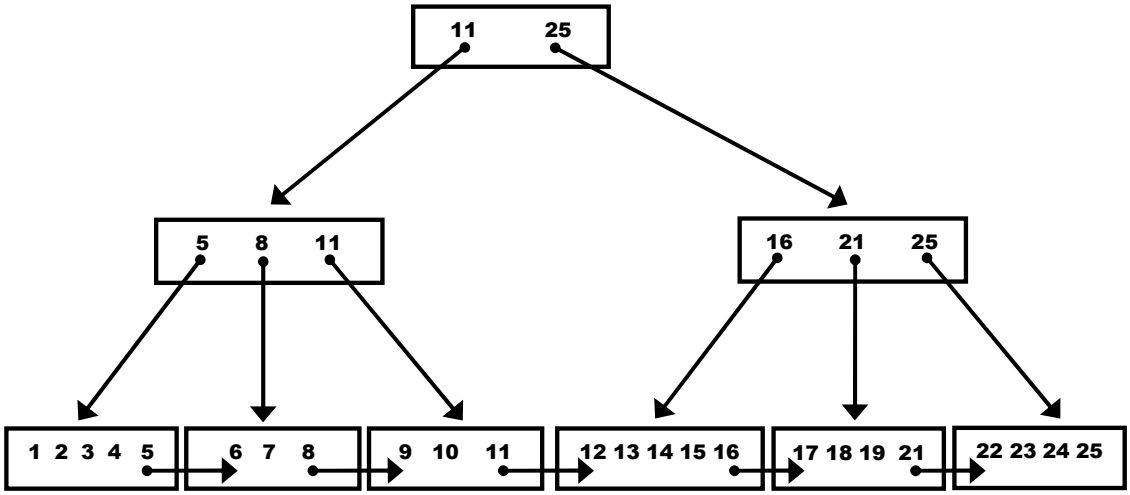


Fig.1.18. Exemple d'index sous forme d'arbre B+

Il existe plusieurs organisations indexées comme :

II.6.2.3. l'organisation indexée IS3

Dont le fichier est séquentiel non trié, d'index trié organisé sous la forme d'un arbre-B+.

II.6.2.4. L'organisation séquentielle indexée ISAM

Dont le fichier est trié d'index trié non dense composé d'une zone primaire et d'une zone de débordement ; une piste saturée dans une extension logique constituée par une liste d'articles en débordement.

II.6.2.5. L'organisation séquentielle indexée régulière VSAM

Ici le fichier est trié non dense dont l'ensemble fichier plus index est organisé sous forme d'un arbre B+.

II.6.3. Méthodes d'accès multi-attributs

En plus, il existe un ensemble de méthodes d'accès multi-attributs [5] qui permettent d'accéder rapidement à un groupe d'articles à partir des valeurs de plusieurs données, aucune d'entre elles n'étant en général discriminante. Ces méthodes utilisent ce qu'on appelle un index secondaire qui est un index sur une

donnée non discriminante, donnant pour chaque valeur de la donnée la liste des adresses d'articles ayant cette valeur. Parmi ces méthodes, on peut citer les suivantes :

II.6.3.1. hachage multi-attributs

C'est une méthode d'accès multidimensionnelle dans laquelle l'adresse d'un paquet d'articles est déterminée en appliquant des fonctions de hachage à différents champs d'un article. On trouve les deux types de hachage, le hachage multi-attributs statique et le hachage multi-attributs dynamique.

II.6.3.2. index bitmap

Index sur une donnée A constitué par une matrice de bits indiquant par un bit à 1 en position (i,j) la présence de la j^{ème} valeur possible de l'attribut indexé A pour l'article i du fichier, sinon son absence.

Données

Personne	Sport
Perrec	Athlétisme
Cantona	Foot
Virenque	Vélo
Leconte	Tennis
Barthez	Foot
Jalabert	Vélo
Pioline	Tennis

Index Bitmap

Athlétisme	Foot	Tennis	Vélo
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
0	1	0	0
0	0	0	1
0	0	1	0

Fig.1.19. Exemple d'index bitmap simple

Conclusion

Vue que le domaine des BD est aujourd'hui très riche, ses notions de base ne peuvent pas être délaisser, ainsi le présent chapitre a introduit les notions importantes de ce domaine, comme le modèle relationnel ainsi que les organisations et les méthodes d'accès aux fichiers représentant l'implémentation physique de base pour les relations d'une BDR.

Chapitre 2

Multiordinateurs & réseaux P2P

(peer to peer)

Introduction

Durant l'évolution technologique, plusieurs concepts ont été apparus, les systèmes distribués furent les premiers parmi eux, ce qui a conduit à l'apparition de ce qu'on appelle « les réseaux informatique ».

Dans leur progression, les réseaux informatiques sont diversifiés, tantôt on parle d'architecture client/serveur et tantôt on parle d'architecture paire à paire (ou peer to peer). Cette nouvelle configuration, apparue ces dernières années sous différentes appellations comme multiordinateurs, grille de calcul, ou tout simplement réseau de station de travail, fut la base de grands projets de recherche dans plusieurs domaines

Les réseaux peer to peer visent à réaliser un réseau complètement décentralisé par la connexion de sites individuels entre eux pour les faire communiquer, sauvegarder et transférer les données ou encore exécuter des calculs distribués. Ce genre de réseaux permet aussi d'assurer la scalabilité des données, leur redondance et leur disponibilité.

Très utilisés aujourd'hui, les réseaux peer to peer sont un type important dans le monde Internet, surtout lorsqu'il s'agit de partage de fichiers ou tout simplement de données entre les internautes. Pour le faire, plusieurs architectures et topologies ont été proposées et plusieurs plates-formes ont été réalisées pour les supporter.

Ce chapitre vient pour évoquer les différentes architectures des réseaux p2p et les multiples applications qui les utilisent.

I. Les systèmes informatiques

On peut classer les systèmes informatiques en deux grandes catégories : les systèmes centralisés et les systèmes distribués, comme c'est illustré par (Fig.2.1).

A leur tour, les systèmes distribués peuvent être subdivisés en deux catégories : le modèle client/serveur (plat ou hiérarchique) et le modèle pair à pair (pur ou hybride)

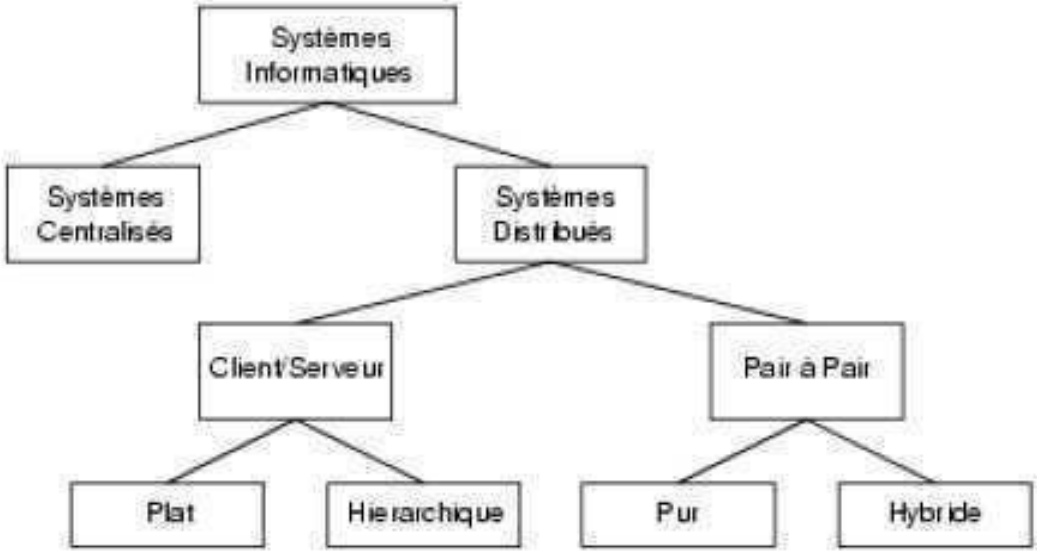


Fig.2.1. Classification des systèmes informatiques

I.1. Systèmes distribués

Un système distribué d'après M.Mosbah [1] est un ensemble d'entités autonomes de calcul (ordinateurs, PDA, …) interconnectées et qui peuvent communiquer entre-elles. D'une autre façon des processus indépendants sur des machines distinctes communiquant par échange de messages.

Les systèmes distribués utilisent souvent des machines parallèles. Selon le partage de la mémoire, on distingue deux types de machines parallèles :

I.1.1. Les machines fortement couplées : dans cette architecture, les processus partagent une mémoire commune.

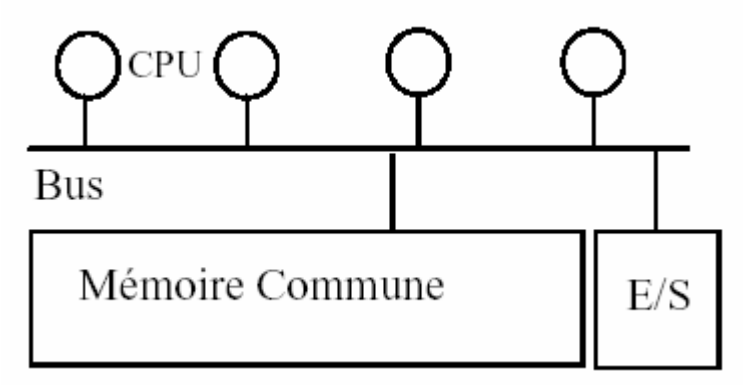


Fig.2.2. Machines fortement couplées

I.1.2. Les machines faiblement couplées : dans cette architecture, chaque processus dispose d'une mémoire locale pour lui et il peut la gérer individuellement.

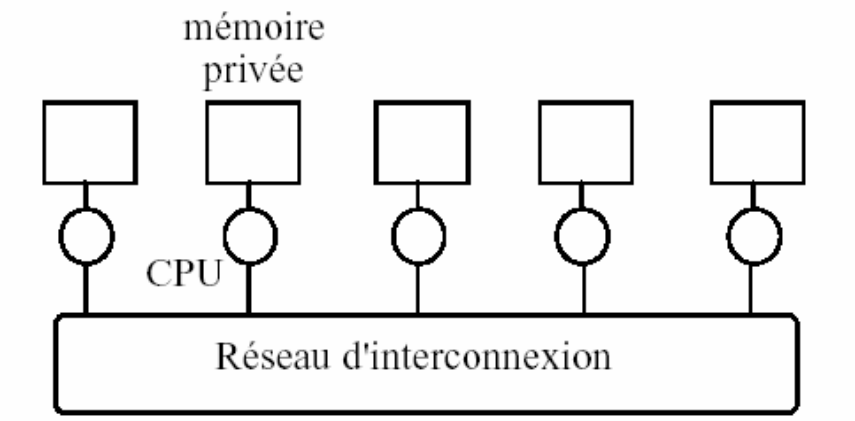


Fig.2.3. Machines faiblement couplées

II. Les multiordinateurs

Les multiordinateurs sont une nouvelle configuration de base qui a été introduite dans les entreprises, elle consiste en une collection d'ordinateurs (PCs, ou stations de travail autonomes) faiblement couplés (généralement reliés par un bus de type LAN, WAN, ou autre) et qui ne partagent pas de mémoire commune, comme c'est illustré sur Fig.2.4.

Cette configuration est devenue de plus en plus dominante à cause de ses points forts, car elle se base sur l'utilisation du matériel déjà existant sur le marché, elle offre des capacités cumulées de stockage et une puissance quasi-illimitée de calcul,

sans oublier le rapport coût / performance (moins chère et plus puissante qu'un gros système).

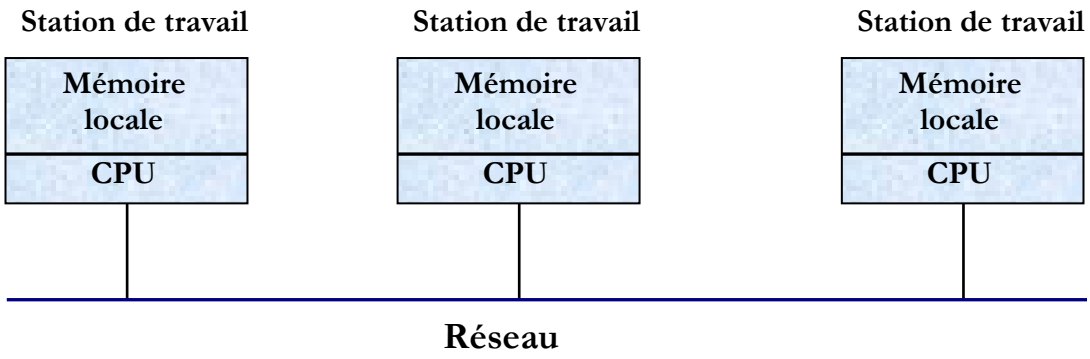


Fig.2.4. Configuration multiordinateur

III. Les réseaux informatiques

Un réseau informatique est un ensemble d'entités interconnectées les unes aux autres, et qui permet de partager des ressources matérielles (imprimantes;..) et immatérielles (fichiers;..) selon des règles bien précises appelées protocoles.

Les réseaux peuvent être catégorisés selon plusieurs critères comme : l'étendue géographique, la relation fonctionnelle entre les composants ou encore la topologie du réseau.

- *Par étendue géographique :* on trouve les types de réseaux suivants : PAN, LAN, MAN et WAN.
- *Par relation fonctionnelle entre les composants :* on trouve les architectures suivantes : client/serveur, architecture multi-tiers et paire à paire (peer to peer).
- *Par topologie :* on distingue plusieurs topologies comme : étoile, bus, anneau, grille, hypercube, arbre et bien d'autres.

III.1. Les systèmes client-serveur

Clients et serveurs sont des entités distinctes fonctionnant de concert sur un réseau pour accomplir une tâche. Ils communiquent par des requêtes client et des réponses serveur. Donc, dans cet environnement (Fig.2.5) les clients ne peuvent voir que le serveur (qui est une machine très puissante en terme de capacités d'entrées/sorties).

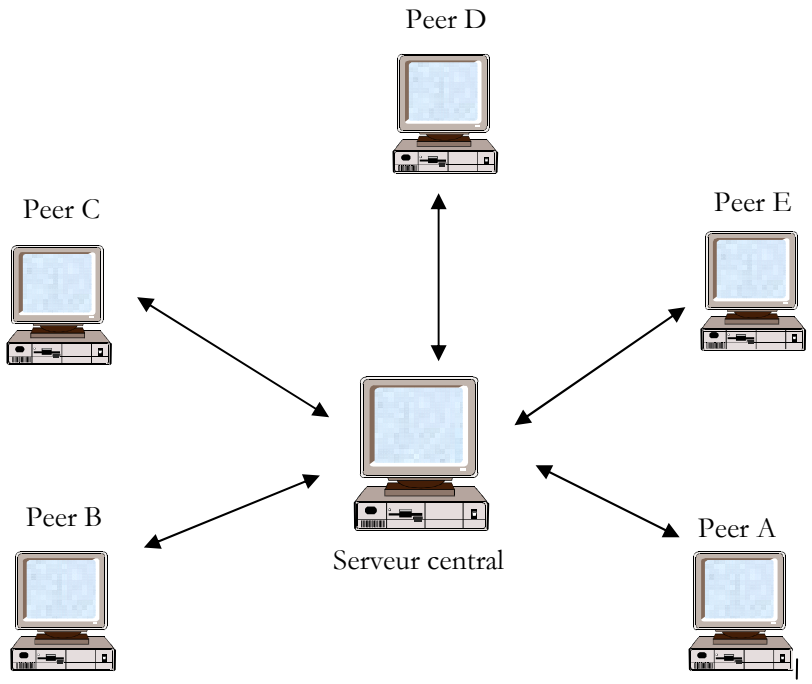


Fig.2.5. Architecture client-serveur

III.2. Evolution vers le peer to peer

Conceptuellement, le peer to peer est une alternative au modèle client-serveur, l'évolution vers le peer to peer a débuté par un jeune étudiant américain appelé 'Shawn Fanning' à la fin de l'année **1998** avec un logiciel permettant l'échange de fichiers musicaux nommé 'Napster' [72] ; avec lequel une liste des fichiers disponibles sur le réseau étant mise à disposition grâce à un serveur. Chacun pouvait

ainsi rechercher puis récupérer des fichiers. Seule la récupération d'objet était donc décentralisée.

En mars **2000** est arrivé Gnutella, premier système poste à poste totalement décentralisé, il permettait en effet la recherche et la récupération d'objets sans nécessiter de serveur. Le premier logiciel permettant son utilisation servait aussi au partage de fichier (sans être limité aux fichiers musicaux).Toutefois, la diffusion de ces fichiers étant souvent soumise à paiement de taxes (droits d'auteurs en général), qui n'étaient pas payés avec les systèmes P2P. Les tentatives des sociétés de production et de distribution pour interdire ou détruire ces systèmes P2P furent le premier moteur de l'évolution de ces systèmes [18].

C'est quoi un réseau p2p ?

Un réseau peer to peer, pair à pair, d'égal à égal ou encore poste à poste est un ensemble de postes autonomes connectés, communiquant directement entre eux et partageant leurs ressources sans avoir recours à une entité centrale d'administration (serveur central), c'est-à-dire tout les postes sont égaux, ils sont à la fois clients et serveurs d'où leur appellation de "servants".

III.2.1. Objectifs des réseaux peer to peer

Les objectifs des réseaux p2p sont nombreux et variés [16, 19] :

- Fiabilité et accroissement de l'autonomie à cause de l'absence d'une autorité centrale.
- Partage et réduction des coûts entre les différents peers.
- Anonymat pouvant être assuré par certaines applications.
- Agrégation des ressources et interopérabilité.
- Communication ad hoc et collaboration.
- …..etc.

Par rapport à l'architecture client/serveur, un réseau p2p continue à marcher, même si plusieurs participants quittent le réseau et il marche mieux en ajoutant d'autres nouveaux participants.

III.2.2. Caractéristiques des systèmes p2p

Parmi les caractéristiques principales des systèmes p2p, on peut trouver :

- l'auto organisation qui consiste en l'adaptation automatique du réseau aux départs/arrivées des nœuds et à leurs pannes.
- La communication symétrique dans les peers clients et serveurs.
- Le contrôle distribué à cause de l'absence d'un gestionnaire centralisé (ou point de contrôle).
- Un mode de communication standard (TCP & HTTP).
- Aucun peer ne possède une vue globale sur le système.
- La libre circulation des fichiers entre systèmes.
- …etc.

III.2.3. Topologies et architectures des p2p

Le réseau peer to peer possède une topologie virtuelle (overlay network) qui varie entre une infrastructure structurée, non structurée et hybride avec une variétés d'architectures, divisées généralement en trois grandes catégories : une architecture centralisée, une architecture décentralisée et une troisième dite hybride.

III.2.3.1. Classification selon le degré de centralisation

A. L'architecture centralisée

Ce type d'architecture illustré par la figure (Fig.2.6) se base sur l'utilisation d'un serveur central auquel se connectent les utilisateurs (peers), son rôle est de mettre en relation directe les clients connectés sur son réseau en s'appuyant sur une indexation centralisée de tous les répertoires et intitulés de fichiers partagés par ces clients.

Parmi les réseaux p2p appartenant à e type d'architecture, on trouve le célèbre réseau ***Napster*** qui utilise un modèle p2p à répertoire centralisé pour les données administratives et les index des fichiers MP3 téléchargeables par les peers.

Cette architecture possède un inconvénient remarquable, c'est que dans le cas où le serveur tombe en panne ou se trouve encombré, le réseau se dématérialise, et pour résoudre ce problème, le serveur central est remplacé par un anneau de serveurs, ainsi la chute des réseaux en cas de panne peut être évitée, l'exemple du réseau le plus connu qui utilise ce type d'architecture est le réseau ***eDonkey.***

Dans cette architecture, chaque serveur a la possibilité d'accéder aux informations des clients connectés sur les autres serveurs et ceci dévoile un autre désavantage qui est l'absence de l'anonymat ce qui facilite les attaques et met en cause la sécurité des peers.

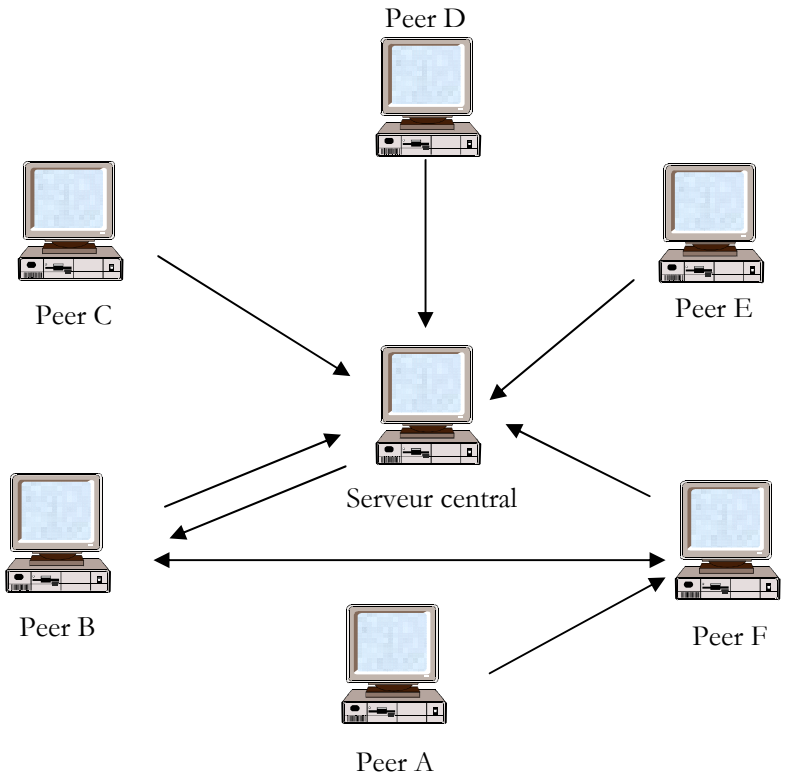


Fig.2.6. Réseau p2p centralisé

B. L'architecture décentralisée

A l'inverse de l'architecture centralisée, l'architecture décentralisée préserve l'anonymat des peers, et ceci à cause de l'absence du serveur central, ainsi la relation

entre ces peers est directe c'est pour cela que ce type de réseau est appelé pur peer to peer. Dans cette architecture illustrée par la figure (Fig.2.7), tous les peers ont le même niveau c'est-à-dire ils peuvent tous jouer le rôle du serveur et celui du client en même temps.

Ce type d'architectures est utilisé dans différents réseaux comme ***Gnutella, Freenet, Chord, CAN, Tapestry, Symphony***….et bien d'autres [18].

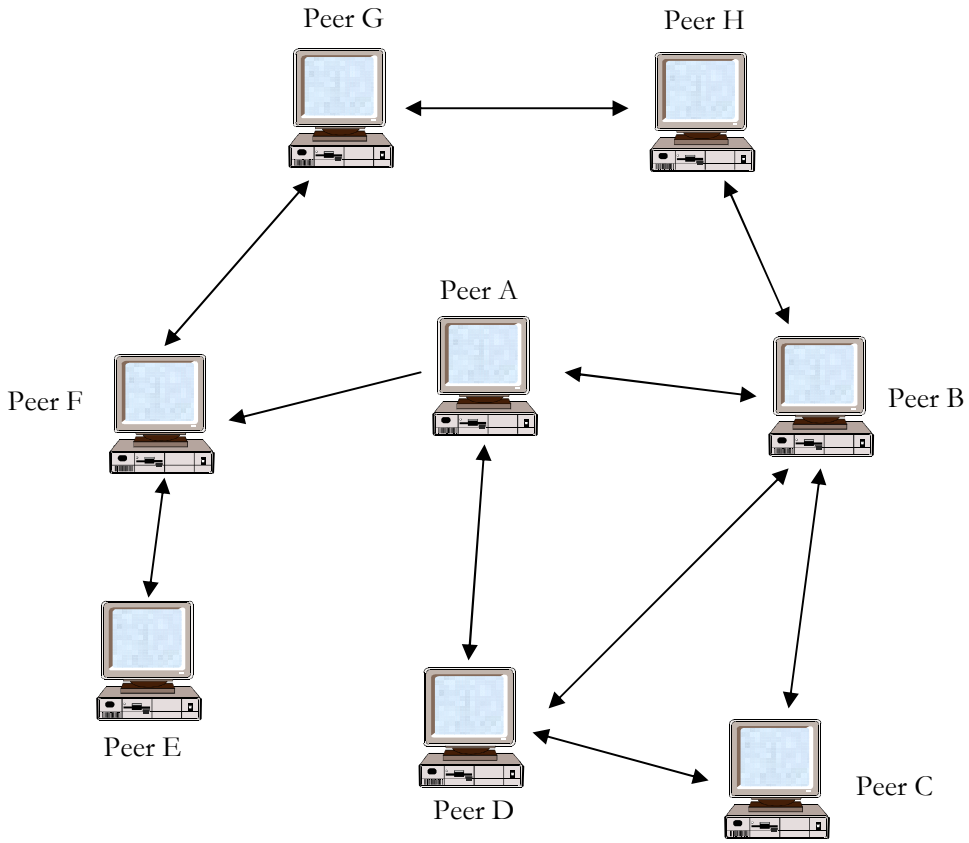


Fig.2.7. Réseau p2p décentralisé

C. L'architecture hybride

Ce modèle combine entre l'architecture centralisée et l'architecture décentralisée comme cela est illustré sur la figure (Fig.2.8), dans ce type d'architecture, on trouve des machines particulières qui jouent le rôle du serveur central appelée "super-node" ou "super-peer", elles disposent d'une bonne bande passante, à ce moment les

nœuds ou peers ayant une faible bande passante sont connectés aux super-peers pourvu d'un index qui retient les méta-informations des peers connectés.

Comme exemple des réseaux utilisant ce type d'architecture on a : ***KaZaA, FastTrack et Gnutella2***.

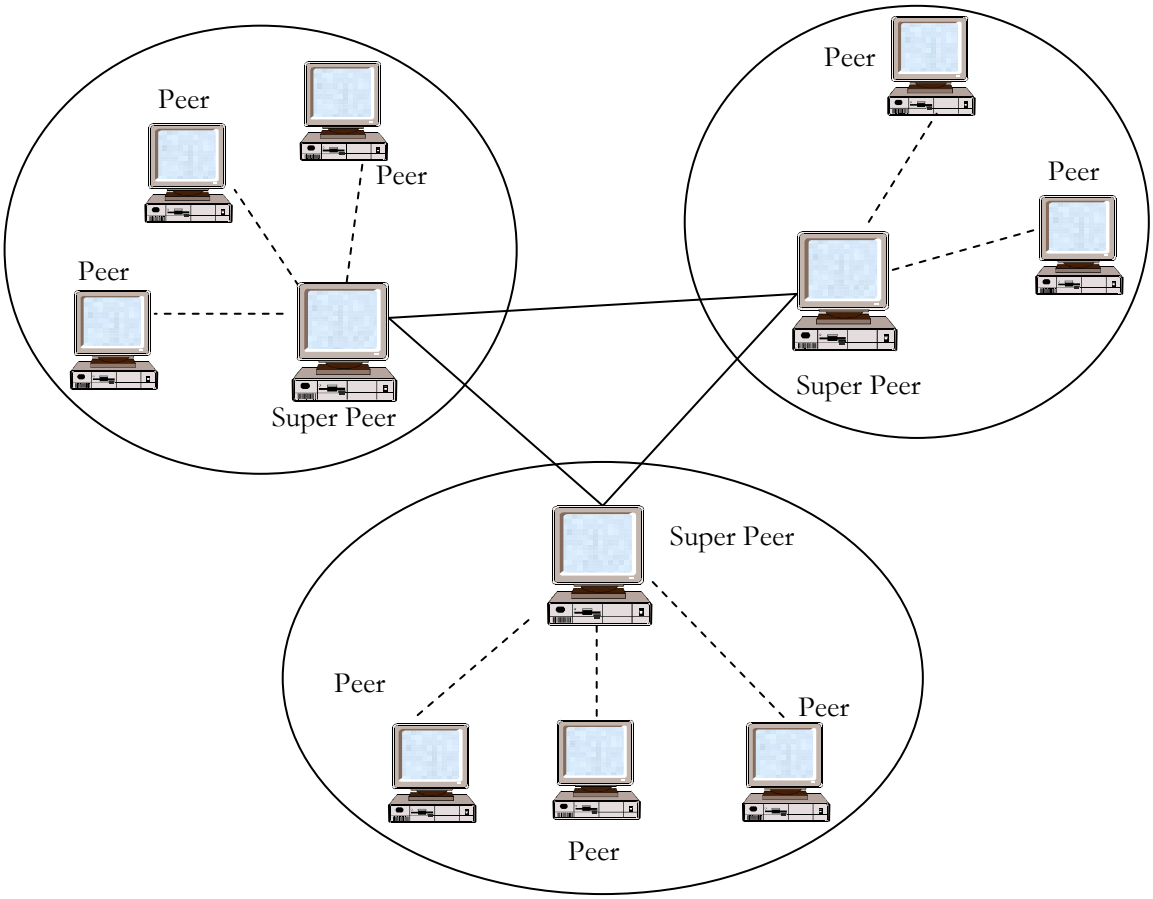


Fig.2.8. Réseau p2p hybride

Amélioration

Le choix d'un super-peer "à la tête" d'un groupe introduit de la sensibilité aux défaillances : si le super-peer n'est plus joignable, tous ses clients sont coupés du réseau. Une des améliorations possibles est donc de choisir parmi K super-peers au lieu d'un seul.

Une restriction est que les super-peers soient partenaires. Dans ce cas, chaque partenaire est relié à chaque client, et possède un index de toutes leurs ressources.

III.2.3.2. Classification selon l'organisation

A. Les réseaux p2p structurés

Ce type de réseau est basé sur l'utilisation d'une table de hachage distribuée 'DHT', c'est un genre d'index distribué entre les peers selon une géométrie particulière pour permettre la localisation des données cherchées. Parmi ces géométries on trouve : un anneau (cas de ***Chord***), un espace d-dimensionnelle (cas de ***CAN***), un arbre binaire (cas de ***P-Grid***) [28]… et bien d'autres.

Ce type d'architecture supporte les évolutions du réseau c'est-à-dire les peers peuvent se connecter et se déconnecter, et si cela arrive très fréquemment, le problème du coût de maintien d'index est posé.

B. Les réseaux p2p non structurés

Ce type de réseau assure l'autonomie des peers, car ces derniers ne sont pas structurés selon une géométrie précise et donc il s'avère nécessaire de localiser les données recherchées, pour cela, les peers utilisent ce qu'on appelle les requêtes de propagation (ou d'inondation), en plus des stratégies plus sophistiquées comme les cheminements aléatoires (random walks) et les indices de routage (routing indices) [21], ou encore des stratégies d'organisation logique du réseau (routage collaboratif); pour rapprocher les peers en fonction de leur contenu ou leur comportement [28].

Parmi les exemples de réseaux représentant ce type d'architecture on a : Napster, Publius, Gnutella, Kazaa, Edutella, FreeHaven… et bien d'autres.

C. Les réseaux p2p hybrides

Ce type de réseau combine entre une topologie structurée et une topologie non structurée, il tient en compte de l'hétérogénéité des peers en terme de capacités [28], c'est à dire, il utilise la notion de super-peers comme c'est déjà présenté.

On peut résumer les architectures et leurs topologies à l'aide d'un tableau comportant l'ensemble des applications p2p connues selon [21, 30].

Architecture & Topologie	Centralisée	Décentralisée	Hybride
Structurée		Chord, CAN, Tapestry, Pastry	
Non structurée	Napster, Publius	Gnutella, FreeHaven	Kazza, Morpheus, Gnutella, Edutella
Hybride		Freenet, OceanStore, Mnemosyne, Scan, PAST, Tarzan, Kademlia	

Tableau 2.1. Exemple des réseaux p2p selon leurs architectures et leurs topologies

III.2.4. Applications des technologies p2p

Le paradigme du p2p a été implémenté dans différents types de domaines, les plus distingués sont :

III.2.4.1. Le calcul distribué

Le but dans ces applications est de tirer profit des ressources de calcul des peers existants tout en divisant le gros calcul en petites unités de calcul plus ou moins indépendantes. Donc chaque peer s'occupe d'une ou de plusieurs unités et pour effectuer ce calcul, les peers doivent communiquer et échanger les données nécessaires.

Plusieurs projets utilisant ce type d'application ont été apparus, citons à titre d'exemple : senti@home, genome@home [21], le projet Decryptron en génétique [22], le Global Grid Forum (GGF) [19], le projet Data Grid [23] qui vise à développer des infrastructures de calcul distribué,… et bien d'autres.

III.2.4.2. Communication et collaboration

Ce type d'applications vise à faciliter la communication directe et la collaboration entre les utilisateurs (peers) en temps réel sans l'utilisation d'un coordinateur central. Ces tâches (communication & collaboration) peuvent être opérées par plusieurs applications [16, 19, 21, 22] comme : la messagerie instantanée (ICQ, NetMeeting, AOL, Yahoo, MSN, Jabber;..), la voix sur IP (Skype), les jeux en réseau (Doom), et les applications permettant de travailler sur des projets distribués (Groove, Magi, PowerPoint distribué, CAP P2P, Kamer,..).

III.2.4.3. Les services Internet

Un ensemble d'applications basé sur l'infrastructure p2p a été émergé pour supporter une variété de services Internet comme la voix sur IP (Skype [22] & GooleTalk [24]), les moteurs de recherche (InfraSearch) [16], les applications sécurisées (comme l'e-commerce) [25], … et bien d'autres.

III.2.4.4. Diffusion de contenu

Ce type d'applications est désigné à faire partager l'ensemble des informations et des données entre les utilisateurs, ces applications varient entre de simples systèmes d'échange de contenu (Napster, Gnutella, Morpheus, Freenet, Kaaza, BitTorent [19]), et d'un ensemble d'applications permettant la distribution des fichiers (PAST, OceanStore, CAN, Chord [19], Farsite [26]).

III.2.4.5. Bases de données distribuées

Des travaux considérables ont été réalisés pour concevoir des systèmes de bases de données distribuées basés sur une infrastructure p2p. Parmi lesquels on trouve le modèle relationnel local (LRM) proposé par Bernstein [27], le projet du moteur de requête scalable et distribué PIER [21], le système Piazza qui fournie une infrastructure pour construire des applications de web sémantique, le projet Edutella qui utilise le standard RDF, … et bien d'autres.

III.2.4.6. Plates-formes

Elles proposent des environnements et des infrastructures génériques pour permettre de développer des applications p2p offrant les fonctions de base comme la gestion des peers, le nommage, la découverte des ressources, la communication entre les peers, la sécurité, l'agrégation des ressources…etc. Ainsi, on trouve la plate-forme Jxta proposée par Sun, la plate-forme .NET proposée par MicroSoft, la plate-forme Anthill,….., et bien d'autre.

Conclusion

Dans ce chapitre, on vient d'expliquer les différentes architectures des systèmes p2p pouvant représenter des multiordinateurs, on vient aussi d'exposer les multiples applications pouvant exploiter ce genre de réseaux.

Chapitre 3

Les SDDS

(Structures de Données Distribuées et Scalables)

Introduction

Aujourd'hui avec les progrès technologiques, plusieurs concepts ont évolués et d'autres nouveaux sont apparus, notamment les architectures constituées d'ordinateurs géographiquement distribués. Ces nouveaux systèmes sont désignés par une variété d'appellations comme multiordinateurs, réseau de stations de travail, grille de calcul.., Etc.

L'objectif principal pour lequel ces systèmes ont vu le jour est l'exploitation maximale des ressources cumulées de stockage et de traitement ; ce que les structures de données classiques n'ont pu réaliser. Pour cela une nouvelle famille de structure de données a été introduite en ***1993*** par Dr. Witold Litwin, dénommée SDDS (structure de données distribuées et scalables), il s'agit de fichiers résidents en mémoire centrale distribuée (DRAM).

Cette nouvelle structure supporte le traitement parallèle et assure un temps d'accès aux données beaucoup plus rapide que celui des fichiers classiques stockés sur disque, en outre elle assure une capacité de stockage potentiellement illimitée.

Un fichier SDDS débute sur un site et peut s'étendre dynamiquement sur plusieurs sites suite à des opérations d'insertion ; pour cela les algorithmes d'adressage d'une SDDS ont été conçus spécifiquement pour être scalables (la scalabilité ici consiste à maintenir les performances d'un système quand le volume de données stockées augmente).

Depuis l'apparition de cette nouvelle structure, plusieurs types de SDDS ont été proposés par Litwin, Neimat et Schneider, les plus connus sont celles basées sur le hachage linéaire (LH*) et celles utilisant le partitionnement par intervalle (RP*).

Dans ce qui suit, nous expliquons ces nouvelles structures de données tout en invoquant leurs deux grandes classes (les LH* & les RP*) avec des exemples de leurs dérivées.

I. Les systèmes de fichiers distribués

Un système de fichiers distribués simule les fonctions d'un système de fichier local en environnement distribué, il dispose en plus de certaines fonctionnalités spécifiques comme la tolérance aux pannes, la réplication, le partage, …Etc, et il est géré par un système de gestion de fichiers distribué (SGFD).

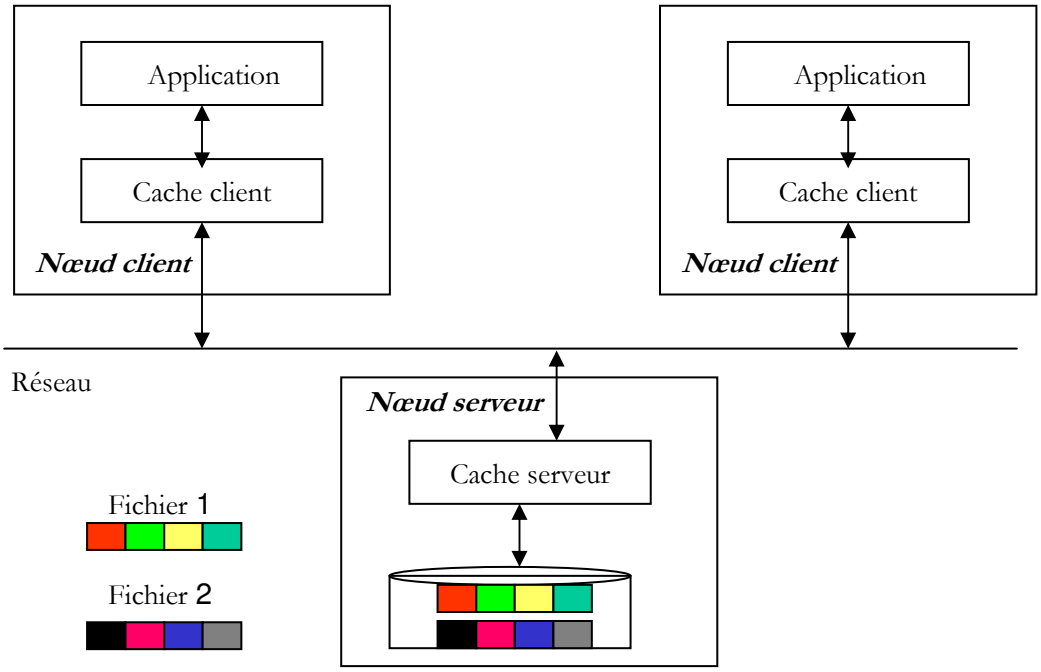


Fig.3.1. Architecture d'un SGFD

II. Les structures de données distribuées et scalables (SDDS):

Les SDDS sont une nouvelle classe de structure de données conçues spécialement pour les multiordinateurs. Elles stockent les données sur des sites appelés serveurs qui sont accédés par d'autres sites appelés clients.

Un fichier SDDS est manipulé par des sites clients, chaque client possède son propre schéma d'adressage appelé "image" qui lui permet d'accéder au bon serveur. Le fichier SDDS débute sur un seul site serveur et peut être étendu par insertion à un nombre quelconque de sites, ceci rend sa capacité de stockage potentiellement illimitée.

II.1. Règles de base des SDDS:

Les SDDS sont régies par un ensemble de règles [1, 6, 7, 8] qui sont:

- L'absence de répertoire central d'accès, ce qui favorise l'élimination des goulots d'étranglement, et augmente les performances d'accès du système.
- Les fichiers SDDS s'étendent sur plusieurs serveurs par des opérations d'éclatement et se rétrécissent par des fusions de cases (une case est une mémoire d'un serveur stockant les données de la SDDS) et ceci est fait d'une façon dynamique et transparente pour l'application.
- Chaque client gère sa propre image du fichier, car les mises à jour de la structure d'une SDDS ne sont pas envoyées aux clients dune façon synchrone. Un client peut avoir une image incorrecte, ce qui provoquerait une erreur d'adressage de sa part.
- Un serveur est capable de détecter les erreurs d'adressage produites, et d'acheminer les requêtes - provenant du client - vers le serveur adéquat, ainsi ce dernier envoie un message d'ajustement d'image (IAM) au client qui ajuste son image pour éviter de refaire la même erreur une autre fois.

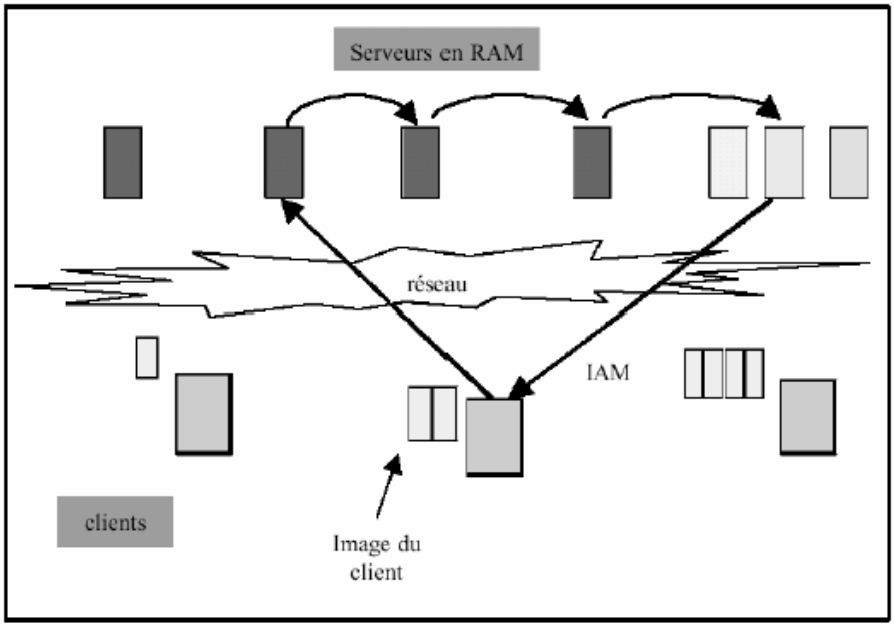


Fig.3.2. Schéma d'une structure de données distribuées

II.2. Les contraintes des SDDS:

Dans [2] Dr. Litwin a indiqué deux contraintes

- Si une SDDS n'évolue plus, alors les IAMs (messages d'ajustement d'image) font converger toute image d'un client vers celle actuelle.
- L'ensemble des renvois à la suite d'une erreur d'adressage ne se fait qu'en quelques messages.

II.3. Les caractéristiques des SDDS:

Les SDDS disposent d'un ensemble de caractéristiques parmi lesquelles :

II.3.1. la scalabilité:

Elle consiste à prendre en charge n'importe quelle quantité de données et à maintenir les performances quand le volume de données stockées varie [3], c'est-à-dire il n'y a pas de limites théoriques de capacité taille, pas de réorganisation totale de la structure et en pratique le temps d'accès aux données est plus ou moins constant.

II.3.2. la distribution:

Souvent lorsque la quantité des données augmente, un seul site devient incapable de la retenir et de la manipuler quelles que soient ses capacités de stockage et de calcul, alors, il devient nécessaire dans ce cas de distribuer ces données sur d'autres sites. Généralement, deux types de distributions peuvent être utilisés, le premier consiste en la distribution par hachage et le deuxième consiste en la distribution par intervalle. (Ils seront expliqués ci-après).

II.3.3. la disponibilité:

Consiste à assurer l'existence des données quelles que soient les conditions (même en cas de panne d'un ou de plusieurs sites serveurs).

II.4. Classification des SDDS:

Les SDDS sont regroupées en deux grandes catégories : des SDDS basées sur les algorithmes de hachage comme les LH* et des SDDS pour les fichiers ordonnés (par rapport à la clé primaire des enregistrements) comme les RP*. Fig.3.3 présente les SDDS les plus connues :

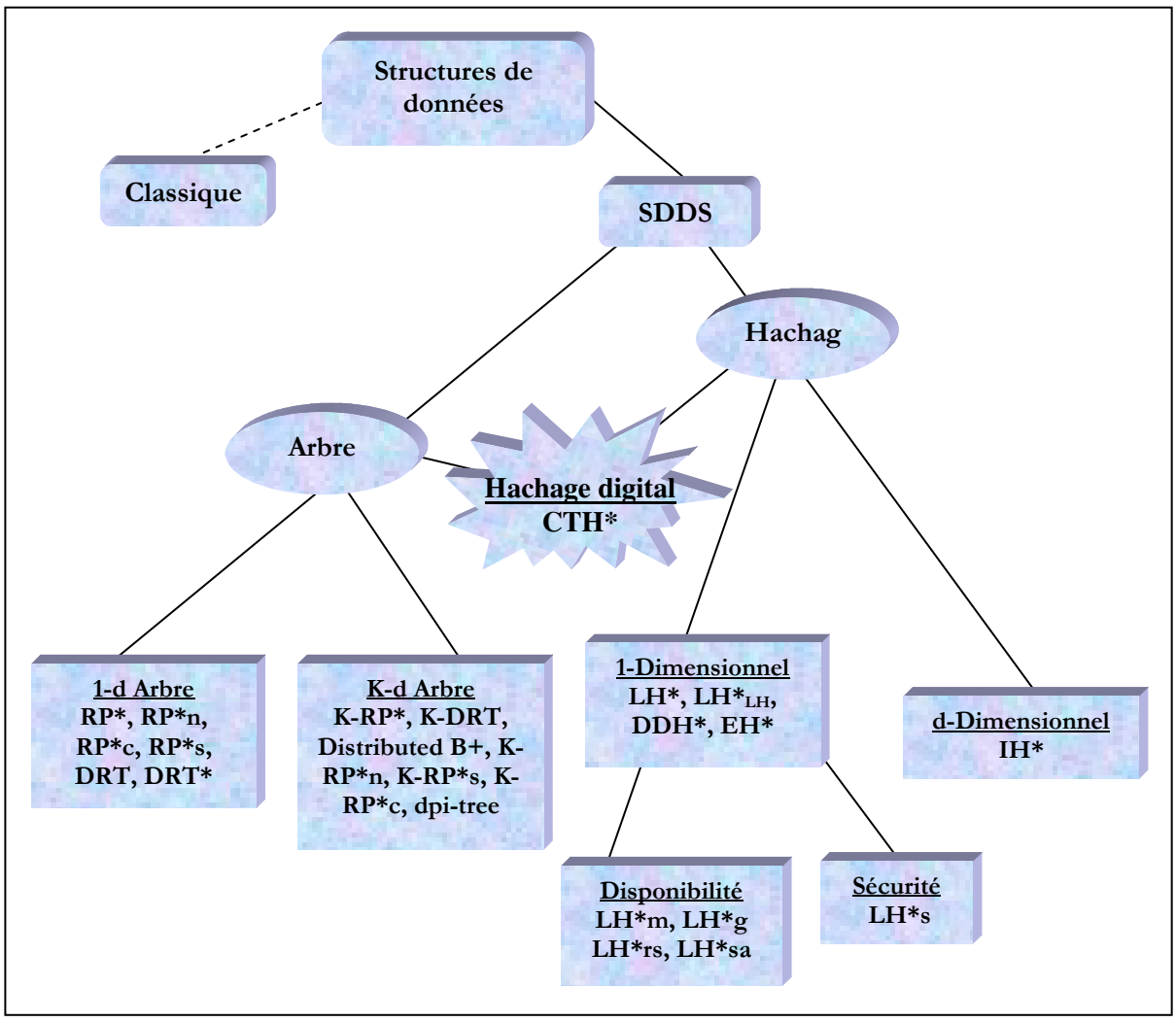


Fig.3.3. Classification des SDDS les plus connues

II.5. Les stratégies de distribution des données:

On distingue trois stratégies essentielles, qui sont :

- La distribution par donneur de cartes.
- La distribution par hachage.

- la distribution par intervalle.

D'abord quels sont les buts de la distribution des données?

Il faut dire que les objectifs de la distribution de données sont multiples [4] :

- Limiter le transfert d'information en répartissant les données là où elles sont le plus utilisées.
- Accroître les performances par la répartition de la charge de travail sur plusieurs unités de traitement opérant en parallèle.
- Augmenter la fiabilité en faisant effectuer le même traitement par plusieurs ordinateurs et en dupliquant les données sur plusieurs sites.
- Etendre la disponibilité des informations en les dupliquant sur plusieurs sites.
- Faire croître en souplesse une BD.
- Fusionner en douceur deux systèmes d'information en faisant coopérer leurs BD.
- …Etc.

II.5.1. La distribution par donneur de cartes

C'est la plus simple des stratégies de distribution des données, elle consiste à allouer au **i**ème article (ou unité de fragmentation) le nœud numéro (i mod N) ; avec ***N***: le nombre des nœuds existants. Cette stratégie garantit un équilibre optimum de charge entre les nœuds et convient bien aux requêtes nécessitant un parcours séquentiel, et permet de les paralléliser [7].

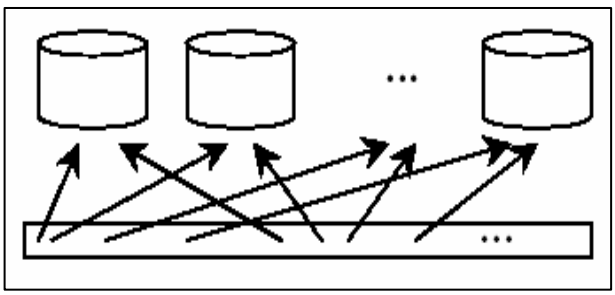


Fig.3.4. Illustration de la distribution par donneur de cartes

II.5.2. La distribution par hachage

Les organisations et méthodes d'accès par hachage sont basées sur l'utilisation d'une fonction de calcul nommée fonction de hachage qui, appliquée à la clé, détermine l'adresse relative d'une zone appelée paquet (bucket en anglais) dans laquelle est placé l'article [5].

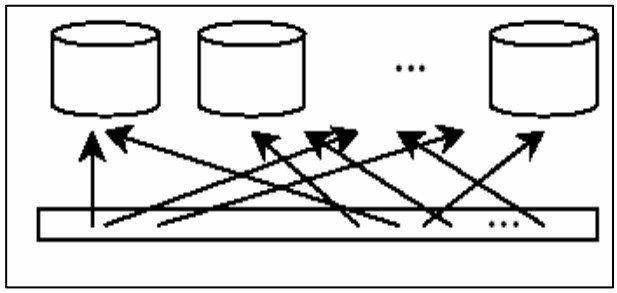


Fig.3.5. Illustration de la distribution par hachage

II.5.3. la distribution par intervalle:

En plus de la distribution par hachage, une autre distribution est utilisée, c'est celle connue par la distribution par intervalle qui préserve l'ordre des enregistrements dans un fichier.

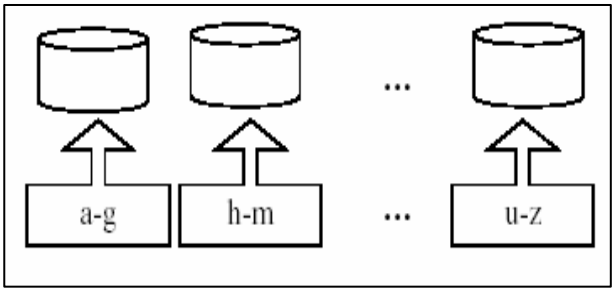


Fig.3.6. Illustration de la distribution par intervalle

II.6. Les SDDS basées sur le hachage

Les SDDS basées sur le hachage constituent une extension des schémas de hachage classique sur les multiordinateurs, on trouve parmi les SDDS de cette catégorie les SDDS suivantes :

- LH* (Distributed Linear Hashing) et ses dérivées.
- DDH* (Distributed Dynamique Hashing).
- EH* (Distributed Extensible Hashing).
- IH* (Distributed Interpolation-based Enashing).
- … Et bien d'autres.

II.6.1. Rappel sur le hachage linéaire

La méthode du hachage linéaire introduite par Dr.Litwin en ***1980*** [14] est définit selon Gardarin [5] par une méthode de hachage dynamique nécessitant la gestion de débordement et consistant à:

- Eclater le paquet pointé par un pointeur (n) courant quand un paquet est plein.
- Mémoriser le niveau d'éclatement (i) du fichier afin de déterminer le nombre de bits de la fonction de hachage à appliquer avant et après le pointeur courant.

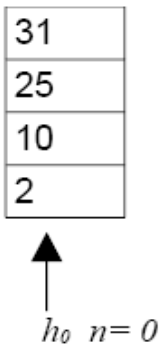
Dans cette méthode les fichiers sont organisés en paquets et ces paquets sont adressables à l'aide d'une fonction de hachage $h_i(C) = C \bmod n * 2^i$ avec :

hi : la fonction de hachage.

i = 0,1,2,3,……

n : le nombre initial des paquets.

Lorsque le paquet est débordé, une opération d'éclatement est effectuée, un exemple simple peut expliquer cette opération [12] :



n = 0
i = 0

Pour insérer un enregistrement dans un paquet, nous avons besoin de manipuler les deux pointeurs (i) et (n).

Considérons qu'un paquet possède la capacité de retenir quatre enregistrements, les quatre espaces de ces enregistrements sont calculés en utilisant l'algorithme d'adressage suivant:

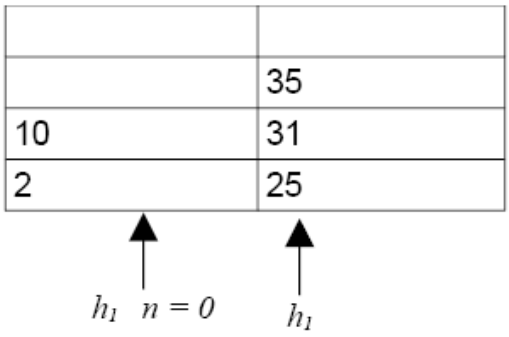
Algorithme d'adressage d'un enregistrement de clé C:

a : =h_i(C);

si a<n alors

a : = h_{i+1}(C);

L'insertion d'un nouvel enregistrement (35) cause un éclatement.



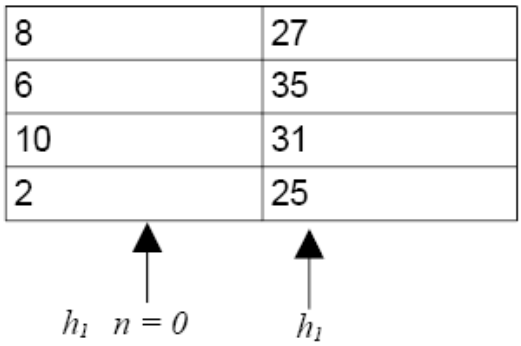
i = 1
n = 0
h_1 *(C) = C mod 2*

h1: indique la fonction de hachage utilisée dans le paquet.

n=0: indique le paquet le plus à gauche possédant la plus petite valeur de la fonction de hachage.

i=1: indique la plus petite valeur de la fonction de hachage.

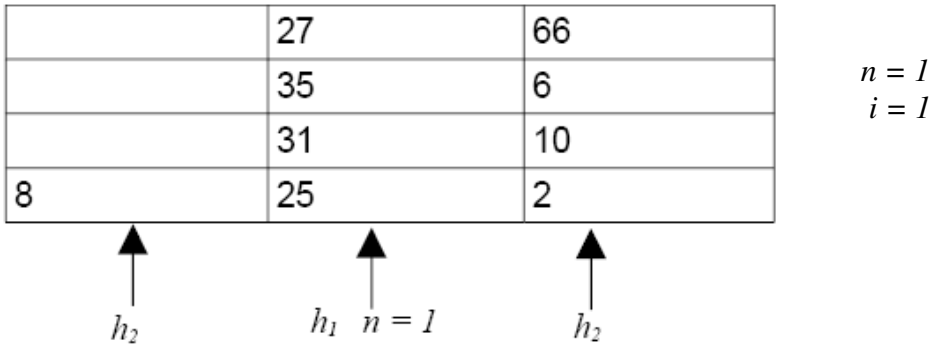
L'ajout de trois autres enregistrements (27), (8) et (6) donne le résultat suivant:



h_1 *(C) = C mod 2*

Lorsque n =1

Si un autre enregistrement est ajouté (66), il sera transféré au premier paquet qui correspond à la fonction de hachage, et si le paquet est débordé, il y aura une opération d'éclatement.



À ce moment, les enregistrements (2), (6) et (10) seront transférés au deuxième paquet, et les pointeurs seront réinitialisés:

II.6.2. Présentation des SDDS LH*:

La SDDS LH* (Fig.3.7) est la première SDDS qui a été proposée par Litwin, Neimat et Schneider [7], c'est la plus connue des SDDS existantes, elle est basée sur le hachage linéaire.

Plusieurs variantes ont été proposées dans ce type de SDDS (exemple : LH*$_{LH}$, LH*m, LH*g, LH*s, LH*sa, LH*rs), elles visent surtout à garantir la haute disponibilité des données et elles font appel à des principes de récupération de données, tel que : le miroitage (miroring), la fragmentation (striping), le groupement d'enregistrements (grouping) et le Reed-Solomon Code.

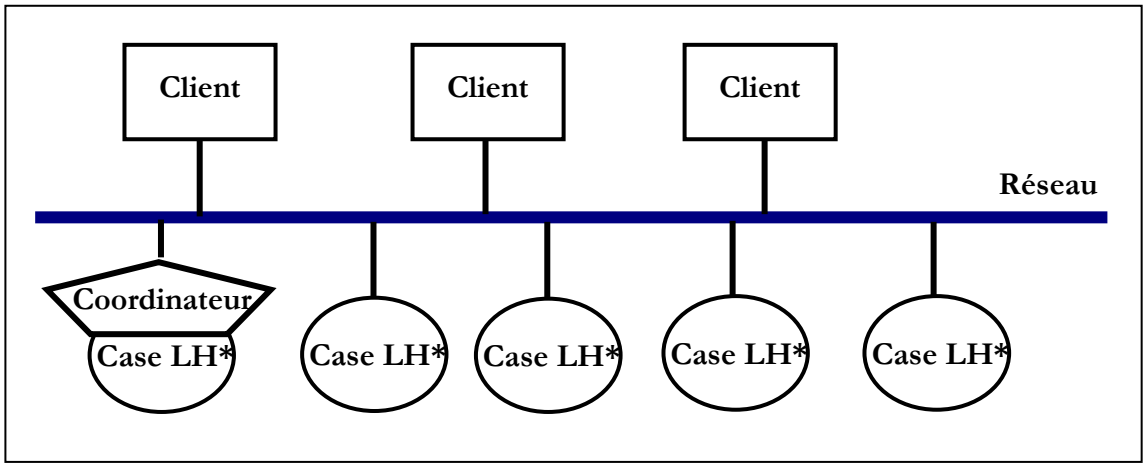


*Fig.3.7. Illustration de la SDDS LH**

II.6.3. Les principes des SDDS LH*:

Un fichier LH* est un ensemble d'enregistrements identifié par une clé primaire, ces enregistrements sont stockés dans des paquets (ou cases) numérotés qui ont une capacité **b** (b>>1), le fichier commence avec le paquet 0, il peut s'étendre avec des insertions ou se rétréci avec des fusions de cases.

Les éclatements sont déclenchés par des débordements de paquets, chaque éclatement déplace presque la moitié des enregistrements vers un autre serveur.

Dans LH*, le paquet débordé rapporte le débordement à un site dédié appelé coordinateur qui initie la procédure d'éclatement de la case n. pour effectuer l'adressage d'un enregistrement, une fonction de hachage $\mathbf{h_i\ (C) = C \bmod N * 2^i}$, i=0,1,… est appliquée sur la clé de cet enregistrement (comme le cas du hachage linéaire classique).

Algorithme d'adressage d'un enregistrement de clé C:

a : =h_i(C);

si *a*<*n* alors

a : = h_{i+1}(C);

L'éclatement résulte du remplacement de la fonction (h_i) actuellement utilisée pour le paquet n par la fonction (h_{i+1}), la valeur i est le niveau du paquet.

A n'importe quel moment, un fichier LH* peut avoir des paquets avec seulement un niveau i ou un niveau i+1.

Algorithme de mise à jour du couple (i, n) après l'éclatement d'une case

n :=n+1

Si *n>=2^i* alors

n :=0;

i :=i+1;

Fin si

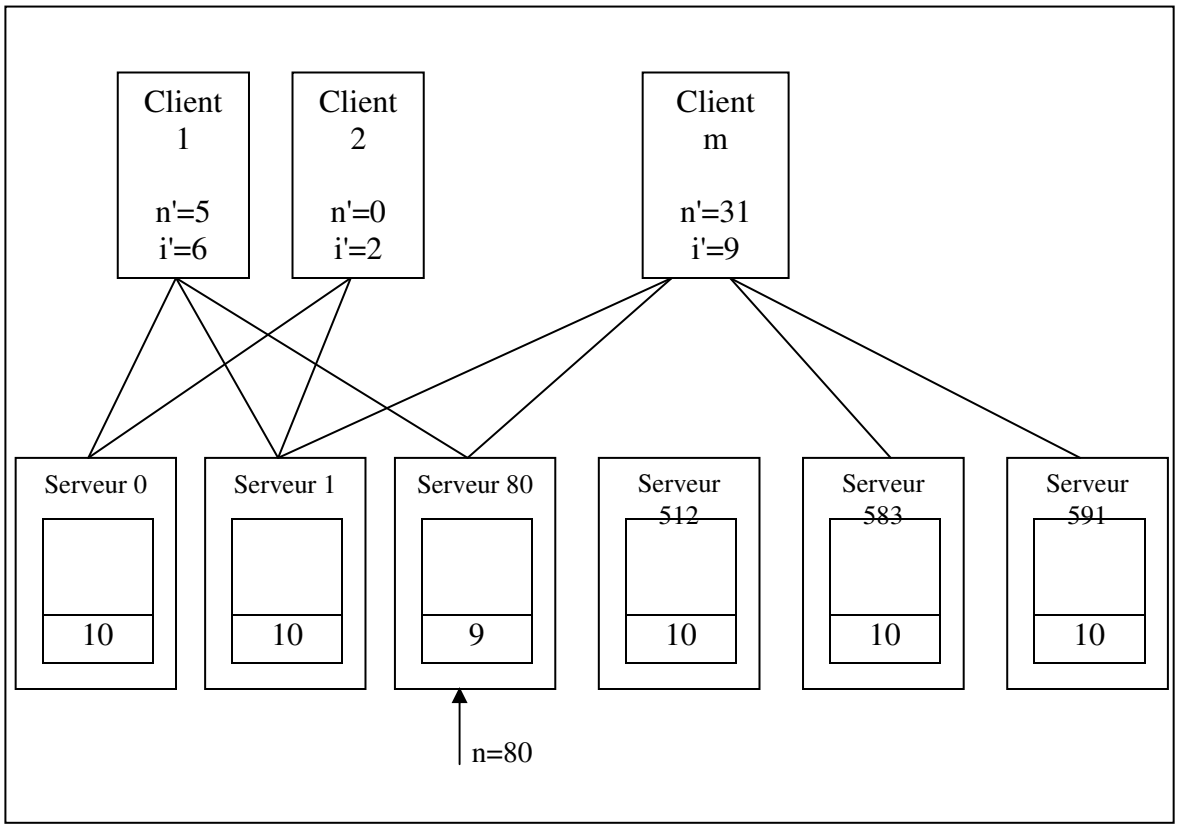


*Fig .3.8. Principes de LH**

Inversement à l'éclatement dû au débordement du fichier, il existe la fusion qui résulte de la diminution du taux de remplissage du fichier (après des suppressions

par exemple) et qui provoque la mise à jour du couple (i, n) selon l'algorithme suivant:

Algorithme de mise à jour du couple (i, n) après la fusion de deux cases

n :=n-1;

Si *n<0* alors

i :=i-1;

n :=2^i -1;

Fin si

Comme toute SDDS, un client LH* possède sa propre image de fichier, il possède deux valeurs i' et n' qui sont utilisées pour le calcul de l'adresse a' de l'enregistrement identifié par la clé C en utilisant son propre algorithme d'adressage:

Algorithme d'adressage exécuté par le client

a' := $h_i'(C)$;

si *(a'<n')* alors

a' := $h_{i'+1}(C)$;

Le site coordinateur maintient les valeurs de i et n ; le client possède une copie de ces valeurs qui seront changées dès qu'un éclatement aura lieu.

Lorsque le client fait une erreur d'adressage, il reçoit un message d'ajustement d'image de la part du serveur qui reçoit ses requêtes.

Algorithme d'ajustement de l'image du client

Si *j* > *i'* alors

i' := *j* – 1;

n' := *a* +1 ;

Si *n'* ≥2^*i'* alors

n' = 0;

i' := *i'* +1 ;

La requête envoyée par le client est vérifiée par la case si elle lui est destinée ou non, sinon elle la traite et l'envoi vers une autre case LH*, le schéma LH* garantit qu'une requête atteint le bon serveur au maximum au bout de deux renvois, et voici l'algorithme de test et de renvoi utilisé dans cette opération :

Algorithme de test et de renvoi exécuté par la case LH*

$a' := h_j(c)$;
si $a' = a$ alors
 accepter c ;
sinon $a'' := h_{j-1}(c)$;
 si $a'' > a$ et $a'' < a'$ alors
 $a' := a''$;
 envoyer c au paquet a' ;

II.6.4. Les variantes des LH*:

II.6.4.1. LH*$_{LH}$

Proposé par Karlesson, Litwin et Risch [7], cette SDDS utilise deux niveaux d'indexation, le premier est un niveau d'indexation *réseau* permettant au client de trouver la case LH* pour exécuter la requête, et le deuxième est un niveau d'indexation *interne* géré par l'algorithme de hachage linéaire et qui permet au client de trouver la case LH* pour exécuter une requête, la structure d'une case LH*$_{LH}$ est illustrée par la figure suivante.

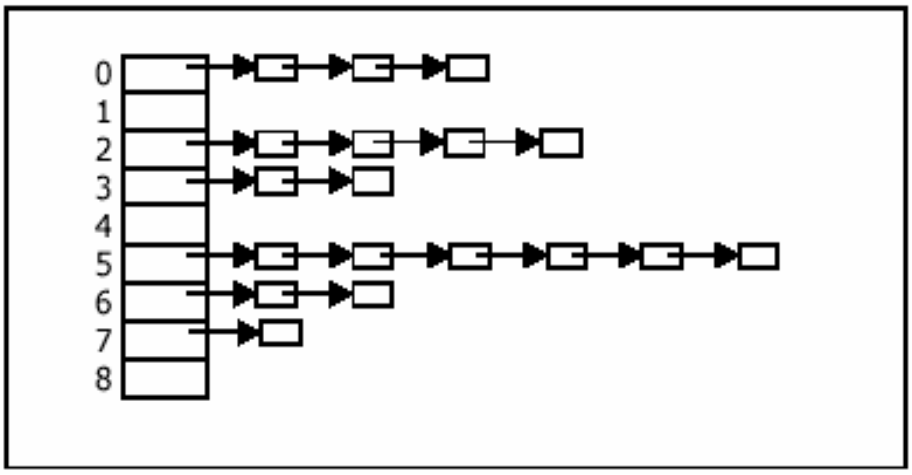


Fig .3.9. Illustration de la SDDS LH$_{LH}$*

II.6.4.2. LH*m

Elle vise à garantir la disponibilité des fichiers en dupliquant les fichiers sur d'autres sites serveurs, elle permet la tolérance aux fautes, mais en terme de stockage elle est trop coûteuse.

L'existence d'un miroir implique que pour un fichier LH*, un enregistrement existe au moins dans deux cases LH*.

A chaque miroir correspond un ensemble de clients primaires. Les deux ensembles correspondant aux miroirs sont disjoints. Un client primaire C du miroir F est noté F-client.

Un F-client peut accéder à F', tout en ayant le statut d'un client secondaire, et vice-versa. Le fichier F est dit primaire, tandis que son miroir F' est dit secondaire.

L'accès à F' est permis mais il est exceptionnel, il est réservé aux cas d'échec ou d'équilibrage de charge entre les deux fichiers [7].

II.6.4.3. LH*g

Elle implémente le concept de groupage, chaque groupe est supplémenté d'un serveur dit de parité. Elle assure la haute disponibilité tout en permettant au groupe de survivre à l'échec d'un serveur.

Un fichier LH*g est une paire de deux fichiers LH* : F1 et F2: étant respectivement le fichier primaire et le fichier de parité. Une case m appartient à F1, appartient au groupe de cases g = [m/k], tel que k est la taille d'un groupe de cases.

Chaque enregistrement de données occupe un rang r (ou ordre d'insertion), il possède trois champs : la clef de l'enregistrement C, le couple numéro du groupe et celui du rang (g, r) et un champ Attributs non-clé.

Quant à un enregistrement de parité, il est identifié par le couple (g, r), la liste des clés des enregistrements de rang « r » et appartenant au groupe « g », et un champ Bits de parité. Ce dernier est la parité impaire des champs Attributs non-clé des enregistrements [7].

II.6.4.4. LH*s

Le schéma LH*s présente un aspect de sécurité renforcé. Il assure la tolérance aux fautes, car il permet la reconstitution des enregistrements en cas de destruction des segments.

Un fichier LH*$_S$ est constitué de k+1 fichiers segments LH*, étant S1… Sk+1. Le fichier segment de parité Sk+1 permet la récupération de données perdues, appartenant à un fichier segment de données.

Le client procède en deux étapes pour insérer un enregistrement de clé C et d'attribuer la séquence de bits B : [b1.. bk bk+1 b2k … bmk]. Dans une première étape, il crée « k » fragments (k > 1) tel que chaque fragment « i » est la concaténation des bits [bi bk+i b2k+i …]. Chaque fragment i est envoyé par la suite au segment correspondant Si. Dans une deuxième étape, il génère le fragment de parité Sk+1 qui comprend la clé C et la chaîne de bits [b'1 b'2 b'3 … b'm], tel que bj' est égal au $\bigoplus_{j \in [1,k]} b_{ij}$. , où $b_{i,j}$ désigne le j^{éme} bit du fichier segment Si.

Le schéma LH*$_S$ est 1-disponible. Il présente également un aspect de sécurité renforcé.

En effet, chaque fichier segment n'a qu'un bit de k bits consécutifs de B. Désignons par « x » le nombre de bits manquants pour un intrus, ce dernier doit énumérer les 2x possibilités pour reconstituer un fragment, x étant égale à | B | (1- 1/k).

La sécurité de LH*$_S$ est au prix d'une complexité dans la restitution et la recherche des enregistrements, puisque les manipulations client nécessitent une coopération de tous les segments. Ainsi, une variante de LH*$_S$, se propose de faire la fragmentation au niveau attributs de cases [7].

II.6.4.5. LH*$_{SA}$

Elle permet la récupération automatique de plusieurs pages à la fois en utilisant plusieurs fichiers de parité [9]

Contrairement aux schémas décrits ci-dessus, le schéma de disponibilité de LH*$_{SA}$ est k-disponible. La k-disponibilité du schéma LH*$_{SA}$ se matérialise par le fait qu'une case de données appartient à plusieurs groupes de parité.

L'article propose des fonctions fi tel que i varie de 1, 2, …, k et un paramètre J du fichier permettant de générer des groupes de parité. Un groupe de parité est un ensemble au maximum de k cases de données [7].

$$f_t : m \rightarrow g_t \quad tel\ que$$
$$g_t = (m \bmod k^{t-1}) + \left\lfloor m / k^t \right\rfloor$$
$$i \in [1..J]$$

Les groupes générés par chaque fonction sont :

$$f_1 : (m, m+1, m+2, \ldots, m+(k-1)) \quad pour \quad m = 0, \ldots, k, \ldots \quad pas = k^0$$
$$f_2 : (m, m+k, m+2k, \ldots, m+(k-1)k) \quad pour \quad m = 0, \ldots, k-1, k^2, \ldots, k^2+(k-1), 2k^2, \ldots \quad pas = k^1$$
$$f_3 : (m, m+k^2, m+2k^2, \ldots, m+(k-1)k^2) \quad pour \quad m = 0, \ldots, k^2-1, k^3 \ldots, k^3+(k^2-1), 2k^3, \ldots \quad pas = k^2$$
$$\ldots$$
$$f_t : (m, m+k^{t-1}, m+2k^{t-1}, \ldots, m+(k-1)k^{t-1}) \quad pour \quad m = 0, \ldots, k^{t-1}-1, k^t, \ldots k^t+(k^{t-1}-1), 2k^t, \ldots \quad pas = k^{t-1}$$

II.6.4.6. LH*$_{RS}$

Elle utilise le concept de groupes de parité pour assurer la haute disponibilité, et les codes solomon pour le calcul des données de parité.

Elle distribue les enregistrements de données sur les cases (ou serveurs) de données en utilisant le hachage linéaire. Tout ce qui concerne la distribution des données et la gestion des serveurs en surcharge est gérée par LH*.

A cet effet, la haute disponibilité dans LH*$_{RS}$ se matérialise par le concept de groupes de parité. Un fichier LH*$_{RS}$ est ainsi subdivisé en groupes de parité, où un groupe de parité est formé de m cases de données et de k cases de parité. Le groupe est dit alors k-disponible, car il peut survivre à l'échec d'au maximum k cases.

Le concept de groupes de parité à l'échelle des cases s'applique au niveau des enregistrements des cases, on parle alors de segments de parité. Notons que, chaque

enregistrement de données a un rang r qui reflète sa position dans la case de données.

Un groupe d'enregistrements, ou segment de parité, de rang r contient tous les enregistrements de données ayant le même rang dans le même groupe de parité. Les enregistrements de parité de clé r sont calculés à partir d'un groupe d'enregistrements de données de même rang et même groupe.

La figure suivante illustre un groupe de quatre cases de données, et auquel sont rattachées deux cases de parité.

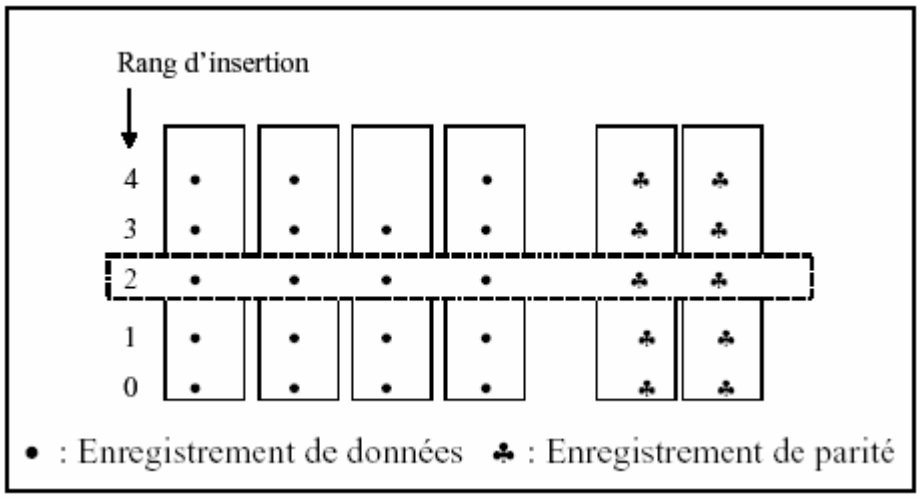


Fig .3.10. Un groupe de parité d'un fichier LH$_{RS}$ (m = 4, k = 2).*

II.7. Les SDDS basées sur la distribution par intervalle

Une SDDS basée sur la distribution par intervalle selon [10] répartit l'espace des clés par intervalles de cases. Dans cette organisation, un intervalle unique de l'espace des clés est associé à chaque case du fichier SDDS.

Parmi les SDDS connues de cette distribution, on trouve celles qui se basent sur les arbres 1-d comme:

- RP* (Distributed Rang Partitionning) et ses variétés.
- DRT (Distributed Relaxed Trees).
- DRT* (Distributed Random Tree).…

Et celles qui se basent sur les arbres k-d comme:

- K-RP* et ses variétés.
- K-DRT.
- Arbre B+ distribué (Distributed B+ tree). …
- Et bien d'autres.

II.7.1. Présentation des SDDS RP*

Comme toute SDDS, les SDDS RP* sont logées sur des serveurs qui peuvent être accédées par des sites clients en utilisant des requêtes de recherche d'une clé ou des requêtes par intervalles, selon la technique d'adressage utilisée au niveau du client, on trouve trois variétés des SDDS RP* qui sont: les RP*n, les RP*c et les RP*s. on trouve aussi une SDDS à haute disponibilité dénommée $RP*_{HA}$.

II.7.2. Les principes des SDDS RP*:

Un fichier RP* est constitué d'enregistrements identifiés chacun par une clé primaire, ces enregistrements sont stockés sur les serveurs dans des espaces mémoires appelés cases ou paquets. Chaque case est logiquement ordonnée suivant les valeurs des clés, elle possède une capacité **b** (b>>1).

Chaque paquet possède un en-tête contenant deux valeurs: une clé minimale notée λ et une clé maximale notéeΛ, l'intervalle $[\lambda, \Lambda]$ est appelé portée ou rang de la case. La structure d'une case SDDS RP* est illustrée par Fig.3.11. Un enregistrement de clé C appartient à une case si $\lambda < C \leq \Lambda$ c'est-à-dire $C \in]\lambda, \Lambda]$.

*Fig.3.11. Structure d'une SDDS RP**

Un fichier RP* comporte une case unique notée case 0 avec $\lambda = -\infty$ et $\Lambda = +\infty$. Initialement, toutes les insertions vont à la case 0, elle est éclatée en deux lorsqu'elle est débordée. Cet éclatement se fait après l'identification de la clé du milieu, ensuite presque la moitié des enregistrements est transférée dans une nouvelle case appelé case 1 d'une portée de [C,Λ] et la portée de la case 0 deviendra [λ, C]; avec C la clé de l'enregistrement en milieu. Fig.3.12. illustre l'évolution d'un fichier RP*.

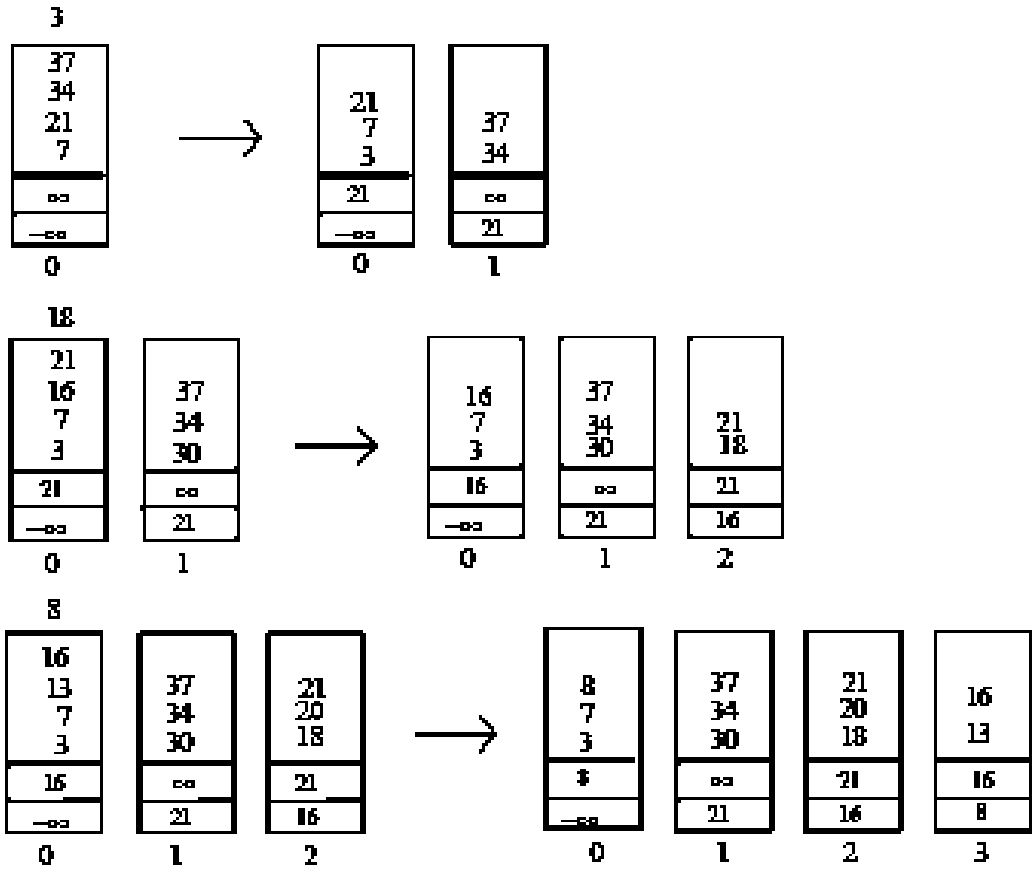


*Fig.3.12. Evolution d'un fichier RP**

L'éclatement d'une case RP* est réalisé selon l'algorithme suivant [6] :

Algorithme d'éclatement d'une case RP*:

1/ déterminer (Cm) la clé de l'enregistrement du milieu de la case de débordement

2/ créer une nouvelle case j

3/ déterminer l'en-tête de la case j

$\lambda j := Cm$;

$\Lambda j := \Lambda$;

- Copier dans la case j les enregistrements de la case i dont la clé C>Cm

4/ modifier l'en-tête de la case i

$\lambda i := \lambda$;

$\Lambda i := Cm$;

- Effacer les enregistrements de la case i dont la clé C>Cm

Les clients accèdent aux fichiers à l'aide des requêtes, une requête peut porter sur un enregistrement ou un intervalle d'enregistrements ou bien encore un ensemble d'enregistrements vérifiant une condition sur un champs non clé, donc les types de requêtes comme c'est présenté par A.Diène [10] peut être divisés en :

- requête à clé unique ou simple : correspond à une recherche ou à une mise à jour.
- Requête par intervalle : c'est une mise à jour des champs non clés de l'ensemble d'enregistrements de clé C tel que $C \in [C1, C2]$ avec C1<C2. [C1, C2] est le rang de la requête.
- Parcours ordonné : c'est un parcours des enregistrements dans l'ordre logique suivant les valeurs croissantes ou décroissantes des clés. L'ensemble de ces clés appartient à l'intervalle de la requête.

II.7.3. Les variantes des RP*:

Les SDDS de type RP* se distinguent selon la présence ou l'absence d'index et la manière de communication entre les clients et les serveurs. Généralement, on distingue deux types de messages de communication, le multicast qui est destiné à un ensemble de machines appartenant à un même groupe sur un réseau donné, et l'unicast qui est destiné à une seule machine du réseau.

II.7.3.1. RP*n

Décrit un partitionnement des enregistrements par intervalle définit sur l'espace des clés, dans leurs communication, les clients émis des messages multicasts aux serveurs qui répondent avec des messages unicasts.

II.7.2. RP*c

Ajoute à RP* une image au niveau de chaque client, ainsi le mode de communication entre les clients et les serveurs sera basé sur les messages unicasts. Les messages multicasts ne seront utilisés que pour les redirections (cas d'erreur).

II.7.3. RP*s

Ajoute un index réparti au niveau des serveurs, ce qui permet l'utilisation des messages unicasts lors des redirections.

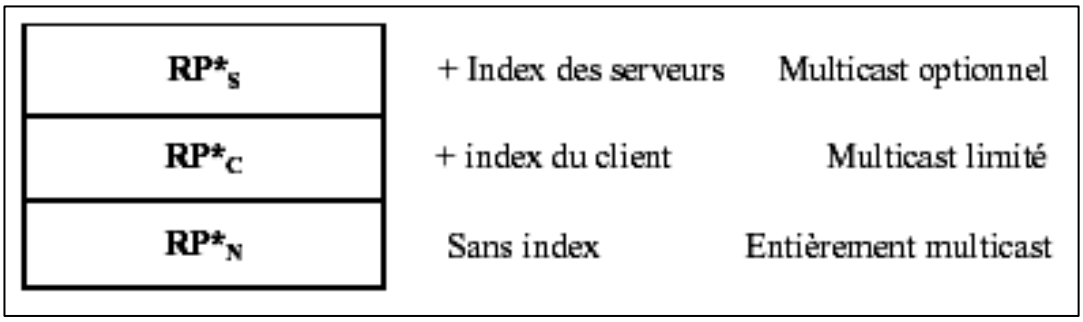


*Fig.3.13. Structure de la famille des SDDS RP**

II.7.4. RP*ha

Combine entre la récupération efficace des enregistrements et le parcours ordonné avec la possibilité de tolérance aux pannes, ceci est assuré par la réplication de chaque intervalle de l'espace des clés sur deux serveurs. Il y'a aussi un hôte dédié qui se comporte comme un coordinateur d'éclatement et de recouvrement, il connaît toutes les organisations courantes du fichier.

L'éclatement d'un paquet coûte cinq messages, seulement un parmi eux transporte un grand volume de données, tandis que le recouvrement d'un paquet inclut un message multicast et cinq messages unicasts où deux parmi eux transportent les données.

Conclusion

Dans ce chapitre on a expliquer les SDDS d'une manière simple, on a aussi pris soins de discerner les SDDS basées-hachage et les SDDS basées-arbre. Pour en savoir, les types ou les variétés des SDDS sont en progression continue, et ceci afin de faciliter la manipulation des données et de tirer profit des ressources des multiordinateurs en terme de stockage, de sécurité et de vitesse de traitement.

Chapitre 4

SDDS et BD

Introduction

Une base de données distribuée sur un multiordinateur peer to peer est tout simplement appelée base de données peer to peer (BDP2P), cette notion a fait l'objet de nombreuses recherches dans plusieurs universités, ainsi, différents projets ont été réalisés pour étudier les différents mécanismes de relation entre ces BD, la gestion et la coordination de communication entre les peers ainsi que la gestion des requêtes sur ces BD.

En effet, ce genre de base de données est utilisé par une variété d'applications généralement caractérisées par un taux de scalabilité plus ou moins important. Ainsi, pour profiter des caractéristiques des multiordinateurs, la notion des SDDS a été introduite dans le domaine des bases de données.

Donc, ce chapitre est conçu afin d'exposer les différents projets réalisés dans le domaine des BDP2P, et aussi ceux réalisés pour l'utilisation des SDDS dans une configuration P2P, et dans le domaine des BD distribuées sur les sites du multiordinateur.

I. État de l'art sur les BD P2P

Nombreux sont les projets qui travaillent sur les BD P2P, l'idée principale de ces projets est de permettre aux BD de réaliser leurs tâches dans un environnement peer to peer tout en tirant profit des caractéristiques de ce type d'environnement.

Comme c'est mentionné ci avant, les réseaux P2P possèdent plusieurs caractéristiques qui peuvent servir d'appuie lors de la conception et l'implémentation de bases de données**.** Donc, les systèmes de gestion de données doivent fournir l'ensemble des fonctionnalités nécessaires comme la gestion et l'optimisation des requêtes, les vues, les contraintes d'intégrité,…etc, pour définir les relations entre les données.

I.1. Définition d'une BD P2P

Une base de données peer to peer est un ensemble de sites autonomes connectés par un réseau peer to peer. Chaque site (peer) échange des données et des services avec les autres peers, il possède une base de donnée avec un schéma de donnée local mais ne dispose pas de schéma global.

I.2. Projets de recherches sur les BD P2P

Roshelova Albena dans [36] a identifié un ensemble de projets qui travaillent sur la relation entre les BD et les systèmes P2P, en passant à la présentation d'un système de gestion de base de données pour finir avec la proposition d'un ensemble de techniques permettant le développement d'un tel système dans un environnement P2P tout en préservant les relations sémantiques d'interdépendance entre les bases de données.

Et parmi les projets distingués, qui travaillent dans ce domaine, nous pouvons citer les suivants :

Piazaa : C'est un projet étudié par l'université de Washington, il vise à définir les relations sémantiques entre les pairs de bases de données existantes sur les différents peers du réseau à travers l'utilisation de "*schéma de médiation*".

Des techniques sont proposées pour l'utilisation de ces relations en vue de répondre aux différentes requêtes.

LRM : C'est un projet étudié par l'université de Trento, il vise à définir une nouvelle structure théorique pour l'établissement de relations entre les schémas, il introduit un nouveau concept qui est la coordination de bases de données permettant aux peers de gérer sémantiquement l'interdépendance de leurs bases de données en utilisant le paradigme peer to peer.

PMS : C'est une architecture de médiation basée sur le paradigme peer to peer où les sources et les médiateurs coopèrent pour le but d'estimer les réponses aux requêtes données.

Hyperion : Etudié par l'université de Toronto, ce projet vise à définir l'architecture de gestion des données P2P et à étudier l'intégration des données et les mécanismes de relation et d'échange dans un environnement P2P.

Peer DB : Un projet développé à l'université de Singapore, c'est un système de gestion de bases de données dans un environnement p2p, il est totalement autonome à cause de l'absence totale de centralisation. Actuellement, les recherches faites par ce projet sont focalisées sur les techniques sophistiquées de modélisation sémantique des données, la gestion des requêtes, les processus de mise à jour et les mécanismes intelligents de coordination entre les BD.

Autres projets : D'autres projets ont été cité dans [34], ils favorisent tous l'utilisation du paradigme p2p, parmi lesquels on trouve : AmbientDB, Diane, HePTox, Osiris, Scopes, TripCom, et UniStore.

I.3. Dimensions de conception de BD P2P

Angéla Bonifati a parlé dans [34] des dimensions de conception d'une BD P2P, de ses fonctionnalités et de ses classes d'application, ces dimensions sont énumérées comme suit :

I.3.1. La topologie

Il faut prendre en compte la topologie du réseau à adopter, comme les réseaux structurés basés sur les tables de hachage distribuées "DHT "(CAN, Chord,..), ou les réseaux semi structurés basés sur les super-peers (Kazaa,..), ou bien les réseaux non structurés basés sur l'indexation des requêtes (Gnutella,..).

I.3.2. L'autonomie

Elle dépend du réseau choisi, à titre d'exemple : les peers dans un réseau basé DHT sont autonomes par rapport aux fonctions de hachage distribuées, par contre les peers dans un réseau basé sur les super-peers sont autonomes par rapport à l'existence des super-peers qui les coordonnent.

I.3.3. La disponibilité des données

Un peer doit décider de partager la totalité de ses données ou une partie d'elles avec les autres peers, pour cela, on suppose qu'il y a un degré de possession de données passant de "share all" vers "share nothing", alors la disponibilité du maximum (resp. minimum) de données est atteinte lorsque le peer partage tout (ou rien) de ses données.

I.3.4. La possession des données

Cette dimension est strictement attachée avec la précédente (disponibilité des données), il faut penser au degré de possession que le peer veut avoir en partageant ses données. Par exemple : le peer doit décider si les données à partager avec les autres peers peuvent être "propriété" du réseau ou elles doivent rester maintenues par des droits d'accès.

I.3.5. La réplication des données

La décision prise par le peer indique s'il y a une permission de réplication de données au-delà de leur dépôt local, ainsi le niveau maximal de réplication est atteint lorsque le peer permet aux autres peers de copier ses données, à ce moment, le niveau minimal de réplication n'est permis que si une panne survienne.

I.3.6. La capacité des requêtes

Cette dimension décrit les fonctions de recherche permises par le réseau, et qui peuvent varier entre la recherche par clé, par requêtes simples ou sophistiquées; en utilisant par exemple : CompleteSQL, XQuery Stats, …etc.

I.3.7. La politique de mise à jour

Cette dimension décrit le niveau d'adaptabilité qui est réservé aux données et aux méta-données des peers. Les mises à jour sont toujours possibles sur les données locales et sur toutes les données dont le réseau est le propriétaire. Ainsi, les changements des dépôts locaux doivent être diffusés aux indexes globaux.

I.3.8. Contenu des réseaux homogène .vs. Hétérogènes

Dans les réseaux p2p, chaque peer possède un schéma de données local qui peut être différent des schémas utilisés par les autres peers du réseau. Si un seul schéma est employé alors le réseau est considéré comme étant homogène, autrement, il est hétérogène.

I.3.9. La découverte des ressources, des données et des services

Cette dimension décrit les mécanismes de découverte des données, des ressources et des services. Elle dépend de la topologie du réseau, par exemple : toutes les données sont indexées en utilisant une DHT dans des réseaux structurés, donc toutes les nouvelles données entrées dans le réseau ont besoin d'une clé dans la DHT. Par opposition, dans les réseaux non structurés, un simple mécanisme de

propagation de message garantie que les peers prends en compte les données récemment entrées.

En revanche, le peer peut décider quel type de découverte est nécessaire d'être utilisé dans une BD P2P.

I.3.10. Les garanties possibles

Cette dimension concerne la consistance et l'atomicité des opérations du réseau. Il semble que le concept de transactions connu dans les BD centralisées ne peut pas être appliqué directement au BD P2P, par exemple, lorsque des résultats partiels sont trouvés; la politique de "tout ou rien" pour les transactions n'est pas acceptable.

I.4. Les fonctionnalités requises

Parmi les fonctionnalités d'une BD P2P, Bonifati a cité :

I.4.1. Le modèle de données

Le modèle de données choisi à l'intérieur d'une BD P2P donnée est évidemment pertinent, ce modèle peut être en XML, relationnel, Orienté-Objet, RDF ou une combinaison de ces modèles.

I.4.2. Le langage de requêtes

Le langage de requête correspondant doit être choisi en fonction du modèle de données, il peut être : SQL, XQuery, SPARQL, IR-based,…etc.

I.4.3. L'interface BD

Pour l'utilisateur et l'administrateur, cette interface BD a le rôle de spécifier quoi faire et comment le faire, il semble très intéressant de construire une interface qui permet au réseau d'être transparent.

I.4.4. La découverte des ressources et des services

Cette fonctionnalité fournie les méthodes de découverte définissant l'usage du réseau.

I.4.5. Les transactions distribuées

Cette fonctionnalité fournie les procédés offrant les garanties sur les données et les opérations sur ces données.

I.5. Classes d'applications des BD P2P

Bonifati a mentionné aussi un ensemble d'applications travaillant sur ou avec les BD P2P, cité comme suit :

I.5.1. Cross-organizational workflow (organigramme de mouvement inter-organisationnel)

Chaque nœud de ce *workflow* est un peer dans le réseau, il ignore la totalité du *workflow* et ne connaît que la prochaine tâche à réaliser.

Les modèles et les tâches peuvent être répliqués et distribués entre les peers selon des propriétés comme leurs sémantiques, leurs performances, leur qualité de service ou autres critères.

I.5.2. Les dépôts scientifiques

Chaque peer détient des données scientifiques qui peuvent être intéressantes pour les autres peers, quelques unes de ces données peuvent avoir besoin d'être réconciliées ou intégrées avec d'autres données externes, ou de subir d'autres opérations importantes telles que les ajouts et les transactions.

I.5.3. Index multimédia collaboratif

Pour les données images par exemple, une opération d'indexation semble cruciale en vue de garantir l'efficacité du processus de découverte d'ajout et de suppression,

donc les données ici ont besoin d'être annotées afin d'assurer la prise en compte des recherches et l'exécution des opérations parallèlement sans conflits.

I.5.4. Archives web

Une version centralisée de l'archive web existe sous le nom de *archive.org*, toutefois la version P2P de l'archive web doit être capable de traiter un volume aussi important des données historiques du web que la version centralisée.

II. Utilisation des SDDS dans une configuration peer to peer

Comme déjà vu, les SDDS ont été spécialement conçues pour les configurations multiordinateurs afin de pallier certaines insuffisances des structures de données classiques et d'atteindre certains objectifs comme la scalabilité et l'exploitation maximale des capacités cumulées de stockage et de traitement.

Plusieurs projets utilisant les SDDS ont été réalisés pour des configurations multiordinateurs de type client-serveur parmi lesquels on trouve :

Le système ***SDDS-CP*** implémenté sous Windows NT qui est un système de communication des SDDS RP*, est conçu, comme la plupart des applications réseaux, d'un côté se comportant identiquement à un client et d'un autre côté à un serveur. La partie serveur de l'application a pour fonction de procurer des services définis pour les clients [10].

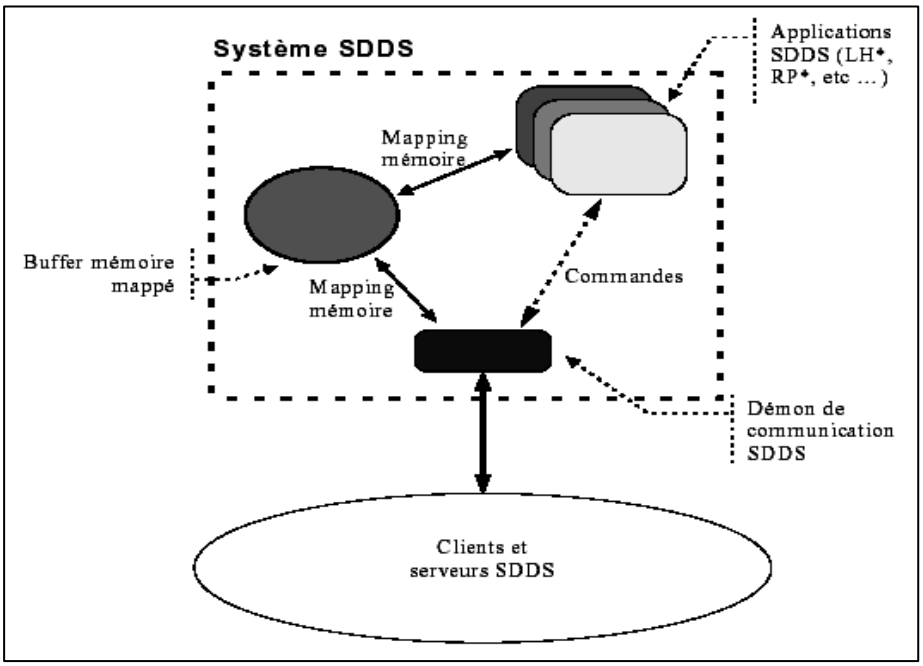


Fig.4.1. Architecture globale du système SDDS

Le gestionnaire prototype des SDDS appelé ***SDDS-2000*** (Fig.4.2) implémenté sur Windows 2000 pour les multiordinateurs Wintel, [64, 2] et depuis lequel, plusieurs autres versions ont été proposées (**SDDS-200X** avec X=3..9).

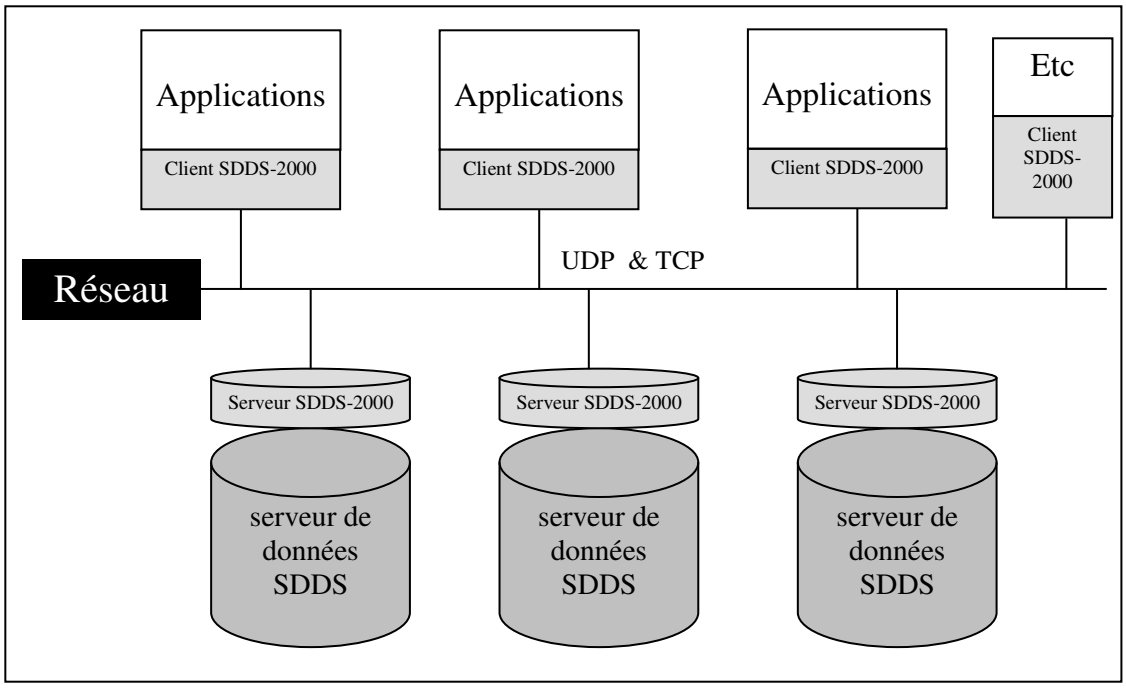


Fig.4.2. Architecture globale du système SDDS-2000
sous une configuration client/serveur

Il est aussi conçu pour la configuration peer to peer

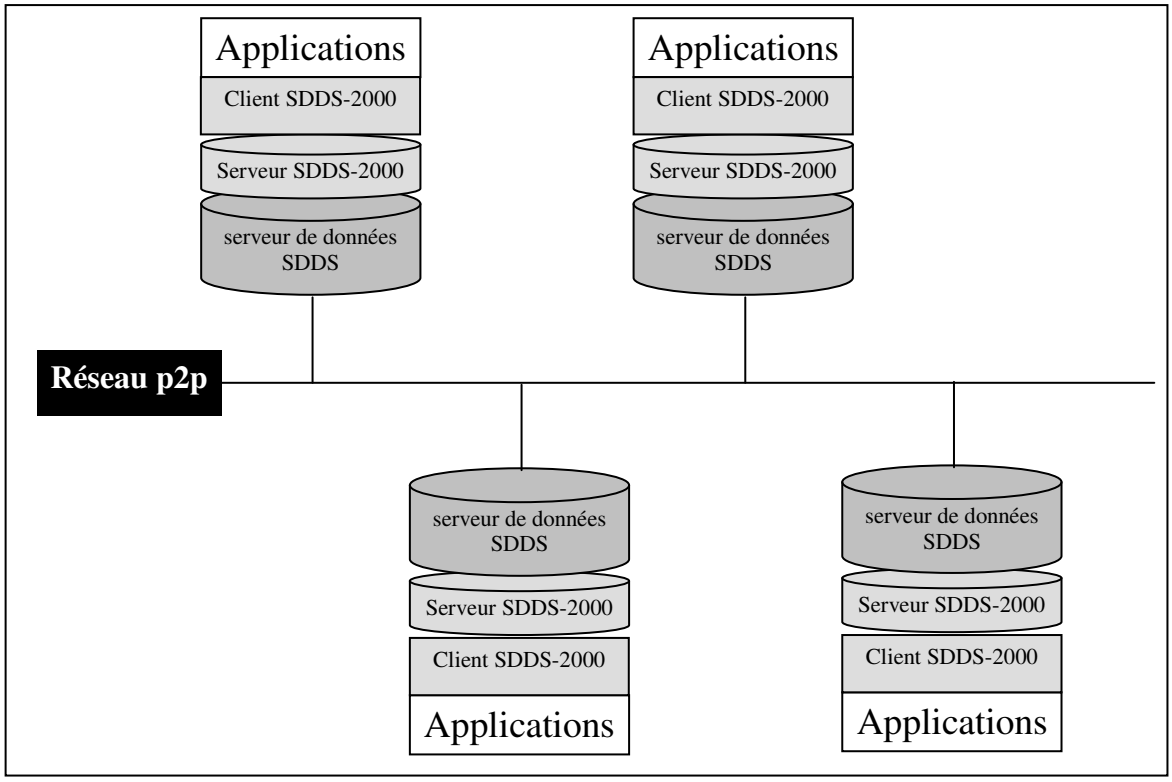


Fig.4.3. Architecture globale du système SDDS-2000
sous une configuration peer to peer

En outre, H.Yakouben et W.Litwin ont travaillé sur la SDDS ***LH*rs*** et ont proposé une SDDS conçue spécialement pour l'environnement peer to peer (Fig.4.4), qu'ils ont appelé ***LH*rsP2P*** [68],

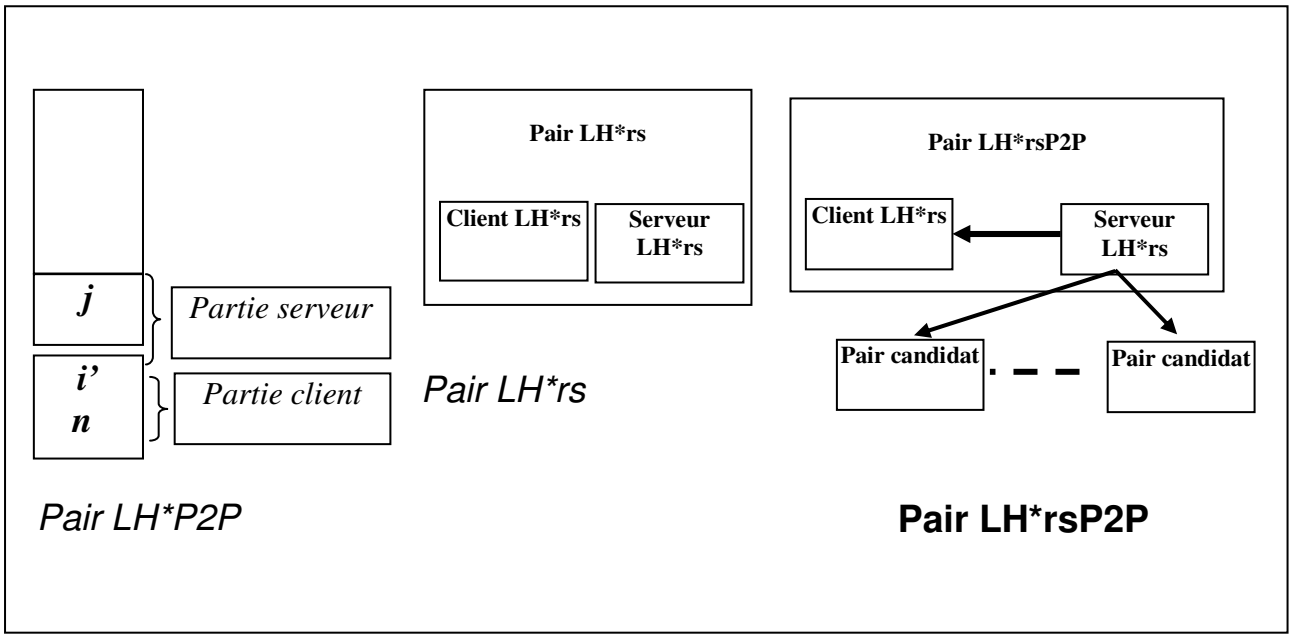


*Fig.4.4. Illustration de la SDDS LH*rs P2P*

III. Utilisation des SDDS avec les SGBD

Des systèmes résultant du couplage d'un SGBD avec le système SDDS-2000 ont été proposés [65, 66, 70], parmi lesquels on trouve :

SD-Amos qui consiste en un serveur SDDS utilisant un SGBD en interne comme gestionnaire de mémoire, dont les données sont stockées dans AMOS-II (le SGBD en question) et la répartition scalable des données est gérée par la couche SDDS.

DB2-SDDS, qui vise à concevoir une interface permettant à DB2 d'utiliser les services du client SDDS pour envoyer une requête de recherche sur des données distantes après la construction d'une liaison entre le SGBD et un entrepôt de données externes avec des accès rapides hors SGBD.

Amos-SDDS qui consiste en la définition d'une architecture pour l'usage de fichier RP* comme source de stockage interne par le SGBD Amos-II, il essaye d'étendre le gestionnaire SDDS afin qu'il puisse supporter des

opérations de bases de données et d'effectuer les traitements parallèles en se basant sur l'envoi de fonctions sur des sites distants.

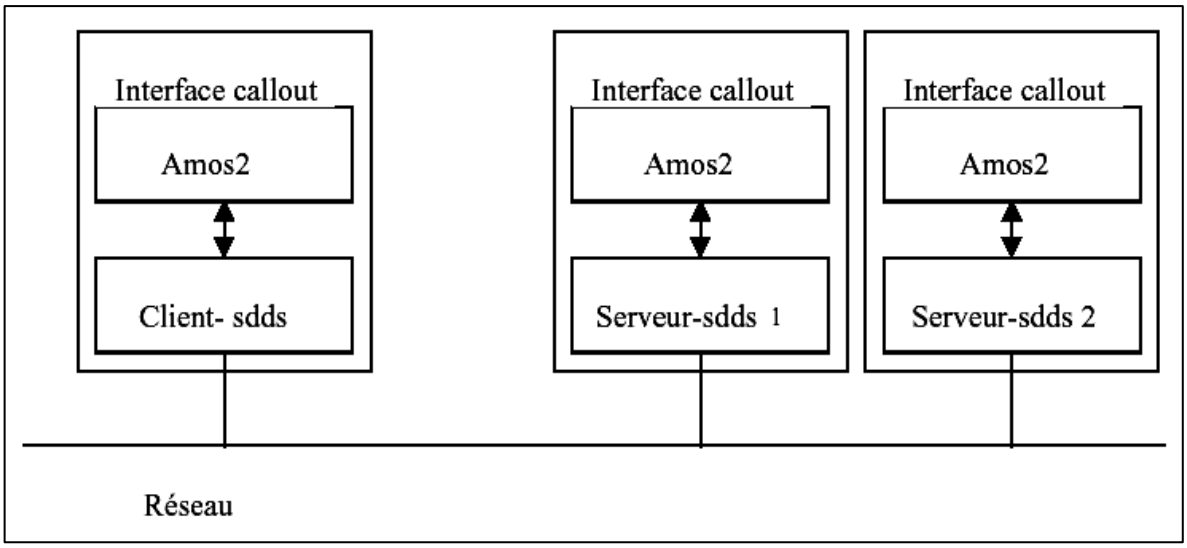


Fig.4.5. Architecture globale du système Amos-SDDS

et ***Monet***, un SGBD dans lequel est intégré une SDDS LH*, les réalisateur de ce projet se sont basés sur les étapes d'interprétation et d'optimisation de requêtes de cet SGBD, ainsi l'interpréteur des opérations d'algèbre relationnel a été étendu pour permettre l'accès transparent aux données distribuées ou stockées localement. Cependant, l'optimisation des opérations de déploiement des différentes stratégies et scénarios à l'intérieur de l'opérateur primaire associé avec la SDDS a ajouté la caractéristique d'auto-adaptation au système de requêtes (c'est-à-dire : il s'adapte dynamiquement aux situations imprévues).

Le système prototype de stockage dénommé ***ICON SDDS-2000,*** qui a été projeté pour être un système de base de connaissance puissant et sophistiqué illustré par la figure suivante.

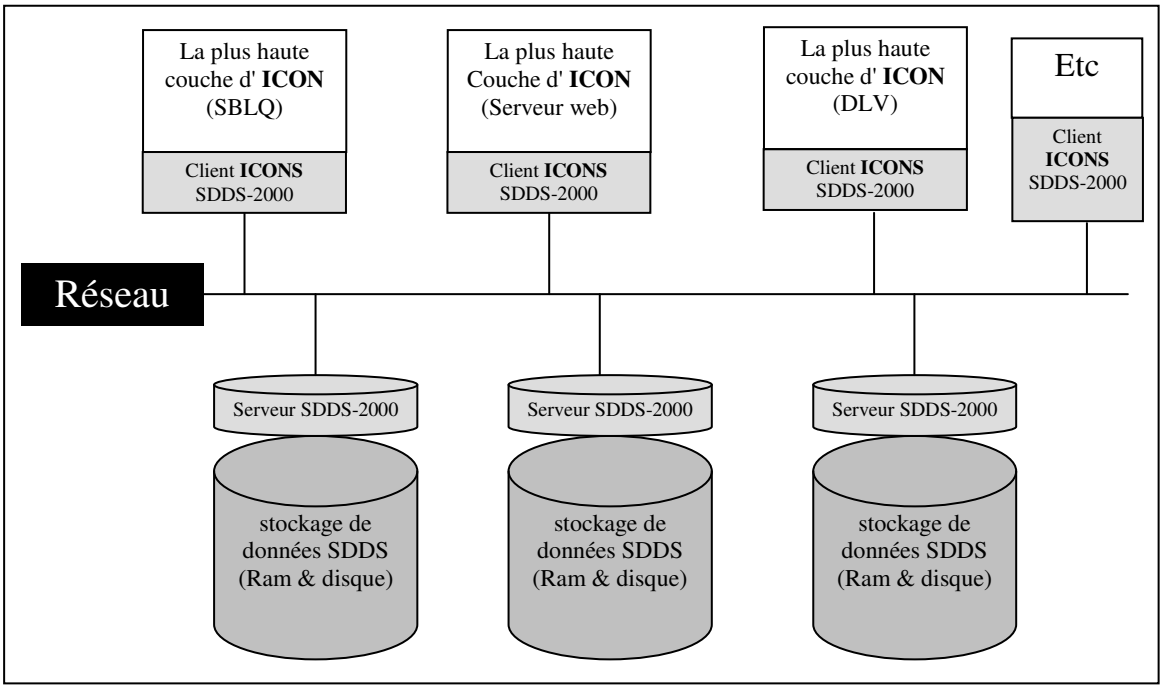


Fig.4.6. L'architecture générale du système de stockage ICONS SDDS

Le langage de requête pour les besoins de la gestion des données relationnelles : le ***SD-SQL Server*** proposé par S.Sahri dans le cadre des projets de recherche dirigés par Dr. W.Litwin à l'université Dauphine (Paris) [67, 69].

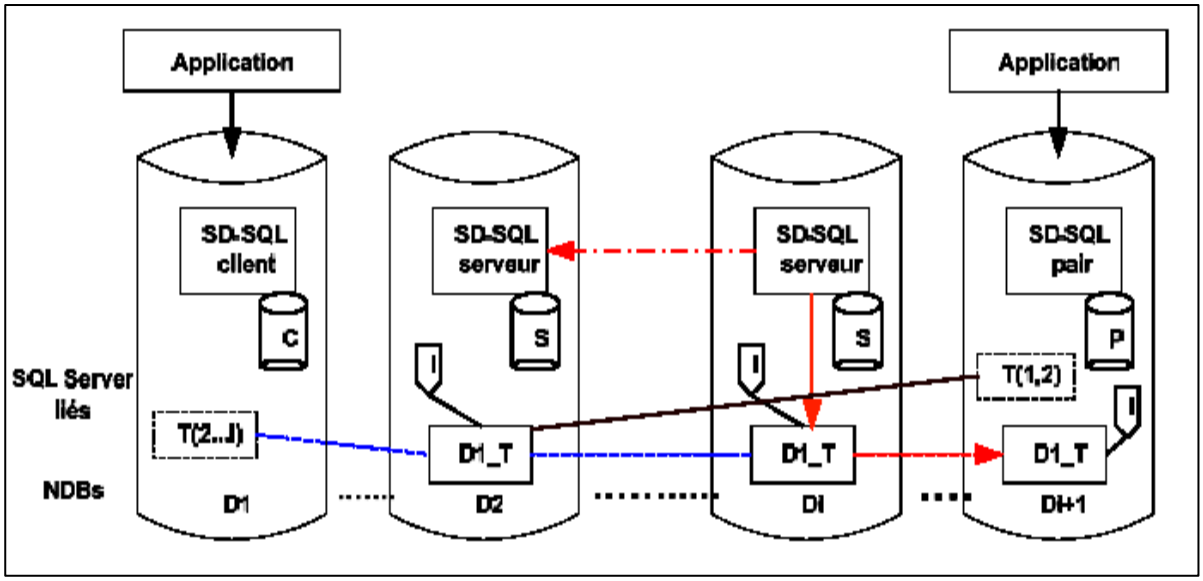


Fig.4.7. Architecture de SD-SQL Server

Le principe de ce projet est l'application de la technologie des SDDS aux SGBD pour permettre de gérer le partitionnement distribué et scalable des données, ainsi le concept des SGBD distribués et scalables est apparu.

SD-SQL Server implémente une nouvelle architecture de SGBD relationnel dont la base est dite base scalable, elle grandit par la partition dynamique, scalable et distribuée de ses tables, qui sont aussi scalables. Ces tables scalables sont organisées en segments où chaque segment est placé sur un nœud du réseau P2P lié de SQL Server. La partition et son évolution sont invisibles à l'usager/application.

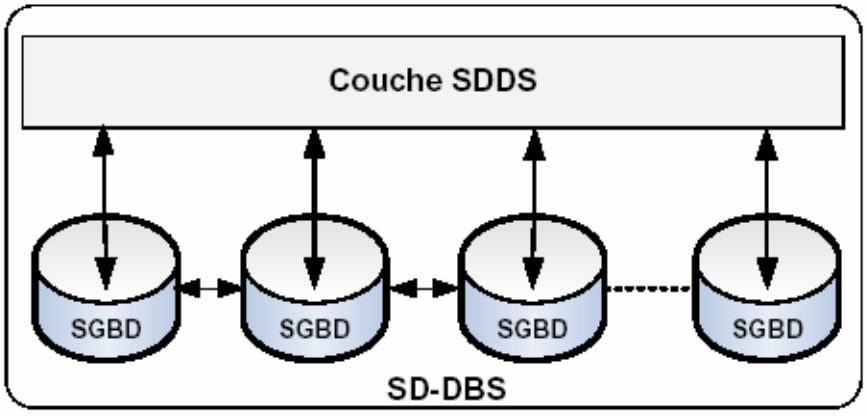


Fig.4.8. Présentation générale d'un système distribué de bases de données scalables

Un SD-DBS constitue une couche supplémentaire sur une collection d'instances de SGBD liés entre eux, comme c'est illustré dans la Figure 4.8. Cette couche est basée sur les principes des SDDS.

…. Et bien d'autres.

Conclusion

Ce chapitre vient d'exposer les différents projets réalisés dans le domaine des bases de données peer to peer distribuées sur les différents sites d'un multiordinateur, il mentionne aussi les différents projets réalisés pour la nouvelle structure de données (SDDS) en particulier ceux qui ont un rapport avec les SGBD.

Chapitre 5

L'utilisation des SDDS dans le niveau physique d'une BDR

Introduction

Les SGBD ont connu plusieurs organisations selon la technique employée pour leur implantation physique. Les développements remarquables faits dans des domaines aussi divers que les réseaux de communication ont permis à l'approche bases de données réparties de devenir une solution alternative à la centralisation.

Généralement, L'explosion et la complexité des données rendent les bases de données de plus en plus volumineuses, pour faire face à cela, les noyaux des SGBD doivent tirer le meilleur Profit des nouvelles architectures de machines. Alors, Les structures de données distribuées et scalables - SDDS - ont été proposées pour constituer de telles architectures, elles permettent de fournir un mécanisme général d'accès à des données réparties dynamiquement.

Comme c'est déjà mentionné, ces structures assurent des temps d'accès beaucoup plus courts que les temps d'accès aux données stockées sur les disques ainsi que Le traitement parallèle utilisé pour le but de maximiser le débit d'un système distribué en réduisant le temps total des requêtes et minimisant leur temps de réponse.

Dans ce chapitre, nous proposons une étude théorique pour voir la possibilité de l'utilisation de cette nouvelle structure dans le niveau physique d'une BD relationnelle. On commence par le passage des structures de données classiques aux SDDS, vient par la suite l'idée de remplacement du fichier classique de la BDR par un fichier SDDS, pour arriver à la répercussion de ce choix sur l'architecture fonctionnelle de la BD relationnelle.

Enfin, cette possibilité de remplacement est expliquée à travers le traitement et l'optimisation des requêtes appliquées sur les enregistrements de la SDDS.

I. Le passage des structures de données classiques aux SDDS

Divers projets ont été effectués dans le domaine des structures de données distribuées, ils consistent tous à chercher une meilleure solution pour assurer la disponibilité et la scalabilité des données, pour cela, les chercheurs se sont basés sur les structures de données classiques et ont essayé de les adapter aux environnements distribués. C'est le cas de toutes les SDDS proposées jusqu'à présent.

Dans ce qui suit, nous essayons d'indiquer l'ensemble des structures classiques et les SDDS qui en découlent.

I.1. Structures basées sur le hachage

Plusieurs chercheurs considèrent que les SDDS constituent une extension des schémas de hachage classiques sur les multiordinateurs comme par exemple les SDDS : LH*, DDH*, EH*, IH* et leurs dérivés qui utilisent dans leur principes les structures de données classiques basées sur le hachage linéaire, dynamique, extensible ou autres.

Exemple d'une structure classique basée-hachage et de sa SDDS correspondante :

DLH est une distribution de LH sur une machine parallèle à mémoire partagée, elle consiste à distribuer les paramètres i et n de fichier LH sur tous les processeurs de la machine, ce qui entraîne des erreurs d'adressage. Son inconvénient principal est que le nombre de processeurs sur une machine parallèle est limité (32 processeurs au maximum), donc DLH n'est pas une SDDS car elle ne satisfit pas la propriété de scalabilité.

Pour remédier à ce problème, une SDDS été proposée, la LH*, qui ne diffère pas beaucoup dans ses principes généraux de ceux de DLH.

Donc, un fichier LH* est distribué sur plusieurs serveurs LH*, il est accédé par des applications fonctionnant sur des machines autonomes dites clients LH*. Le

fichier est constitué d'enregistrements regroupés en M cases de taille b résidantes en mémoires principales, à raison d'une case par serveur.

L'adresse physique en réseau de chaque case est sauvegardée dans des tables d'allocation physiques disponibles chez les clients et chez les serveurs. Une trace de niveau j (indice de la fonction de hachage) est sauvegardée dans chaque serveur et un serveur particulier appelé coordinateur d'éclatement garde une trace des valeurs réelles de i et n de fichier [9].

I.2. structures basées sur les arbres (structures ordonnées)

A leurs tours, les SDDS basées sur les arbres étendent les structures ordonnées traditionnelles au environnements distribués comme par exemple les SDDS : RP*, DRT, DRT*n, arbres B+ distribué, K-RP*, K-DRT et leurs dérivés qui exploitent les principes des B-arbres, arbres B+, K-d-arbres ainsi que d'autres organisations classiques.

Exemple d'une structure classique basée-arbre et de sa SDDS correspondante :

Dans le domaine de stockage des données, un arbre est une structure de données hiérarchique et généralement dynamique, représentée par un ensemble non vide de nœuds (sommets) et des arrêtes assujettis à certaines conditions.

Les arbres ordonnés sont des arbres dont les données associées aux feuilles (nœud externe : qui n'a pas de successeur) sont triées en parcourant toutes les feuilles de gauche à droite. Les principaux avantages des arbres ordonnés se résument en des recherches rapides non séquentielles et des insertions ou suppressions ne provoquant pas des décalages.

Le principal inconvénient des arbres ordonnés est qu'ils deviennent déséquilibrés après plusieurs insertions ou suppressions des feuilles d'une manière non uniforme, les accès deviennent ainsi presque séquentiels.

Pour remédier à ce problème, les arbres B (B-arbres) été proposés, ils permettent d'assurer l'équilibre de l'arbre au détriment de la perte d'une partie de l'espace de stockage.

Plusieurs schémas de structures de fichiers distribuées basés sur les arbres ont été proposés, range-partitionning (RP) a été celui qui a connu le plus de succès, mais il reste limité par sa non scalabilité. Pour pallier à ce problème, une SDDS appelée RP* a été proposée en généralisant le concept des arbres B aux environnements distribués[9].

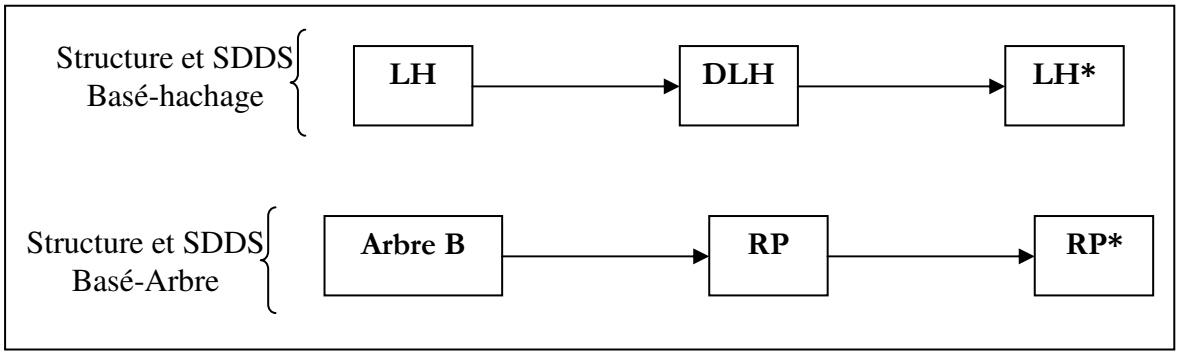


Fig.5.1.Exemple d'illustration des organisations classiques et des SDDS qui l'en découlent (méthodes : basé-hachage & basé-arbre)

II. Remplacement du fichier classique de la BDR par un fichier SDDS

Une BDR se distingue -comme c'est déjà vu- par un ensemble de relations, chacune d'elles contient un ensemble de tuples représenté dans son implémentation physique sous forme d'enregistrements. L'attribut clé de la relation peut correspondre à la clé de l'enregistrement, les autres attributs non clé représentent les champs restants de l'enregistrement.

Principe de remplacement

Les enregistrements de la relation sont regroupés en pages et dans le cas d'une BDR distribuée, ces pages sont réparties sur plusieurs sites. Ainsi l'accès aux données stockées dans les enregistrements est effectué selon une méthode d'accès précise parmi celles connues.

En utilisant un fichier SDDS à la place d'un fichier classique nous gardons les mêmes principes de l'implémentation physique, dans ce cas l'implémentation relationnelle typique peut être vue comme suit :

Une ***page relationnelle*** sera remplacée par un ***paquet d'enregistrement de la SDDS***.
Un ***tuple relationnel*** sera remplacé par un ***enregistrement de la SDDS***.
L'attribut ***clé de la relation*** sera remplacé par la ***clé de l'enregistrement de la SDDS***.
Les ***attributs non clé*** de la relation seront remplacés par les ***champs non clé de l'enregistrement SDDS***.
La ***méthode d'accès aux enregistrements*** relationnels conduit à ***choisir la SDDS adéquate*** pour l'implémentation (c'est-à-dire une SDDS basée-hachage, Basée-arbre ou autres).

II.1. Répercussion des SDDS sur l'architecture fonctionnelle d'un SGBDR

Pour l'architecture fonctionnelle d'un SGBD relationnel plusieurs fonctions peuvent être assurées parmi lesquelles le traitement des requêtes qui est le centre de notre intérêt et avec lequel nous montrons la possibilité et l'importance de l'utilisation des SDDS dans l'implémentation physique de la BD relationnelle.

Apparemment, l'utilisation des SDDS dans un SGBD relationnel peut présenter les avantages suivants :

Faciliter la manipulation des données,

Assurer la transparence de distribution et d'accès aux données,

Assurer la scalabilité des données,

Faciliter la gestion des accès concurrents

Minimiser le temps d'accès aux données,

Augmenter la disponibilité des données et leur sécurité,

Permettre l'exécution des requêtes parallèles,

.. et bien d'autres.

II.2. L'utilisation des SDDS dans le traitement et l'optimisation des requêtes relationnelles

Pour la gestion des tables dans une BDR, le SGBD fait recours au SGF qui s'occupe de la gestion des fichiers composant ces tables, bien sûr le SGF dispose de plusieurs modules lui permettant d'accéder à l'information enregistrée pour la traiter.

Le SGBD manipule les données de la BD à l'aide d'un langage de manipulation de données comme le SQL par exemple, avec ce langage de requêtes, l'utilisateur précise les propriétés des données qui l'intéressent sans fournir d'algorithme d'accès. Les optimiseurs de requête sont les composants qui réalisent cette dernière tâche, leurs objectifs sont de :

vérifier la correction syntaxique de la question,

rechercher des questions équivalentes plus simples,

déterminer l'ordre des opérations élémentaires d'accès en vue de :

réduire le nombre d'entrées/sorties,

réduire le temps CPU,

réduire la taille des ressources mémoires nécessaires,

optimiser en premier les questions les plus fréquentes.

Chaque SGBD contient un optimiseur de requêtes qui peut être légèrement différent d'un SGBD à l'autre. Les algorithmes utilisés sont rarement connus en détail mais les grandes étapes de l'optimisation sont en général :

1. Représenter la requête sous une forme interne et la décomposer en une séquence d'opérations élémentaires.

2. Transformer la requête par :

simplification : remplacement d'une question par une question équivalente plus simple.

ordonnancement des opérations élémentaires.

3. Construire un ensemble de plans d'exécution candidats.

4. Calculer le coût de chaque plan et choisir le meilleur.

5. Exécuter la requête.

Les différentes étapes d'optimisation des requêtes par un SGBD relationnel sont indiquées par Fig.5.2.

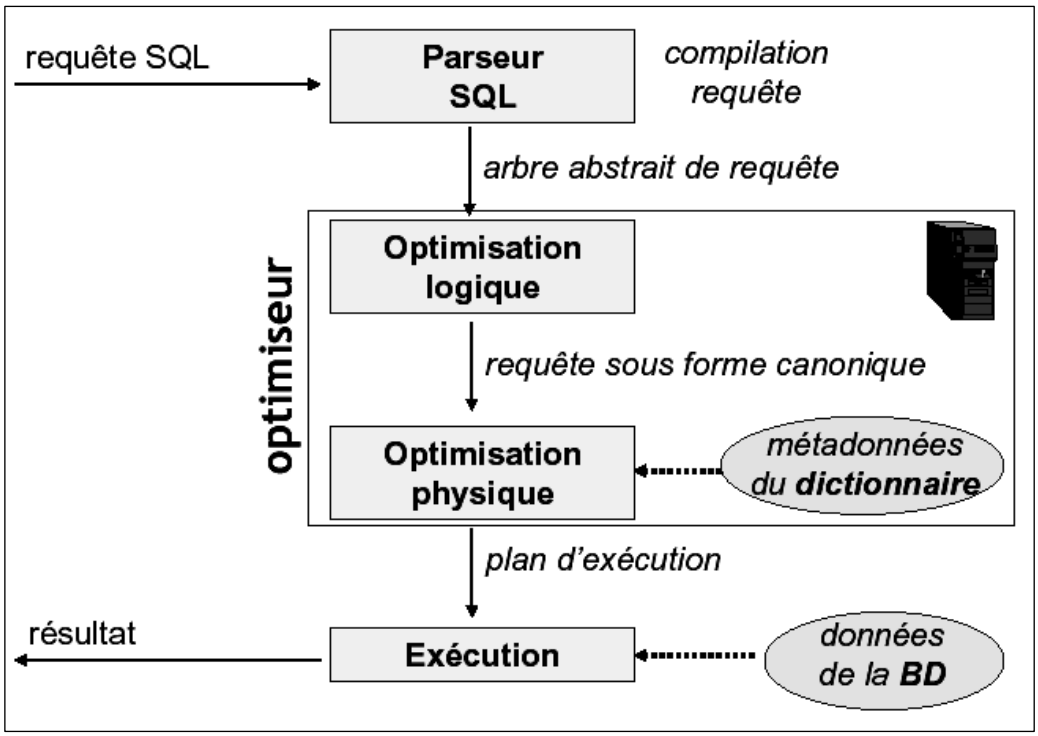


Fig.5.2. Etapes d'optimisation des requêtes SQL

A. Optimisation logique des requêtes (réécriture)

Elle permet d'obtenir une représentation canonique de la requête, sous la forme d'un arbre algébrique, dans lequel les opérations sont ordonnées et les critères mis sous forme normale. Elle peut comporter elle-même deux étapes :

Réécriture sémantique : qui transforme la question en prenant compte les connaissances sémantiques sur les données (exemple : les contraintes d'intégrité).

Réécriture syntaxique : qui profite des propriétés simples de l'algèbre relationnelle pour ordonner les opérations.

B. Optimisation physique

Son objectif majeur est le choix des procédures de niveau inférieur les plus adaptées pour la réalisation des opérateurs relationnels qui sont au centre du moteur du SGBD responsable de l'exécution des plans :

1- *Opérateur de sélection :*

Le placement des données dans les fichiers est effectué dans le but d'optimiser les sélections et les jointures les plus fréquentes. Dans ce cas deux algorithmes doivent être considérés :

La sélection par balayage séquentiel nécessitant de comparer chaque tuple de la relation opérande avec le critère.

L'utilisation d'index qui associe les valeurs d'un attribut avec la liste des adresses de tuples ayant cette valeur. Il est généralement organisé comme un arbre-B ou un arbre-B+.

2- *Opérateur de tri :*

Le tri est un opérateur important car il est utilisé pour les requêtes demandant une présentation de résultats triés, mais aussi pour éliminer les doubles, effectuer certaines jointures et calculer des agrégats.

3- *Opérateur de jointure :*

Comme avec les sélections, il est possible de distinguer deux types d'algorithmes selon la présence ou l'absence d'index sur les attributs de jointures ou non :

Jointure avec index : si les deux relations sont indexées sur les attributs de jointure, il suffit de fusionner les deux index pour obtenir les couples

d'adresses joignant, mais si seulement l'une des relation est indexée, il faut balayer la deuxième relation et accéder au fichier indexé pour chaque tuple.

Jointure sans index : en l'absence de l'index, il existe trois algorithmes fondamentaux :

a) les boucles imbriquées : il consiste à lire séquentiellement la première relation et à comparer chaque tuple lu avec chaque tuple de la deuxième relation.

b) le tri-fusion : son algorithme effectue le tri des deux relations sur l'attribut respectif de jointure. Ensuite, la fusion des tuples ayant la même valeur est effectuée.

c) le hachage : son algorithme consiste à hacher la petite relation et de garder l'adresse des enregistrements et leurs valeurs correspondantes d'attributs de jointure dans une table (table de hachage), et la comparer à la deuxième relation selon l'attribut de jointure.

4- Calcul des agrégats :

Les algorithmes permettant de calculer les agrégats sont basés soit sur le tri, soit sur une combinaison de hachage et de tri.

Pour la gestion des données réparties sur un multiordinateur, les règles d'exécution et les méthodes d'optimisation de requêtes définies par un contexte centralisé sont toujours valables, mais il faut prendre en compte la répartition de ces données sur différents sites et la communication entre ces derniers pour le transfert des données.

Donc comme pour le traitement de requêtes en BD centralisée, on produit l'arbre algébrique de la requête que l'on optimise avec les mêmes critères. Chaque feuille de l'arbre représente une relation et chaque nœud représente une opération algébrique. Après son optimisation, la requête algébrique est enrichie avec les informations nécessaires pour son exécution sur le réseau.

En plus des opérateurs d'optimisation précédemment mentionnés (sélection, tri, jointure), les BD réparties ont abouti à la définition d'un nouvel opérateur : « la semi-jointure », il s'agit d'une double jointure dont le principe est d'effectuer deux

petites jointures plutôt qu'une seule grosse, ainsi le flux de données sur le réseau est minimisé.

II.3. Exemples d'opérations simples de manipulation des données

Après cette brève explication, passant maintenant à l'utilisation des SDDS dans le traitement et l'optimisation des requêtes relationnelles :

Supposons qu'on a une base de données "Hôpital" composée des relations suivantes:

Patient (*NSS*, p_nom, age_p, id_dr, genre_mal, region).
Docteur (*dr_id*, specialité, dr_nom, dr_adr).
Service (*serv_id*, type).
Traitement (*patient_id, dr_id, serv_id*, date, coût).

On veut manipuler les données de cette BD avec les requêtes SQL tout en supposant que les tuples des relations proposées sont logés sous forme d'enregistrements dans des paquets d'une SDDS au niveau physique de cette BD. Cette manipulation peut être présentée par une recherche d'un ou de plusieurs enregistrements, une insertion, une modification, une suppression, ou un calcul particulier des agrégats connus en SQL. On peut utiliser des requêtes simples ou imbriquées, ou se servir des index pour la réalisation de ces opérations, leur exécution peut être aussi faite sur les différents nœuds du multiordinateur parallèlement.

II.3.1. La recherche

Une application peut faire une recherche dans une BD relationnelle à l'aide des opérations de base comme la projection, la sélection et la jointure de tables.

On distingue plusieurs types de recherche : une recherche avec restriction par clé primaire unique, par clé primaire multiple ou par un champ non clé et une recherche sans restriction.

La recherche avec restriction par un champ non clé:

La recherche est effectuée par une sélection qui est une interrogation des données par la clause SELECT du langage SQL, souvent avec une restriction par une qualification.

Selon le modèle relationnel, les sélections peuvent être faites en utilisant le contenu de n'importe quelle colonne et les lignes sont stockées dans n'importe quel ordre. Considérons la requête suivante :

```
SELECT  *
FROM Docteur
WHERE dr_nom = 'Mohamed'
```

Dans ce cas, on peut avoir le résultat selon l'une des deux méthodes suivantes : en balayant toute la table pour retrouver la ou les lignes pour lesquelles le nom du docteur est 'Mohamed', ou bien encore en se basant sur l'utilisation d'un index sur le champ 'dr_nom' de la relation « Docteur » permettant d'accélérer la recherche.

En utilisant une SDDS, cette recherche devient une recherche simple réalisée selon l'un des scénarios suivants :

Scénario 1 : balayage (Scan) des paquets SDDS :

le client envoie sa requête au site initiateur de l'opération (coordinateur, ou site du paquet 0 de la SDDS) qui l'a diffuse par un message multicast aux serveurs logeant les enregistrements de la table « Docteur ».

chaque serveur balaye le paquet de la table SDDS pour extraire les enregistrements vérifiant la condition de sélection donnée (dr_nom = 'Mohamed').

Les résultats trouvés sont ensuite stockés dans une table SDDS intermédiaire qui sera envoyée au site initiateur de l'opération.

Enfin, les tables intermédiaires reçues par le site initiateur de l'opération sont groupées dans une seule table envoyée au client comme réponse finale.

Scénario 2 : utilisation d'index

Une deuxième possibilité de recherche est l'utilisation d'un index sur le champ 'dr_nom'.

Ici, l'opération est effectuée par le balayage des tables-index, les clés des enregistrements où l'occurrence « Mohamed » du champ « dr_nom » apparaisse sont envoyées par un message unicast au coordinateur.

Le coordinateur initialise l'opération de recherche en émettant sa requête par un message multicast aux adresses des serveurs concernés, et récupère la table SDDS résultat pour l'envoyer au client.

On distingue aussi ***une recherche avec restriction par un champ clé:***

- La recherche avec restriction par clé primaire unique:

Supposons qu'on a une recherche par une clé primaire unique exprimée par la requête suivante:

```
SELECT *
FROM Patient
WHERE NSS='17350';
```

Elle peut être traduite par une simple recherche par clé de la SDDS utilisée, ainsi la clé NSS devient la clé primaire dans la table SDDS avec les tuples de la relation « Patient ».

Alors, l'opération de recherche se déroule comme suit : le client possède sa propre image de la table SDDS, il l'utilise pour envoyer sa requête au serveur par un message unicast, ainsi, on peut avoir deux possibilités ; soit l'adresse du serveur contenant l'enregistrement recherché est juste et en conséquence ce dernier est envoyé comme résultat au client ; soit l'image du client est fausse ou n'est pas actualisée, dans ce cas le serveur qui reçoit la requête la redirige vers le serveur adéquat et envoie un message d'ajustement d'image (IAM) au client.

En revanche, si le client ne possède pas encore une image de la table, il procède la recherche par l'envoie d'un message multicast, ainsi, le serveur concerné lui envoie

l'enregistrement résultat par un message unicast.

- Une recherche avec restriction par clé primaire multiple :

La qualification dans ce cas contient plusieurs prédicats reliés par une conjonction comme 'AND' ou 'OR'. Par example:

- requête 1: avec la conjonction AND

```
SELECT *
FROM Patient
WHERE NSS = '17350'
AND id_dr = '1670';
```

Dans cette recherche, la requête sera exécutée comme une intersection des résultats de plusieurs SELECT chacun appliqué à une clé.

Avec l'utilisation d'une SDDS cette requête peut être exécutée selon l'un des deux scénarios suivant :

Scénario 1 : requête directe :

En se basant sur l'image de la table SDDS, le client envoie sa requête par un message unicast au serveur possédant l'enregistrement de la clé désignée (NSS= '17350'), ce denier teste si la condition après l'opérateur de conjonction est réalisée ou non.

Si la condition testée est vérifiée, l'enregistrement résultat sera envoyé par un message unicast au client, sinon, le client sera informé de l'absence du résultat de sa recherche.

Scénario 2: utilisation d'index :

Après l'envoie de la requête au serveur-index l'enregistrement concerné (celui possédant l'occurrence avec la clé id_dr = '1670' -s'il existe-) sera envoyé au site coordinateur qui le vérifie s'il contient dans son ensemble la clé NSS = '17350'.

Si la clé cherchée est trouvée, le coordinateur procède par l'envoie d'un message unicast au serveur qui garde son enregistrement.

L'enregistrement récupéré par le coordinateur est envoyé comme résultat au client.

- requête 2: avec la conjonction OR

```
SELECT *
FROM Patient
WHERE NSS = '17350'
OR id_dr = '1670';
```

Ici, le résultat est l'union des résultats des SELECT appliqués à chaque clé.

Pour une recherche de cette forme avec une table SDDS, l'opération peut être réalisée de la façon suivante :

L'envoie de la requête se fait du client au coordinateur qui la diffuse à tous les sites serveurs stockant la table « Patient », ces derniers vérifient les conditions de la clause WHERE pour extraire les enregistrements demandés.

L'ensemble des résultats délivrés par les serveurs sera tous les enregistrements portant l'occurrence '1670' de l'attribut 'id_dr' ou l'occurrence '17350' dans le champ 'NSS'.

Enfin les enregistrements reçus sont stockés sur une table intermédiaire qui sera envoyée au client.

On peut aussi utilisé un index sur l'attribut id_dr de la table Patient, et ainsi l'opération se déroulera d'une autre façon qu'on peut expliquer par le scénario suivant :

Le coordinateur dirige la requête après sa réception vers le serveur contenant l'enregistrement -s'il existe- qui contient la valeur '17350' comme occurrence pour le champ 'NSS' et récupère cet enregistrement.

La même requête est envoyée en parallèle au seveur-index contenant la table d'index du champ 'id_dr' pour récupérer les clés des enregistrements ayant comme occurrence la valeur '1670' de l'attribut 'id_dr'.

Les enregistrements repérés sont ensuite récupérés à partir de leurs serveurs

par le coordinateur et groupés avec l'enregistrement précédent (identifié par la clé 17350) sur une seule table envoyée finalement comme réponse au client.

Une recherche sans restriction

Pour la plus simple des recherches, on parle de projection qui est la sélection sans restriction, exprimée par une requête de la forme :

SELECT attribut(s) FROM relation. Par exemple :

```
SELECT dr_id, spécialité
FROM Docteur;
```

Le résultat de cette requête est la projection des deux colonnes 'id_dr' qui est l'identificateur du docteur et 'spécialité' qui est la spécialité du docteur.

Dans le cas de l'utilisation d'une SDDS, l'opération de projection se déroulera selon le scénario suivant :

Après l'envoi de la requête par le client, le coordinateur qui la reçue la diffuse aux autres sites serveurs.

Ensuite, une opération de balayage des paquets SDDS est faite par chaque serveur logeant une partie de la table « Docteur ».

Le résultat de l'opération ici est l'ensemble des occurrences du couple d'attributs (id_dr, spécialité) groupé dans une table SDDS intermédiaire qui sera envoyée au coordinateur.

Et comme phase final, le coordinateur regroupe les tables intermédiaires reçues en une seule table qui sera envoyée comme résultat au client après l'élimination des enregistrements redondants.

Remarque

L'élimination des enregistrements redondants peut être faite localement sur les serveurs de la table, ainsi, la taille des tables intermédiaires envoyées au coordinateur sera plus petite ce qui diminue le flux des données sur le réseau et accélère le temps

de traitement du coordinateur.

On peut aussi utiliser pour ce traitement l'opérateur DISTINCT dans la requête SQL, ainsi, elle devient de la forme :

```
SELECT DISTINCT (dr_id, spécialité)
FROM Docteur;
```

En outre, on peut faire une recherche qui respecte l'ordre d'affichage des résultats à l'aide de la clause ORDER BY suivie de la liste des attributs servant de critère de tri, ce tri peut être ascendant si on ajoute le mot clé ASC à la fin de la requête ou descendant en cas d'ajout du mot clé DESC.

Par exemple :

```
SELECT *
FROM Docteur
ORDER BY spécialité ASC;
```

Le résultat de cette requête sera l'ensemble des enregistrements des docteurs trié par ordre croissant selon leurs spécialités.

On peut avoir avec une SDDS des résultats ordonnés de la même façon que pour une BD relationnelle qui utilise les structures de données classiques, dans ce cas, les résultats obtenus par la recherche dans la table SDDS seront stockés dans des tables intermédiaires triées localement sur leurs serveurs et envoyées au coordinateur qui procède à son traitement par leur fusion afin d'obtenir une seule table triée, ou elles sont envoyées directement sans être traitées au coordinateur qui s'occupe lui seul de leur tri avant d'être délivrées au client.

On peut aussi avoir des résultats triés si on utilise une SDDS qui préserve l'ordre des enregistrements comme la RP* par exemple, et donc on n'aura pas besoin de l'utilisation de l'opérateur ORDER BY sur les serveurs pour assurer l'ordonnancement des enregistrements récupérés.

Recherche avec jointure de tables

En plus de la sélection appliquée à une table, la jointure de plusieurs tables est considérée comme un élément important de recherche. En définissant la jointure de deux relations selon une qualification comme étant l'ensemble des tuples du produit cartésien satisfaisant cette qualification. Donc une recherche peut être faite dans une seule relation ou dans plusieurs relations avec ou sans l'utilisation d'index. Prenons par exemple la requête suivante :

```
SELECT p_nom, age_p, dr_nom
FROM Patient, Docteur
WHERE Patient.id_dr = Docteur.dr_id
```

L'exécution de cette requête est possible par deux types d'algorithmes selon la présence ou l'absence d'index sur les attributs de jointure. Dans le cas de l'absence d'index, une des trois méthodes peut être utilisée: les boucles imbriquées, le tri-fusion ou le hachage.

Avec une SDDS, la jointure peut être faite de la même façon, les enregistrements des deux tables « Patient » et « Docteur » seront extraits et stockés dans une table intermédiaire 'table1' avec l'attribut 'identificateur du docteur' comme clé primaire, ensuite les restrictions nécessaires seront appliquées sur cette table pour extraire les couples de tuple (tup1, tup2) ayant la même clé primaire, où tup1 appartient à la relation Patient et tup2 à la relation Docteur. Le résultat final sera gardé dans une autre table 'table2' qui sera envoyée au client.

Si d'autres restrictions sont spécifiées, elles seront appliquées à table2 considérée comme la table de jointure des tables de la clause FROM et le résultat sera l'ensemble des enregistrements satisfaisant ces restrictions.

Par exemple :

```
SELECT p_nom, age_p, dr_nom
FROM Patient, Docteur
WHERE Patient.id_dr = Docteur.dr_id
AND age_p>=25;
```

La restriction sur l'age est appliquée à la table de jointure des deux tables Patient et Docteur.

Remarque

On peut désigner des serveurs particuliers pour effectuer les jointures des tables qu'on appelle « serveurs de jointure ».

II.3.2. La mise à jour

Une opération de mise à jour dans une table d'une BD relationnelle peut être une insertion, une modification ou une suppression d'un tuple.

- Insertion d'un tuple

Effectuée par la clause INSERT INTO. Par exemple, on veut insérer un nouveau tuple dans la table Docteur avec les valeurs suivantes :

```
INSERT INTO Docteur (dr_id, specialité, dr_nom, dr_adr)
VALUES ('1683', 'Pédiatre', 'Mohamed', 'Batna');
```

L'insertion de ce nouveau tuple relationnel est traduite par l'ajout d'un nouvel enregistrement à la table SDDS, donc, à la réception de la requête une recherche locale est effectuée, si la donnée à insérer existe déjà l'opération d'insertion est abandonnée et le client est prévenu, sinon le nouvel enregistrement est ajouté et dans ce cas il peut y avoir un débordement du paquet SDDS ce qui provoquera son éclatement qui sera traité par le site coordinateur.

- Modification d'un tuple

Effectuée par la clause UPDATE. Par exemple, on veut modifier l'adresse du docteur portant l'identificateur '1587' :

```
UPDATE Docteur SET dr_adr ='Biskra'
WHERE dr_id = '1587';
```

En utilisant une SDDS, cette modification est exécutée après le repérage de

l'enregistrement concerné, dans le cas contraire, l'opération sera abandonnée et le client est prévenu de son échec.

- Suppression d'un tuple

Effectuée par la clause DELETE FROM. Par exemple, on veut supprimer de la table Patient le malade portant l'identificateur '19632'.

```
DELETE FROM Patient
WHERE NSS = '19632';
```

En utilisant une SDDS, cette suppression est exécutée après le repérage de l'enregistrement concerné, avec prise en charge de la gestion du paquet s'il y a un rétrécissement.

II.3.3.Le calcul des agrégats

Parmi les fonctions de calcul les plus connus on trouve : COUNT, AVG, SUM, MAX & MIN utilisées respectivement pour compter le nombre de valeurs non nulles, calculer la moyenne des valeurs d'une colonne, effectuer le cumul, chercher le maximum ou le minimum d'une colonne.

Voici quelques requêtes à titre d'exemple:

- La fonction COUNT:

Voici une requête qui compte les tuples de la table service.

```
SELECT COUNT (serv_id)
FROM Service;
```

Avec une SDDS, cette opération peut être effectuée selon le scénario suivant :

Le client envoie sa requête au coordinateur qui la diffuse aux serveurs SDDS.

Chaque serveur exécute localement la requête, il balaye le paquet des enregistrements de la table « Service » en les comptant, ainsi, le résultat sera leur nombre.

Après, le résultat de chaque serveur est délivré par un message unicast au

coordinateur, ce dernier traite ces résultats pour avoir le résultat final qui l'envoie au client.

- La fonction SUM:

Exprimée par une requête qui calcule la somme des docteurs ayant comme spécialité la chirurgie cardiaque

```
SELECT SUM (dr_id)
FROM Docteur
WHERE spécialité = 'chirurgie cardiaque';
```

Avec une SDDS, cette opération peut être effectuée selon le scénario suivant :

Le client envoie sa requête au coordinateur qui la diffuse aux serveurs SDDS.

Chaque serveur exécute localement la requête, il balaye le paquet des enregistrements de la table « Docteur » tout en testant la condition de la clause WHERE.

Après, le résultat de chaque serveur est délivré par un message unicast au coordinateur, ce dernier traite ces résultats pour avoir le résultat final (la somme des docteurs ayant comme spécialité la chirurgie cardiaque) qui l'envoie au client.

On peut aussi utilisé un index sur le champ 'spécialité' de la table « Docteur » et dans ce cas, on peut avoir le scénario suivant :

Après la réception de la requête, elle est exécutée en balayant le serveur-index.

Si l'occurrence 'chirurgie cardiaque' est trouvée l'opération de traitement des enregistrements la concernant est effectuée.

Ensuite, le résultat est envoyé par un message unicast au client.

- Les fonctions MAX & MIN:

Exprimées par une requête qui cherche à extraire respectivement la plus grande et la plus petite valeur du coût de traitement pour le docteur portant l'identificateur '1670'.

```
SELECT MAX (coût), MIN (coût)
FROM Traitement
WHERE dr_id = '1670';
```

En utilisant une SDDS, l'exécution de cette requête se déroulera comme suit :

Le client envoie la requête au coordinateur par un message unicast

Le coordinateur teste la requête et récupère à partir des différents serveurs les enregistrements ayant '1670' comme occurrence pour le champ dr_id et les stocke dans une table-SDDS intermédiaire (T1).

Ensuite, il balaye la table (T1) pour tester et extraire la plus grande et la plus petite valeur des occurrences du champ 'coût'.

Les résultats trouvés sont délivrés au client par un message unicast.

- La fonction AVG:

```
SELECT AVG (age_p)
FROM Patient
WHERE genre_mal = 'Diabète';
```

Cette requête calcule la moyenne d'age des malades diabétiques, elle s'exécute en calculant la somme des valeurs contenues dans la colonne age_p divisée sur le nombre total des enregistrements qui les contiennent.

Avec une SDDS, cette opération peut être divisée en sous opérations parallèles, chacune d'elles est exécutée sur un serveur logeant la table « Patient », elle calcule la somme des valeurs de la colonne 'age_p' et le nombre des enregistrements qui les contiennent, et ceci pour tous les enregistrements portant la valeur 'Diabète' comme occurrence du champ 'genre_mal'.

Les résultats obtenus seront ensuite assemblés sur un seul site (coordinateur) pour être traités avec la formule exprimée par : la division de la somme des sommes obtenues sur la somme des comptages d'enregistrements afin d'obtenir le résultat final qui sera envoyé au client par un message unicast.

Remarque

Dans ce type de traitement, on peut assigner des serveurs dédiés pour chaque fonction, par exemple on fait désigner un serveur pour calculer le nombre d'enregistrement d'une table distribuées sur plusieurs paquets d'une SDDS, un autre pour tester et extraire les valeurs maximale ou minimale d'un ou de plusieurs attributs de la relation, un autre serveur pour le calcul des moyennes d'attributs,…etc, et ceci afin d'alléger le coordinateur de ces opérations et de gagner un peu plus de temps surtout pour des requêtes portant plusieurs fonctions d'agrégats à la fois.

Dans les requêtes précédentes, on appliquait la fonction d'agrégation à l'ensemble du résultat d'une requête. Une fonctionnalité complémentaire consiste à partitionner ce résultat en groupes, et à appliquer la ou les fonctions (s) à chaque groupe.

On construit les groupes en associant les tuples partageant la même valeur pour une ou plusieurs colonnes.

Par exemple :

```
SELECT genre_mal, COUNT (NSS)
FROM Patient
GROUP BY genre_mal;
```

Une requête qui affiche les groupes de maladies avec leur nombre de patients, notons bien qu'on peut faire porter des conditions sur les groupes avec la clause HAVING. La clause WHERE ne peut exprimer des conditions que sur les tuples pris un à un.

Par exemple :

Trouvons le nombre des patients par service ayant dépenser plus de 30000 DA comme coût de traitement en janvier.

```
SELECT COUNT (NSS), serv_id, SUM (cout)
FROM Patient, Traitement
WHERE Traitement.patient_id=Patient.NSS
AND Traitement.date='janvier'
GROUP BY serv_id
HAVING SUM(cout) > '30000 DA'
```

On voit que la condition porte ici sur une propriété de l'ensemble des tuples du groupe, et pas de chaque tuple pris individuellement.

Le traitement de cette requête avec une SDDS sera divisé en plusieurs sous-traitements, par exemple :

Un traitement de jointure pour les tables SDDS « Patient » et « Traitement »,

Une extraction des enregistrements portant comme date la valeur 'janvier',

Le groupage des enregistrements obtenus par service,

L'addition et le test du coût dépensé pour chaque groupe d'enregistrements obtenu,

Et bien sûr le calcul des enregistrements obtenus dans la phase précédente.

II.3.4.Utilisation d'index

Une solution de recherche substituant le balayage est la création d'index, qui permettra de satisfaire aux requêtes les plus fréquentes avec des temps de réponses acceptables. Un index est formé de clés auxquelles SQL peut accéder très rapidement. Il se crée par la commande CREATE INDEX :

CREATE INDEX nom-index

ON table (colonne1, colonne2,...);

On peut remplacer cette opération par la création d'un fichier SDDS avec un ensemble d'enregistrements chacun contient un ensemble de champs divisé en deux parties : une partie pour la clé et une partie pour le groupe d'entrées à la valeur de cette clé.

La clé correspond au champ sur lequel l'indexation est faite, tandis que le groupe d'entrées englobe l'ensemble des clés primaires des enregistrements qui contiennent la valeur de la clé de l'index.

Prenons un exemple avec la BD proposée:

```
CREATE INDEX nom-doct
ON Docteur (dr_nom);
```

L'enregistrement SDDS qui remplacera cet index peut être de la forme suivante :

(dr_nom, (dr_id1, dr_id2… dr_id i)).

Avec dr_nom comme clé de l'enregistrement et (dr_id1, dr_id2… dr_id i) comme l'ensemble des enregistrements ayant l'occurrence correspondante au champ 'dr_id' désigné.

Généralement, l'adjonction d'un index accélère la recherche et ralentit la mise à jour, avec les SDDS ce problème ne se pose pas car l'index SDDS n'est pas affecté par le changement des locutions dû aux éclatements des paquets débordés.

Signalons qu'on peut utiliser les SDDS ordonnées comme les RP* pour les index ordonnés.

II.3.5. Les requêtes imbriquées

Jusqu'à présent les conditions exprimées dans la clause WHERE consistaient en comparaisons de valeurs scalaires. Il est possible également avec SQL d'exprimer des conditions sur des relations. Pour l'essentiel, ces conditions consistent en l'existence d'au moins un tuple dans la relation testée, ou en l'appartenance d'un tuple particulier à la relation.

La relation testée est construite par une requête SELECT ... FROM ... WHERE que l'on appelle sous-requête ou requête imbriquée puisqu'elle apparaît dans la clause WHERE.

Une première utilisation des sous-requêtes est d'offrir une alternative syntaxique à l'expression de jointures. Les jointures concernées sont celles pour lesquelles le résultat est constitué avec des attributs provenant d'une seule des deux tables, l'autre ne servant que pour exprimer des conditions.

Prenons l'exemple suivant : on veut chercher les adresses des docteurs qui ont soignés un malade agé de 80 ans habitant à Batna :

```
SELECT dr_id, dr_adr
FROM Docteur, Patient
WHERE Docteur.dr_id =Patient. id_dr
```

```
AND Patient.age_p=80
AND Patient.region='Batna';
```

Ici, le résultat, 'identificateur du docteur ', provient de la table Patient. D'où l'idée de séparer la requête en deux parties :

1- On constitue avec une sous-requête une liste des docteurs qui ont soignés un malade agé de 80 ans habitant à Batna, puis
2- On utilise ce résultat dans la requête principale pour trouver leurs adresses.

Ainsi, on obtient ceci :

```
SELECT dr_id, dr_adr
FROM Docteur
WHERE dr_id, IN (SELECT id_dr
                 FROM Patient
                 WHERE age_p=80
                 AND region='Batna';)
```

Le mot-clé IN exprime clairement la condition d'appartenance de 'identificateur du docteur ' à la relation formée par la requête imbriquée.

Avec une SDDS, le scénario d'exécution de cette requête se déroulera comme suit :

La sous-requête est exécutée en premier lieu avec l'un des scénarios de recherche portant sur plusieurs clés (déjà expliqué) dans la table SDDS Patient.

En deuxième lieu, le résultat obtenu est gardé dans une table intermédiaire ou une liste qui constituera le champ de recherche pour la requête principale appliquée à la table « Docteur ».

Ensuite, une deuxième recherche est effectuée pour donner l'ensemble des adresses des docteurs appartenant à la liste obtenue de la sous-requête et vérifiant les conditions données.

Enfin, le résultat final sera envoyé au client par un message unicast.

II.3.6.L'Optimisation des requêtes

Pour diminuer le volume de données transmis d'un site à un autre, il faut limiter les transferts d'information aux seules informations utiles. Pour cela il faut systématiquement rajouter des projections dans l'arbre algébrique pour abandonner les attributs inutiles. Il faut aussi noter que les parties de requêtes indépendantes peuvent être exécutées en parallèle sur des sites différents et donc baisser le temps total d'exécution.

Dans cette phase (optimisation), l'optimiseur assigne un coût à toute opération relationnelle d'une requête (par exemple : le nombre de paquets examinés), il examine surtout les jointures, et transforme l'arbre algébrique obtenu de la requête en un plan d'exécution. Noter bien que l'utilisation d'index diminue les coûts.

L'utilisation d'une SDDS permet de diminuer le temps d'accès à cause de la possibilité de maintenir la totalité des données d'une BD en RAM, en plus, avec un plan d'exécution de coût minimal et l'exécution parallèle des requêtes sur le multiordinateur, on obtient un gain de temps en particulier avec la scalabilité des données, et donc, la performance du SGBD peut être maintenue.

Conclusion

Dans ce chapitre, on a essayé d'expliquer la possibilité et l'importance de l'utilisation des SDDS à la place des fichiers traditionnels qui stockent les tables d'une BD relationnelle implémentée sur un réseau en se basant sur l'exécution et l'optimisation des requêtes SQL.

Ici, les requêtes données sont des requêtes simples qui traitent les tuples d'une table théoriquement implémentés dans des paquets d'une SDDS sous forme d'enregistrements.

Conclusion générale & perspectives

Ainsi se termine notre mémoire alors que la recherche dans le domaine des bases de données et des SDDS ne finira pas de se développer. Car l'apparition de ces nouvelles structures de données a ouvert un vaste champ d'exploration et de recherches dans la technologie informatique aux horizons sans limites.

Avec l'avènement des réseaux informatiques, un nouveau concept a été introduit : celui des multiordinateurs, et afin de profiter au maximum des ressources de stockage et de traitement de ces réseaux d'ordinateurs, une structure de données a été inventée, la structure de données distribuées et scalables (SDDS). Depuis, plusieurs SDDS ont été proposées, à savoir les SDDS basées sur le hachage constituant une généralisation des méthodes d'accès classiques sur les multiordinateurs et les SDDS basées-arbres qui représentent l'élargissement des structures classiques d'accès ordonné sur un environnement distribué.

Ces SDDS ont été conçues initialement pour les fichiers distribués qui ont la caractéristique d'être scalables pour mieux les gérer et bien maintenir cette scalabilité. Les bases de données ont profité eux aussi de ces structures. D'ailleurs plusieurs projets (cités dans ce mémoire) les ont utilisé. Ainsi ces structures peuvent être considérées comme une alternative aux structures classiques pour ce qui est des environnements distribués.

L'objectif de ce mémoire étant la démonstration de la possibilité et les perspectives d'utilisation de ces nouvelles structures de données dans l'implémentation du niveau physique d'une base de données relationnelle distribuée.

Pour ce but, nous avons essayé d'expliquer les différentes notions des bases de données et du modèle relationnel en premier chapitre, les notions de multiordinateur et de réseaux peer to peer en deuxièmes chapitre. Dans le troisième chapitre les nouvelles structures de données sont étudiées pour aboutir au quatrième chapitre qui expose l'ensemble des projets réalisés dans les domaines des SDDS, en relation avec les réseaux p2p et les SGBD, et enfin, le cinquième chapitre où l'on démontre la possibilité de l'utilisation des SDDS dans le niveau physique d'une BDR distribuée sur un réseau multiordinateur.

Malgré que cette étude demeure strictement théorique, elle nous renseigne que les caractéristiques de la nouvelle structure de données proposée (SDDS) lui permettent d'être utilisable dans la plupart des domaines qui emploient des quantités importantes de données distribuées pouvant être gérées par un SGBD relationnel. Ainsi en utilisant ces structures, le niveau physique du SGBDR sera plus commode en terme de stockage distribué, d'accès parallèle et concurrent et aussi de manipulation de requêtes SQL. Et comme perspective de ce travail, nous pensons réaliser une étude plus détaillée et matérialisée sur l'utilisation de ces structures dans le niveau physique d'un ou de plusieurs SGBD relationnel (s) connu (s), pour mieux l'affirmer.

Bibliographie

[1] M.Mosbah, "*Modèles et approches formels pour les systèmes distribués*", support de cours, http://dept-info.labri.fr/~mosbah/cours.pdf.

[2] Witold Litwin, "*Structures de données distribuées et scalables*", support de cours, http://ceria.dauphine.fr/cours98/coursBD/sdds.ppt.

[3] Boukhelef Djelloul, "*IH* hachage multidimensionnel distribué et scalable*", thèse de PHD, Institut National D'informatique (INI) Alger, http://ceria.dauphine.fr/visiteur/demo.ppt

[4] Sylvain Brandel, "*Bases de données réparties*", support de cours, Université Louis-Pasteur de strasbourg, 2003-2004, http://www710.univ-Lyon1.fr/sbrandel/ enseignement/archives2006-2007/supports/BDRsuport.pdf.

[5] Georges Gardarin, "*Bases de données*", édition Eyrolles, Sixième tirage, 2005.

[6] Y.B. Ndiaye, "*Mise en oeuvre de l'architecture de communication des structures de données distribuées et scalables RP*$_N$ et RP*$_C$*", mémoire de DEA, Université Cheikh Antat rop de Dakar, 1998.

[7] Rim Moussa, "*Contribution à la conception et l'implantation de la structure de données distribuée & scalable à haute disponibilité LH*RS*", thèse de doctorat, Université Dauphine (Paris), 2004.

[8] T.Mianakirdila, "*Mesure des performances de la scalabilité des requêtes parallèles sur les SDDS RP**", mémoire de DEA, Université Cheikh Antat rop de Dakar, 2000.

[9] Abdeslam Boularias & Belaid Saad, "*Implémentation et étude des performances de la SDDS Compact Trie Haching (CTH*) sous Windows 2000*", thèses d'ingéniorat, Institut National D'informatique (INI) Alger, 2004.

[10] A.W.Diène, "*Organisation interne des SDDS RP**", mémoire de DEA, Université Cheikh Antat rop de Dakar, 1998.

[11] W.Litwin, M.Neimat et D.A.Schneider, "*LH*- a scalable, distributed data strucure*", http://portal.acm.org/citation.cfm?id=236713.

[12] Salman Zubair Toor, "*LH* Algoritm: scalable distributed data structure (SDDS) and its implementation on swithches multicomputers*", support de cours, http://user.it.uu.se/torer/kurser/mdb/2007/termpapers/salmantoor.pdf

[13] Y.Ndiaye & A.W.Diène, "*couplage de structures de données distribuées et scalables avec un SGBD relationnel-Objet*", http://user.it.uu.se/udbl/pub/amos_sdds.pdf.

[14] W. Litwin, "*Linear hashing: A new tool for file and table addressing*", Proceeding de VLDB, 1980, http://www.cs.cmu.edu/afs/cs.cmu.edu/user/christor/www/courses/826-resources/papers+book/linear-hashing.pdf.

[15] W.Litwin & Fethi Benour, "*Scalable distributed data structures & high-performance computing*", http//ceria.dauphine.fr/aurora.ppt.

[16] Dijoux Alexandre & Emma Samuel, "*Peer-to-peer*", master professionnel système informatique et réseaux, Université Claude Bernard (Lyon1), 2006-2007, http://master-info.univ-Lyon1.fr/M2SIR/2006/peer2peer.pdf

[17] George Gardarin, "*Mediation p2p*", http://georges.gardarin.free.fr/cours_mediationp2p/p2p2006.ppt

[18] http://fr.wikipedia.org/wiki/pair-Ã-pair

[19] Gabrielle Feltin, Guillaume Doyen & Olivier Festor, "*Les protocoles peer to peer, leur utilisation et leur détection*", Loria, 2003, http://2003.jres.org/actes/paper.70.pdf

[20] Lawrence Lessig, "*Peer to peer*", livre édité par andy oram, 1ère edition; 2001.

[21] Stephanos Androutsellis-Theotokis & Diani Dis Spinellis, "*A survey of peer to peer content distribution technologies*", article du proceeding ACM computing surveys, vol 36 N°4 pp335-371, décembre 2004,

[22] Julien Laflaquière, "*Les autres applications des technologies peer to peer*", article mis en ligne le 25/06/2005 sur multitudes, http://multitudes.samizdat.net/spip.php?article1979.

[23] Fausto Giunchiglia & Ilya Zaihrayeu, "*Implementing database coordination in p2p networks*", rapport technique#DIT-03-035, Université de Trento (Italie), novembre 2003.

[24] D'après stratégie Télécom & multimédia, "*DT 29 - Grille de calcul, P2P, applications et problématique*" ; 23 mai 2008, http://www.strategiestm.com/DT-29-Grille-de-calcul-p2p.html.

[25] Tinos Sellis & Yannis Vassiliou, "*Data management research at the knowledge and database system lab (NTV Athens)*", article du proceeding SIGMOD record, vol 35 N°2, juin 2006.

[26] Karl Aberer & Maufred Hauswirth, "*An overview on peer to peer information system*", article extrait du tutorial (*peer to peer information system*) publié à ICDE en 2002.

[27] David Del Vecchio & Sang H.Son, "*Flexible update management in peer to peer database systems*", http://www.cs.virginia.edu/papers/vecchio-p2p-IDEAS05.pdf.

[28] http://liris.cnrs.fr/nicolas.lumineau/p2p.html

[29] Boubaker Monia, Letion Ludavic & Pierron Aurelien, "*Peer to peer-kazaa, freenet-jxta*", fevrier 2005, http://www-igm.univ-mlv.fr/~duris/NTREZO/20042005/boubaker-lelion-pierron-p2p.pdf.

[30] Sacha Krakowiak, "*Gestion repartee de données-2, diffusion à grande échelle, techniques p2p*", exposé de master en système et logiciel, école doctorale de Grenoble, http://proton.inrialpes.fr/~krakouvia/enseignement/M2P-GI/lips/gestion-donnees-2.pdf.

[31] Samuel Dalens, Vicent Lepage & En Lin, "*Routage dans les réseaux peer to peer*", http://www-poleia.lip6.fr/~sarr/GRDBD/routagep2pdalenstlepageen-1.pdf.

[32] http://www.commentcamarche.net.

[33] http://fr.wikipedia.org/wiki/RÃ©seau_informatique.

[34] Angela, P.A.Boncz, P.Janacik, B.König-Ries, A.Illarramendi, W.Lehner, P.J.Marron, W.May, A.Ouksel, K.Rômer, B.Sapkota, K.U.Sattler, H.Schweppe, R.Steinmetz & C.Türker, "*Working group report on managing and integrating data in p2p databases*", article du proceeding de seminaire daystuhl N°6431 sur (scalable data managment in evolving network), Allemagne, 23-27 octobre 2007.

[35] B.Angela, E.Q.Chang, L.Lokshmanon, T.Ho & R.Pottinger, "*HePTox: marrying XML and heterogeneity in your p2p databases*", article du proceeding de la 31ème conférence VLDB, Trondheium, Norway, 2005.

[36] Albena Roshelova, "*A peer to peer database management system*", rapport technique#DIT-04-057, Université de Trento (Italie), juin 2004.

[37] Ilya Zaihrayeu, "*Query answering in peer to peer database networks*", rapport technique#DIT-03-012, Université de Trento (Italie), mars 2003.

[38] Fausto Giunchiglia & Ilya Zaihrayeu, "*Marketing peer to peer databases interact-avision for an architecture supporting data coordination*", rapport technique#DIT-02-0012, Université de Trento (Italie), juin 2002.

[39] Enrico Franconi, Gabriel Kuper, Endrei Lopatenko & Ilya Zaihrayeu, "*The CODB robust peer to peer database system*", rapport technique#DIT-04-028, Université de Trento (Italie), avril 2004.

[40] Ilya Zaihrayeu & Fausto Giunchiglia, "*Coordinating mobile databases: a system demonstration*", rapport technique#DIT-04-030, Université de Trento (Italie), mai 2004.

[41] Enrico Franconi, Gabriel Kuper, Andrei Lopatenko & Ilya Zaihrayeu, "*Queries and updates in the CODB peer to peer database system*", rapport technique#DIT-04-088, Université de Trento (Italie), août 2004.

[42] Albena Roshelova, "*Bulding database coordination in p2p system using ECA rules*", rapport technique#DIT-04-002, Université de Trento (Italie), juin 2004.

[43] Tore MØrkved, "*Peer to peer programming with wireless devices*", thèse de master en technologie d'information et de communication, Université de New South Wales (Sydney-Australia) & agder university college (Grinstad-Norway), juin 2005.

[44] www.techniques-ingenieur.fr

[45] Georges Gardarin, "*Hachage et indexation*", résumé de chapitre, 2002, http://www.georges.gardarin.free.fr/Cours_BD/8-Hachage.ppt.

[46] Brahim Belattar, "*Bases de données à l'usage de l'étudiant*", support de cours, Université de Batna, Algérie, http://www.4shared.com/file/40937324/1fd587ec/cours_bases_des_donnes_-_dr_brahim_belattar_univ_batna.html?s=1.

[47] J. Darmont, " *Bases De Données* ", support de cours,CNAM, Centre associé de Clermont-Ferrand,1997-1998, http://eric.univ-lyon2.fr/~jdarmont/docs/old/1-introduction.pdf.

[48] " *Stockage des Données : Disques et Fichiers, Chapitre9 ",* http://www.site.uottawa.ca/ékiringa/courses07/csi2532/ch9_disks_files_csi2532-07.ppt.

[49] " *Chapitre 7 : L'organisation physique des relations dans un SGBD* ", http://medias.obs-mip.fr/www/activite/formation/informatique/conccept/chap7.htm.

[50] " *Fichiers* ", http://lbdwww.epfl/f/teaching/courses/slidesIBD/fichiers.pdf.

[51] http://eric.univ-lyon2.fr/éjdarmont/docs/old/8-physique.pdf.

[52] Robert Godin, "*Système de gestion de bases de données*", volume1, édition Loze-Dion, 2000.

[53] G. Moulard, " *SGBD* ", support de cours, http://guillaume.moulard.free.fr/GM-SGBD-V1.pdf.

[54] Y. Pollet, " *Systèmes de gestion de bases de données* ", http://pagesperso-orange.fr/jchiesa/supports/bdd/pollet1.ppt.

[55] " *Concepts des bases de données relationnelles* ". http://lifc.univ-fcomte.fr/~lasalle/bda/COURSHTM/cours/chap01/lec01.

[56] E. F Codd, " *A relational model for large shared data banks* ", communication de l'ACM, vol 13, n°6, juin 1970, p 377-387.

[57] http://www-lsr.imag.fr/les.personnes/herve.martin/HTML/Le modèle relationnel.htm.

[58] G.D. Knott, "*Expandable open addressing hash table storage and retrival*", ACM SIGFIDET Workshop on data description, access and control, 1971, ACM Ed, p.186-206.

[59] R. Fagin, J. Nivergelt,N.Pippengar, H.R.Strong, " *Extendible haching-A fast access method for dynamic files*", ACM TODS, vol 4, n°3, septmebre 1979, p 315-344.

[60] P. Larson, " *Dynamic haching* ", journal BIT, n°18, 1978, p 184-201.

[61] W. Litwin, " *Virtual haching : A dynamically changing haching* ", 4th very large data bases, Berlin, septembre 1978, p 517-523.

[62] P. Larson, " *Linear haching with partial expansions* ", 6th very large databases, Montreal, october 1980.

[63] W.Litwin, " *Linear haching – A new tool for file and table adressing* ", 6th very large data bases, Montreal, octobre 1980.

[64] W.Litwin, "*SDDS-2000: A Prototype System SDDS-2000: A Prototype System for Scalable Distributed Data for Scalable Distributed Data Structures on Windows Structures on Windows 2000*", présenté à :Université North-Western, Evanston & Microsoft, Redmond, 2001, http://ceria.dauphine.fr/SDDS-chicago.ppt.

[65] Y.Ndiaye, A.W.Diène, W.Litwin & T.Risch, "*Scalable Distributed Data Structures for High-Performance Databases*", http://user.it.uu.se/~udbl/publ/amos_sdds.pdf.

[66] Y.Ndiaye, "*Interopérabilité d'un Serveur de Structures de Données Distribuées et Scalables et d'un SGBD Relationnel-Objet*", 2001, http://ceria.dauphine.fr/Presentation_yn_fr.ppt.

[67] W.Litwin & G.Levy, "*Access Algorithms and Data Structures Underlying a Distributed Knowledge Base*", Rapport technique du projet (*Intelligent Content Management System*), CERIA, 2003, http://www.icons.rodan.pl/docs/ICONS_WP5_T1_D18_0105.pdf.

[68] H.Yakouben, "*LH*rsP2P : une nouvelle Structure de Données Distribuée et Scalable pour un environnement pair à pair*", 2006, http://ceria.dauphine.fr/YakoubenHomePage/ présentation%20LHrsP2P. ppt.

[69] S.SAHRI, " *Conception et Implantation d'un Système de Bases de Données Distribuées & Scalables: SD-SQL Server* ", thèse de doctorat, Université Paris Dauphine, 2006, http://ceria.dauphine.fr/soror/these.pdf.

[70] Yakham Ndiaye, Aly W. Diene & Fethi Bennour, " *Couplage de Structures de Données Distribuées et Scalables avec un SGBD relationnel-objet* ", http://ceria.dauphine.fr/AmosSdds.pdf.

[71] Witold Litwin, "*Structures physiques*", support de cours, http://ceria.dauphine.fr/cours98/coursBD/strph.ppt.

[72] *"Faut-il interdire le Peer to Peer?", http*://www.chez-breizh.fr/news/indispensable.html.

[73] DIJOUX Alexandre & EMMA Samuel, "*Peer-To-Peer",* Master Professionnel Système informatique et réseaux, Université Claude Bernard LYON 1, 2007.

[74] http://hautrive.free.fr/reseaux/architectures/organisation-des-reseaux.html.

[75] http://www.commentcamarche.net/contents/cs/csintro.php3.

[76] http://fr.wikipedia.org/wiki/Client-serveur.

[77] http://www.vulgarisation-informatique.com/peer-to-peer.php.

[78] http://www.commentcamarche.net/contents/initiation/peer.php3.

[79] Guy Pujolle, Ahmed Serhrouchni, Marc-Michel Pic, Anne-Gaëlle Geffroy, Olivier Bomsel, Pierre Sirinelli & Frédéric Goldsmith, *Livre Blanc sur le " Peer-to-Peer ",* Syndicat National de l'édition phonographique (SNEP), Octobre 2007.

[80] http://laurent-audibert.developpez.com/Cours-BD/html/Cours-BD004.html.

[81] D. Herman, P. Burgevin & P. Fresnais, *" Exploitation d'une base de données",* chapitre 6 du document *" Bases De Données, Une Introduction Au Modèle Relationnel ",* Université de Rennes 1-Ifsic, http://perso.univ-rennes1.fr/daniel.herman/Editions-des-noisettes-et-des-sentiers/Bases-de-donnees/Intro.../bd5-exploitation.pdf.

[82] Stéphane Schildknecht, *" Haute disponibilité et bases de données",* le portail francophone des services autour de PostgreSQL, 07/2005.

[83] *" continuité et disponibilité ",* http://www.nec-computers.fr/269/solutions/continuite-disponibilite.html.

[84] *"Chapitre 10. Transactions et accès concurrents ",* http://www.dil.univ-mrs.fr/~massat/docs/hibernate-2/reference/fr/html.

[85] *" SQL Server 2005 : haute disponibilité et hautes performances ",* le magazine du développeur : programmez, paris, octobre 2007.

[86] Jean-Paul Argudo, *"PostgreSQL et la haute disponibilité ",* rapport de créativité Commons, BY-NC-SA, Avril 2009.

[87] M.Leonard, *" Structures des bases de données* ", édition DUNOD Informatique, Paris, 1988.

[88] W.Litwin, " *Scalable Distributed Data Structures, Part 2* ", 2010, http//ceria.dauphine.fr/ SDDS-10-partie2-09-10.pptx.

[89] http://mediasfrance.org/Activite/Formation/Informatique/concept/chap8.htm

[90] www.sciencedirect.com

[91] Laurent Audibert, "*Base de Données et langage SQL"*, support de cours, Institut universitaire de Technologie de Villetaneuse – Département Informatique, www.system- *deptinfo.unice.fr/~**grin**/messupports/**sql**.pdf* linux.net/config/COURS...MYSQL/MySQL/Cours-BD.pdf

[92] Richard Grin, *" Langage SQL "*, version 5.7 du polycopié, support de cours, Université de Nice Sophia-Antipolis; janvier 2008, http://deptinfo.unice.fr/~grin/messupports/sql.pdf

[93] Hamri Amina, Kabla Imene, *" Opérations Algébriques de base sur la Structure de donnéesdistribuée et scalable Distributed Partitioned Binary Search Tree (PBST*) "*, MEMOIRE D'INGENIEUR, Institut National d'Informatique, JUIN 2008,

http://share.esi.dz/index.php?option=com_docman&task=doc_download&gid=257&Itemid=1

[94] *"Gestion de données Réparties"*
http://www.lirmm.fr/~libourel/MIFPRU/M2/COURS1-Integration08.pdf

[95] Rim Moussa, *"Systèmes de Gestion de Bases de Données Réparties & Mécanismes de Répartition avec Oracle"*, Ecole Supérieure de Technologie et d'Informatique à Carthage, 2006-2007,
http://ceria.dauphine.fr/Rim/Teaching_fichiers/BDR/BDR07.pdf

[96] Marc Lemaire, *"Modèle relationnel et langage SQL"*, novembre 2007
http://www.u-cergy.fr/rech/pages/lemaire/pdf/poly_sql.pdf

[97] J.S. Karlsson, M.L. Kersten, *"Scalable Storage for a DBMS using Transparent Distribution",* INS-R9710 December 31, 1997
http://oai.cwi.nl/oai/asset/4704/04704D.pdf

[98] Walid-Khaled Hidouci, *"Vers une approche des transactions avancées dans les SGBD parallèles integrant les modèles acteur et SDDS (scalable distributed data tructures) "* ; thèse de doctorat, institut national d'informatique (INI), Alger, juillet 2007,
http://hidouci.esi.dz/these_hidouci_2007.pdf

[99] Witold Litwin, Soror Sahri, Thomas Schwarz, *"SD-SQL Server: Scalable Distributed Database System"*, Ceria Res. Rep. 2005-12-13, December 2005
http://ceria.dauphine.fr/soror/sd-sql-serverResRep3.pdf

[100] W. Litwin, T. Risch, Th. Schwarz, *"An Architecture for a Scalable Distributed DBS: Application to SQL Server 2000"*, (Extended Abstract), 2nd Intl. Workshop on Cooperative Internet Computing (CIC 2002), August, 2002, Hong Kong, http://ceria.dauphine.fr/sd-sql-server2_tr3.pdf

[101] Witold Litwin, Cédric du Mouza & Philippe Rigaux, *"SD-Rtree: A Scalable Distributed Rtrees"*, http://ceria.dauphine.fr/cours98/CoursBD/sd-rtree2-wl.ppt

[102] D.E Zegour, Pr W.K Hidouci, *"Towards a complete scalable distributed data structure"*, institut national d'informatique (INI) -Alger, Avril 2009
http://zegour.uuuq.com/Ftp/Sdm.pdf

[103] D.E Zegour, "*Scalable Distributed Compact Trie Hashing (CTH*)"*, Institut National D'informatique (INI) -Alger,
http://ceria.dauphine.fr/visiteur/CTH%20%20IST.pdf

[104] Rim Moussa, *"Implantation Partielle & Mesures de Performances de LH*RS : Une SDDS à Haute Disponibilité",*
http://ceria.dauphine.fr/Rim/Publications_fichiers/DEA00.pdf

[105] Witold Litwin, Rim Moussa, Thomas Schwarz, *"LH*RS—A Highly-Available Scalable Distributed Data Structure"*, Universités: Paris Dauphine & Santa Clara, ACM Transactions on Database Systems, Vol. 30, No. 3, September 2005, Pages 769–811.

[106] M. Aridj, D.E Zegour, *"HD*: une nouvelle et rapide structure de données distribuée et scalable ordonnées"*
http://www.univ-chlef.dz/seminaires/seminaires_2009/aridj_setif.pdf

[107] Benmoussa Yahia, *"Etude Comparative Entre les Schémas de Structures de Données Distribuées et Scalables DRT et DCTH"*, 2003/2004
http://share.esi.dz/index.php?option=com_docman&task=doc_download&gid=54&Itemid=1

[108] Idrissa SARR , *"Traitement des transactions SQL dans un environnement pair à pair : Algorithme pour équilibrer la charge des mises à jour"*, thèse de DEA, université de DAKAR, Fevrier 2006
http://www-poleia.lip6.fr/~sarr/publication/dea_idy.pdf

[109] Andy D. Pimentel and Louis O. Hertzberger, *"Evaluation of LH*LH for a Multicomputer Architecture"*, University of Amsterdam Kruislaan 403, 1098 SJ Amsterdam, The Netherlands Appeared in Proc. of the EuroPar 99 Conference, Toulouse, France, September 1999.

[110] *"Mémoire partagée distribuée pour systèmes dynamiques à grande échelle"*, Thèse , Université de Rennes 1, 2007
ftp://ftp.irisa.fr/techreports/theses/2007/gramoli-f.pdf

[111] N. Lounes, *"RAP_ACT : reprise après pannes dans le SGBD parallèle a base d'acteurs ACT21"*, Juillet 2005
http://share.esi.dz/index.php?option=com_docman&task=doc_download&gid=68&Itemid=1

[112] Serge Abiteboul, *"Évaluation et optimisation de requêtes"* à partir des transparents de Philippe Rigaux, Dauphine, INRIA Saclay, Avril 2008
http://www-rocq.inria.fr/~abitebou/DBCOURSE/optimization.pdf

[113] Michel Lemaître, *"Structures de Données et Algorithmes"*, notes de cours (partie 1) ONERA, Centre de Toulouse, document mis à jour le 24 Octobre 2001,
http://www.scribd.com/doc/45689214/polySA

[114] http://www.ehs.washington.edu/forms/rbs/rpha.doc

[115] Dmitry Shaporenkov, *"RP*ha - a High-Availability Scalable Distributed Data Structure Based on Range Partitioning"*, Saint-Petersburg State University,
http://lvk.cs.msu.su/files/mco2003/shaporenkov.pdf

[116] Riad Mokadem, *"Stockage Utilisant Les Signatures Dans Les Bases De Données Distribuées Et Scalables"*, septembre 2002, http://ceria.dauphine.fr/Riad%20mokadem/Rapport_RiadDEA.pdf

[117] Imadali Sofiane, Chabane Farid, *"Système transactionnel dans PBST*"*, 2010, http://share.esi.dz/index.php?option=com_docman&task=doc_download&gid=365&Itemid=1

[118] Philippe Rigaux, *"Cours de bases de données"*, juin 2001 http://www.info.univ-angers.fr/~gh/internet/cbd.pdf

Résumé

La technologie informatique a connaît une évolution importante dans le domaine des bases de données, surtout avec l'apparition des réseaux informatiques qui ont permis d'élargir et de faciliter la communication, le traitement et également le stockage des données.

En effet, un nouveau concept est né, celui de multiordinateur, permettant d'offrir des capacités gigantesques de stockage et de calcul parallèle. Les applications actuelles n'arrivent pas à tirer profit de ces capacités. Pour cela, une nouvelle structure de données dénommée Structure de Données Distribuées et Scalables (SDDS) est apparue. Il s'agit de fichiers résidants en mémoire vive distribuée permettant un accès beaucoup plus rapide que les structures de données classiques sur des disques. Ainsi elle permet d'assurer la disponibilité des données, leur distribution et leur scalabilité (maintien des performances en cas d'accroissement du volume de données stockées).

Les SDDS ont pratiquement des applications plus ou moins variées, leur utilisation par les SGBD semble très importante du fait que cette structure supporte le traitement parallèle et permet le partitionnement dynamique des données de la base. Cette structure a marqué aussi son importance pour le stockage et le traitement des requêtes.

Ainsi, le but de ce mémoire est de démontrer la possibilité et les perspectives d'utilisation de ces nouvelles structures de données dans le niveau physique d'une base de données relationnelle distribuée.

ملخص

عرفت تكنولوجيا المعلومات تطور كبير في مجال قواعد البيانات, بالأخص مع ظهور الشبكات المعلوماتية التي سمحت بتوسيع و تسهيل الاتصال, المعالجة و حفظ البيانات.

كل هذا أدى الى ظهور مفهوم جديد سمي بـ: multiordinateur و الذي يسمح بتقديم قدرات ضخمة فيما يخص حفظ البيانات و المعالجة المتوازية لها, و لأن التطبيقات الحالية لا تستطيع استغلال هذا الكم الهائل من الموارد ظهر نوع جديد من هياكل البيانات سمي بـ : Structure de Données Distribuées et Scalables (SDDS)و الذي يتمثل في مجموعة من المستندات (fichiers) المقيمة في الذاكرة الحية المتوزعة (DRAM) فهي تسمح بالوصول للمعلومة بسرعة أكبر بكثير من سرعة الوصول إليها في حالة استعمال الهياكل التقليدية المخزنة في القرص الصلب. و بالتالي فهي تسمح بضمان استيداع البيانات و توزعها بالإضافة إلى إمكانية تصاعدها (أي الحفاظ على النتائج القياسية في حالة الزيادة المعتبرة لمقدار البيانات المخزنة).

في الواقع هذه الهياكل الجديدة (SDDS) تستعمل من طرف تطبيقات متنوعة نوعا ما, فاستعمالها من طرف أنظمة تسيير قواعد البيانات (les SGBD) مهم جدا إذ أنها تسمح بالمعالجة المتوازية و التوزيع الديناميكي لبيانات القاعدة, كما أنها أثبتت فعاليتها فيما يخص التخزين وكذا معالجة الطلبات (les requêtes).

إذن, الهدف من هذه المذكرة هو توضيح إمكانية وأهمية استعمال هذه الهياكل بالنسبة للمستوى البنيوي (الداخلي) لقاعدة بيانات ترابطية متوزعة.