République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique.

Mémoire pour l'obtention
d'un diplôme d'ingénieur d'état en informatique.

Option : Système d'information

THEME :

REALISATION D'UN SERVEUR POUR LA SDDS IH

Promoteur : M[r] Boukhelef D

Présenté par :

M[elle]. Benmalek. F
M[elle]. Bouali. S

Organisme d'accueil : département d'informatique Blida.

Précédente de jurys : M[e] Bensttiti S

Membre de jurys :

- Promotion2004/2005-

Dédicace

Je dédie ce modeste ouvrage :

A ma très chère maman, j'espère qu' elle sera toujours fière de moi.

A mes grands parents qui m'ont encouragé et aidé avec leur prière.

A mes oncles Diab, Mourad, Sassi, Abdelazize, Taher qui ont toujours aidé et encouragé.

A mes tantes Halima, Sabrina, Zola,

A mes frères Larbi, Adel, Bouzid, Sofianne.

A ma sœurs fouzia.

A les poussins Imane, Thamer,Marwa, Moataze, Mohsine

A toutes mes amies Bahia, Samira, Nadia, Moufida, Hanane, Souad, Djawida, Dalila, Siham, Hana, Hafida, nadjia

A tous mes camarades d'étude et surtout Tina, Wafa, Souad

A Liela pour les bons moments qu'on a passé ensemble et à sa famille.

A tous ceux que j'aime et qui m'aiment.

B.FAHIMA

Dédicace

Louanges, mérite, gratitude, reconnaissance en reviennent à Dieu tout puissant que son nom soit béni et exalté.

Je dédie ce modeste travail :

A mes très chers parents que j'aime profondément pour leur dévouement et leur intérêt constant. Que dieu tout puissant vous protége, et vous accorde longue vie.

A mon fiancé M'hamed qui n'a cessé de m'encourager tout au long de mon projet.

A mon frère unique : khaled pour leurs attentions et leurs encouragements.

A ma chère et unique sœur lynda, sans oublier son mari, et surtout leur nouveau né anès.

A la femme que je ne pourrais assez exprimer mon éternel amour, respect et gratitude : la mère de M'hamed ainsi que ses frères, ses sœurs et leur petit poussin Fethi.

A mes amis : Darine, Nawel, Badra, assia pour leur soutient moral.

A toute ma famille paternelle et maternelle.

Tous mes remerciements a mon très cher amie et partenaire : Ben malek Hayet.

Et enfin à tous qui me sont chers.

SOUAD

REMERCIEMENTS

Nous remercions avant tout Dieu d'avoir éclairé notre route et nous avoir donné la force, le courage et la volonté qui ont permis de réaliser ce travail.

Nous tenons à exprimer notre profonde gratitude et nos respectueux remerciements au chef de département et à tous les professeurs du département d'informatique de l'Université de Blida.

Nos vifs remerciements vont à Madame la Présidente et aux membres du jury d'avoir accepté d'évaluer notre travail.

Nous tenons à remercier notre promoteur Mr BOUKHELEF pour son suivie et ses orientations.

Nous tenons à exprimer nos vifs remerciements à tous ceux qui, par leurs travaux, leurs idées, leurs présentations, leurs collaborations ou leurs relectures, ont participé de près ou de loin à la réalisation de ce mémoire.

Une mention particulière à Mr SAIFI ABDELHALIM qui nous a aidées et encouragées pendant la réalisation de ce travail.

Résumé

Les Structures de Données Distribuées et Scalables (SDDS) sont une nouvelle classe de structures introduites vers 1993 spécifiquement pour la gestion de fichiers sur un multiordinateur. Un fichier SDDS peut s'étendre dynamiquement, au fur et à mesure des insertions, d'un seul site de stockage à n'importe quel nombre de sites. Les algorithmes d'adressage d'une SDDS sont conçus spécifiquement pour être scalables, notamment par absence d'un répertoire ou index central. La répartition des données est transparente pour l'application. Les données sont aussi stockées de préférence en RAM distribuée. Plusieurs SDDS ont été proposées. Les plus connues sont celles basées sur le hachage linéaire (LH*) et celles utilisant le partitionnement par intervalle (RP*). Notre contribution dans le domaine des SDDS s'est concrétisé par la construction de la SDDS IH* (scalable distributed interpolation-based hashing) présentée dans ce mémoire.

Le SDDS IH* qui constitue le fruit d'une sérieuse investigation dans le domaine des SDDS, est la première SDDS a base d'une technique de hachage multidimensionnelle.

Cette SDDS est une adaptation du hachage par interpolation aux environnements distribués. elle peut également être vue comme étant une extension multidimensionnelle, avec préservation de l'ordre, de la SDDS LH*.

Notre projet a pour objectif la conception d'un serveur pour la SDDS IH*. Ce système fonctionne sur un réseau d'ordinateurs. Le nombre de sites pour un fichier SDDS n'est pas limité.

Mots clés : Multiordinateur, Scalabilité, Structure de Donné Distribué et Scalable.

Abstract

The Scalable and Distributed Data Structures (SDDS) are a new class of data structures proposed specifically for this purpose. An SDDS file is distributed over the *server* nodes. The number of nodes dynamically scales with the file growth. The application accesses the file through a *client* node that makes the data distribution transparent. For scalability, the data address calculations do not involve any centralized directory. The data are also basically stored in the distributed RAM, for faster access than to the traditional disk-based structures. Several SDDS schemes have been proposed. The most studied are the LH* schemes for hash partitioning and the RP* schemes for the range partitioning.

This project elaborates on the design and implementation of a prototype SDDS manager, called IH*. The system runs at networks of LINUX computers. The number of sites for an SDDS file is limited only by that accessible to the system.
After introduction and the discussion of the state-of-the-art, we present the distributed architecture of our system.

Key words: *Multicomputer, scalability, scalable distributed data structures, parallel processing.*

Sommaire

Chapitre I - systeme distribuées

Chapitre II – Hachage par interpolation distribue et scalable

Chapitre III- Protocole de communication pour la SDDS IH* et l'architecture du systeme

Table des Figures

Chapitre I

Chapitre II

Chapitre III

Annexe

Introduction générale

Ces dernières décennies ont vu des avancées considérables de la technologie des circuits a semi-conducteurs. L'évolution est extrêmement rapide dans ce domaine au niveau de leur architecture et de la capacité d'intégration des circuits. Une conséquence directe est l'augmentation forte et régulière de la taille de la mémoire centrale et de la puissance de calcul des ordinateurs. Contrairement aux disques durs que leur évolution n'est pas vraiment importante, au sens de temps d'accès, qui est lié à des contraintes mécaniques. Les réseaux ont aussi connu des développements importants ces dernières années. Ces réseaux grâce à leur vitesse de transfert (10-100 Mb/s), réduisent le temps de communication et permettent aux utilisateurs de partager l'ensemble des ressources réparties. Ces deux derniers paramètres ont changé notre façon d'utiliser les ordinateurs et a suscité un intérêt certain pour la gestion globale de la mémoire physiquement distribuée [BB04].

Actuellement, il est plusieurs fois plus rapide d'accéder à une page mémoire sur une machine distante qu'au disque local. Les réseaux de PCs et de stations de travail sont aussi devenus omniprésents dans les entreprises. Ils offrent des capacités cumulées de stockage (disques et mémoires RAM) et de calcul très importantes, offertes jusque-là seulement par le matériel spécialisé et donc fort cher. Ces capacités deviennent notamment beaucoup plus importantes que celles de superordinateurs[1], à une fraction de coût de ces derniers. Cette évolution a donné naissance à de nouveaux concepts architecturaux. On parle souvent de multiordinateur réseau ou de Réseau de Stations de Travail (Network of Workstation) [ACP95].

Grâce à leur potentiel de stockage et de calcul parallèle et au rapport qualité/prix imbattable, les multiordinateurs deviennent de plus en plus importants. Leurs performances ont permis l'apparition de plusieurs projets au niveau de la recherche en informatique s'intéressant à produire des logiciels permettant de les utiliser à pleine capacité.

Ces recherches ont touché les SGBDs, les serveurs multimédia, les serveurs Web, les applications de calcul hautes performances et les systèmes de fichiers distribués (SFD); en particulier, les structures de données distribuées. Tous ces systèmes et d'autres encore, ont besoin de volumes de données et de puissance de calcul croissants.

[1] Machines multiproccesseurs (supercalculateur).

La recherche sur les multiordinateurs a montré notamment le besoin de nouveaux algorithmes et structures de données. Les Structures de Données Scalables et Distribuées ou SDDSs (Scalable and Distributed Data Structures) ont été introduites dans ce but [LNS93.a, LNS93.b]. Les données d'une SDDS sont réparties dans la mémoire distribuée d'un multiordinateur. Le fichier SDDS s'adapte dynamiquement à l'accroissement du volume de données. Il s'étend de manière progressive d'un seul site à théoriquement n'importe quel nombre de sites.

Le placement de données et son évolution sont transparents pour les applications. Celles-ci appellent en effet les clients SDDS qui gèrent l'accès aux serveurs comme s'il s'agissait de structures de données classiques. Les SDDSs supportent néanmoins le traitement parallèle et assurent potentiellement par leur conception des temps d'accès aux données beaucoup plus courts que ceux aux fichiers traditionnels sur disques.

Plusieurs SDDS ont été proposées. Notamment celles basées sur le hachage; on trouve la famille dite LH* (Linear Hashing) et ses variantes. Cette SDDS, qui est l'extension du « hachage linéaire » aux environnements distribués, constitue le point de départ des structures de données distribuées et scalables [LNS93.a]. DDH (Distributed Dynamique Hashing) est une version distribuée du hachage dynamique.

Comme le hachage généralement ne préserve pas l'ordre des clés, les gens ont pensé à développer d'autres méthodes qui préservent l'ordre des clés; une famille dite RP* (Range Partitioning) et ses variantes est la première SDDS qui répond à ce besoin, elle D'autres types de SDDSs, basées sur le hachage, ont été proposées pour l'accès multi-clé [LN95] et la haute disponibilité [L95, LN96, KLR96, LS99, SDDS].

Objectif du travail :

Le IH* (pour scalable distributed interpolation-based haching), est la première structure de données distribuées et scalable basée sur une technique de hachage multidimensionnelle [Bou02].

Dans le cadre de la conception de nouvelles structures de données distribuées et scalables, ce mémoire se propose de développer un serveur pour la SDDS IH*.

Dans le but de confirmer les perspectives que les SDDS et les multiordinateurs peuvent offrir, notre travail consiste à Réaliser un serveur pour la SDDS IH* sous le système Windows.

La conception de cette nouvelle SDDS nous a nécessité l'étude approfondie des points suivants :

- La technologie des multiordinateurs ;
- Les systèmes distribués;
- Le modèle client /serveur et les bases de données distribuées ;
- Les protocoles de communication inter-machines, en particulier le mécanisme des sockets;

Le Plan du mémoire :

Le présent mémoire est structuré en trois (0 3) chapitres dont une introduction et une conclusion générales :

- Le premier chapitre sera consacré à l'étude des structures de données distribués et scalables. Après une introduction au concept des multiordinateurs, nous passons en revue les principes des systèmes de fichiers distribués classique et leurs inconvénients afin de positionner la nouvelle classe de structure de donnée appelée SDDS : principes, caractéristiques, types, et les principaux travaux de recherche réalisés. on a présenté aussi l'architecture du serveur SDDS ainsi que ses différentes topologies.

- Le deuxième chapitre sera réservé au hachage par interpolation qui constitue la base de notre travail. Cependant, nous ne détaillerons pas toute la méthode mais seulement ce qui permet de comprendre le principe de notre SDDS : fonction de hachage, opérations de base, éclatement, etc.

Aussi on a fait une description détaillée du principe, et caractéristique de la SDDS IH*.Nous y présentons la structure d'un fichier IH*, la fonction de hachage utilisée et les mécanismes d'adressage et d'éclatement adoptés par la SDDS IH*.

- L'objet du troisième chapitre est de concevoir un protocole de communication entre les serveurs et les clients de la SDDS IH*.

  On présente aussi dans ce chapitre l'architecture générale du prototype de la SDDS IH* implémenté. Cette architecture est constituée d'un client, d'un serveur, et d'un site coordinateur.

- Le dernier chapitre du mémoire sera consacré à la conclusion dans laquelle on parle de notre objectif qui est la conception d'un serveur pour la SDDS IH* et nous terminons par une présentation de quelques perspectives.

- En annexe nous exposons certains concepts techniques qui affectent la conception et l'implémentation des systèmes SDDS. Nous présentons dans l'annexe A, quelques définition et notation pour le hachage par interpolation. L'annexeB sera consacré à l'architecture client / serveur. Dans l'annexe C, le mécanisme de communication par socket.

Chapitre I

SYSTEMES DISTRIBUES

SYSTEMES DISTRIBUES

1. Introduction :

Des avancées majeures ont été accomplies dans les performances des réseaux (standarisation de TCP/IP, Fast Ethernet, ATM, commutateurs larges bandes etc...).Un nouveau concept est apparu: celui de multiordinateur qui est une collection de station de travail interconnectés par un réseau local haut vitesse, offrant des capacités quasi illimités de calcul, de mémoire vive et de stockage [C94]. Les applications actuelles n'arrivent pas encore à tirer profit des capacités potentielles des multiordinateurs, d'où une recherche intensive [C94], [GW96].

Les ordinateurs interconnectés par un réseau haut débit sont devenus une configuration de base dans les entreprises. On appelle une telle configuration de plus en plus souvent un multiordinateur. La recherche en multiordinateurs est désormais une direction de première importance en informatique. L'intérêt des chercheurs est motivé par le potentiel de stockage et d'accès très efficace aux données d'un multiordinateur. Ce potentiel correspond notamment en général au rapport performance/prix imbattable à l'heure actuelle. Les multiordinateurs sont en effet d'habitude composés de PC et de stations de travail de grande diffusion. Un fichier traditionnel réside entièrement sur les disques d'un site. Cette situation engendre des limitations bien connues concernant la taille, la scalabilité des performances d'accès et la vulnérabilité aux pannes du fichier [SDDS].

Pour dépasser ces limitations, des schémas distribuant les données sur plusieurs sites serveurs ont été conçus. Au début la fragmentation était statique. Puis, des solutions

dynamiques et scalaires surmontant cette limitation sont apparues. Nous y trouvons tout particulièrement les Structures de Données Distribuées et Scalables (SDDS).

Cette nouvelle classe de structures de données a été proposée pour des fichiers spécifiquement sur des multiordinateurs. Les données d'une SDDS résident de préférence en RAM distribuée. Contrairement à la RAM d'un site, celle-ci est disponible en une grande quantité. Le temps d'accès aux données peut alors être potentiellement plusieurs fois inférieur à celui d'un même fichier sur disque [BB04].

Cette nouvelle famille appelée Structure de Donnée Distribuée et Scalable (SDDS) est apparue en 1993 pour palier cette insuffisance dans le domaine de la gestion des fichiers. Les SDDSs sont conçues spécifiquement pour les multiordinateurs. Ces structures ont la potentialité d'offrir les performances de stockage et de temps de réponse impossible à atteindre pour les structures de données traditionnelles.

Plusieurs SDDSs ont été proposées [LNS93], [Dev93], [LNS94], [KW94],... Certaines sont déjà référencées dans la nouvelle édition du livre fondamental de D. Knuth "The Art of Computer Programming" paru cet été [Kar98]. Des serveurs SDDS prototypes ont été également construits. A Paris 9 Dauphine notamment, et à l'Université de Dakar; deux gestionnaires de SDDS sous Windows NT sont en cours d'implémentation.

L'architecture générale d'un système a été simplifiée par rapport aux implémentations précédentes des SDDSs, a fin de le rendre souple et de minimiser le nombre de messages encombrant le réseau et permettre au système d'évoluer au-delà de son réseau initial. A l'état initial, le système est constitué d'un ensemble de clients SDDS et d'applications qui se communiquent avec les clients pour déposer les requêtes et récupérer les réponses, et d'un serveur particulier appelé Coordinateur qui se charge de gérer la nomination de fichiers et l'allocation et libération de serveur, ainsi que les opérations de création, ouverture, fermeture et suppression de fichiers. A un moment donné, le système contient aussi un ensemble de groupes de serveurs SDDSs, chacun représente un fichier SDDS.

Les programmes client/serveur sont construits au niveau de la couche application pour permettre leur exécution sur toute plate forme supportant TCP/IP. Il existe, toute fois, trois schémas possibles pour l'architecture d'un serveur SDDS :
Serveur iteratif, serveur parallèle, serveur semi-parallèle

2. Multiordinateurs

La tendance générale dans le développement des applications informatiques est vers le traitement en ligne (on-line processing). Beaucoup d'applications informatiques requièrent une analyse rapide de grandes quantités de données. Les architectures traditionnelles traitent les données par un seul processeur ayant une mémoire centrale (RAM) et un disque comme unité de stockage secondaire. Les nouvelles architectures essayent de bénéficier des traitements parallèles et distribués utilisant des machines multiprocesseurs avec une mémoire locale par processeur ou du traitement distribués sur plusieurs sites [Y98].

Une nouvelle tendance pour le traitement distribué est les multiordinateurs, appelés aussi réseau de stations de travail, une collection faiblement couplée d'ordinateurs à grande diffusion (PC et station de travail) relie par un réseau à haut débit.

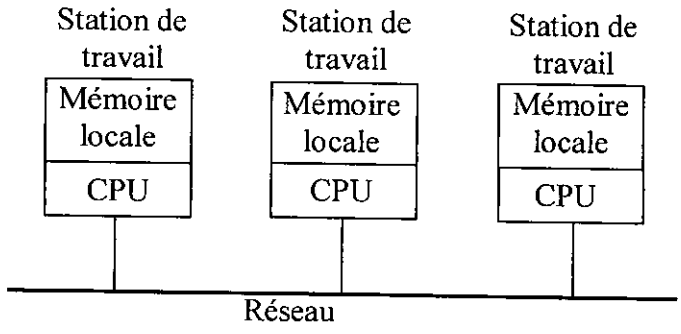


Figure I.1 : Un multiordinateur composé de stations de travail sur un LAN

La technique des multiordinateurs constitue une voie très prometteuse pour les, traitements distribués et parallèles. ceci est due, en fait, à plusieurs facteurs :

1. le premier c'est que quelque soit la capacité d'un seule processeur ou de tout un site, une collection d'ordinateur offre aux applications des capacités cumulées de stockage et de puissance de calculs quasi-illimités qui dépassent souvent celles d'un superordinateur [SDDS, LNS96].

2. le deuxième facteur est l'évolution du hardware et réseaux à haut débit (Ethernet 10mb/s, ATM, etc..), notamment la performance d'accès à la mémoire vive distante par rapport aux disques magnétiques et optiques. il est devenu plus efficace d'utiliser la mémoire distante d'un autre processeur que d'utiliser un disque local [SDDS, LNS94].

Temps d'accès aux données				
Ressource	*RAM local*	*RAM distante (gigabit Ethernet)*	*RAM distante (Ethernet 10Mbps)*	*Disque dur local*
Temps d'accès	*100 ns*	*1 µs*	*100 µs*	*10 ms*
En proportion	*1mn*	*10 mn*	*2 heures*	*8 jours*

Tableau 1. Estimation des temps d'accès aux données[Ndi98]

3. En outre, les organisations d'aujourd'hui utilisent généralement des centaines voir des milliers d'ordinateurs interconnectés par un réseau à grande vitesse, avec des dizaines de méga octets de RAM et quelques giga octets de disque par site. ce qui permet à un fichier distribué d'atteindre Facilement quelques giga octets. à titre d'exemple, les laboratoires HP ont un réseau de 1300 Stations de travail avec un total de 32 go de RAM et quelques téra octets de disque. le multiordinateur de « the Berkeley soda hall inclut 500 stations de travail délivrant 25 Gflops », 64 Go de RAM et environ 1 to de disque [LNS93,LNS94, LNS96, SDDS]. Notons dans ce cadre l'existence sur le marché du serveur IBM RS /6000 qui supporte jusqu'à 16 go de RAM [Ben00].

2.1 Caractéristiques des multiordinateurs

On considère un réseau local typique. les machines sont reliées à un câble appelé segment par Leurs contrôleurs. les segments sont reliés entre eux par des routeurs qui transmettent les messages externes. les postes envoient des message qui sont souvent découpés en paquets. chaque message est écouté par tous les contrôleurs du segment. il existe trois types de messages [BB04] :

- **Unicast :** ce message est retenu par un contrôleur unique et le délivre à son site.
- **Broadcast :** ce type de message porte une adresse reconnue par tous les contrôleurs du réseau. ainsi, tous les sites du réseau reçoivent ce message.
- **multicast :** ce type de messages porte une adresse reconnue seulement par une partie des contrôleurs du segment. Seuls les contrôleurs concernés délivrent ce message à leurs sites respectifs. les autres sites ne sont pas interrompus par leurs contrôleurs.

2.2 Performances du multiordinateur

Associons les performances du protocole (nbre de message maximal) avec les temps d'accès aux mémoire : il en résulte que 100 go de mémoire localisées sur les RAM des pc sont accessible sur un réseau Ethernet en moins d'une milliseconde.

Les temps de réponse sont fortement améliorés. L'adressage peut s'éteindre sans détérioration à des millions d'ordinateurs.

Les premières applications du multiordinateur sont les calculs à haute performance et les accès aux grandes bases de données. dans le contexte de mise en réseau généralisé crée par l'Internet, il ouvre des perspectives techniques et économiques très larges.

3. Structure de données avancées

3.1 Structures de données multidimensionnelles

Les bases de données multidimensionnelles stockent des données multidimensionnelles d'une grande variété telles que : points, lignes, région. Ces données sont représentées, sous un format vectoriel dans un espace euclidien d-dimensionnel.

3.2 Structures de données distribuées

Une nouvelle classe de structures de données a émergé cette dernière décennie. Cette classe a été baptisée SDDS : Scalable Distributed Data Structure. Elle est caractérisée par l'absence d'un site maître, l'absence de dialogue entre les clients et la scalabilité. Cette dernière caractéristique stipule que le fichier peut grandir indéfiniment sans dégradation des performances des opérations sur le fichier. Les SDDS utilisent des machines à partage de rien (nothing sharing). Ces machines constituent ce qu'on appelle un multiordinateur (réseau d'ordinateurs).

4. Structure de donnée distribué et scalable

Les recherches actuelles sont accentuées de plus en plus sur la manière de faire coopérer plusieurs micro-ordinateurs (réseau) dans le but de les rendre aussi puissants qu'un Superordinateur et ceci grâce à leur capacité de stockage cumulée et à leur traitement parallèle. Un tel système est appelé multiordinateur. Les structures de données traditionnelles ne suffisent pas pour les multiordinateurs ainsi définis à cause des nouvelles exigences suivantes : traitement distribué et parallèle, RAM distribuées, fichiers disque distribués et scalabilité. il est donc nécessaire de trouver une structure de donnée adéquate à ce nouveau type de superordinateur.

Cette dernière décennie, une nouvelle classe de structure de donnée est née totalement consacrée aux multiordinateurs. [Dev93, KW94, LNS94, KLR96] Cette classe de structure de donnée a été baptisée Scalable Distributed Data Structure (SDDS). elle est basée sur l'architecture client/serveur. Les propriétés suivantes caractérisent les SDDS :

- Distribution des blocs du fichier sur le serveur (à un taux d'un bloc par serveur),
- Absence de serveur centralisé, Aucun dialogue entre les clients. Chaque client possède une image du fichier. De cette manière, les clients peuvent faire des erreurs d'adressages. Les images des clients sont mises à jour progressivement par des messages appelés Messages de L'Ajustement de l'Image (IAMs) jusqu'à la convergence vers l'image réelle.

Un 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.

Actuellement, il existe deux classes principales de méthodes SDDS assez différents : celles qui sont basées sur LH (Linear Hashing) et celle basées sur RP (Range Partitionning). En plus de ces deux classes il existe d'autres SDDS. Il a été montré que les fichiers SDDS Peuvent être accédés beaucoup plus rapidement que les fichiers traditionnels.

Dans les SDDS (" structures de données distribuées et scalables "). Les données sont sur les serveurs; il n'existe ni répertoire central d'accès ni mises à jour synchrone des clients. Le client calcule l'adresse d'une donnée à partir d'une image locale de la structure des données. Comme les mises à jour sont asynchrones, il se peut que cette image soit obsolète. Un serveur peut donc recevoir une requête qui ne lui est pas destinée. Dans ce cas, il la route vers le serveur probablement destinataire. Le processus se poursuit jusqu'à ce que le bon serveur soit trouvé. Il envoie alors au client, outre la donnée elle-même, un message (" Image ajust message ") qui permet au client de corriger son image de la structure des données.

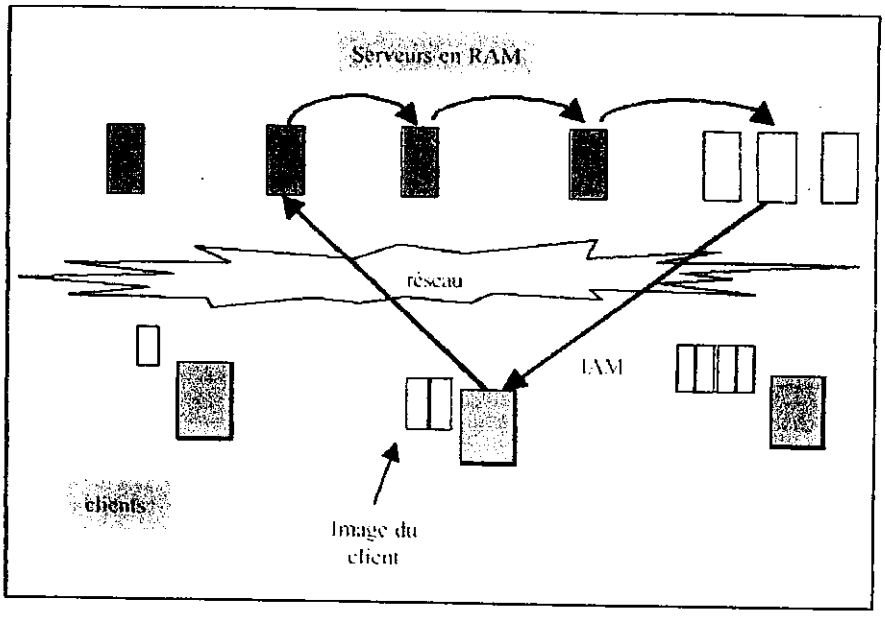


Figure I.2 : Schéma d'une structure de donnée distribuée [BB04]

4.1. Règles de base des SDDS [SDDS]

- Les données sont sur des sites serveurs.
- Il n'y a pas de répertoire central d'accès.
- Les mises à jour de la structure d'une SDDS ne sont pas envoyées aux clients d'une manière synchrone.
- Un client peut faire des erreurs d'adressage par suite d'une image inadéquate de la structure de SDDS.
- Chaque serveur vérifie l'adresse de la requête et l'achemine vers un autre serveur si une erreur est détectée. l'ensemble des renvois à la suite d'une erreur d'adressage ne se fait qu'en quelques messages.
- Le serveur adéquat envoie alors un message correctif (IAM–Image Ajustement message) au client ayant commis l'erreur d'adressage. le client ajuste son image pour ne plus faire la même erreur.

4.2. Caractéristiques des SDDS

Les SDDS possèdent plusieurs caractéristiques qui découlent essentiellement des systèmes distribués [LNS96]. Parmi ces caractéristiques, on peut citer : la scalabilité, la distribution et la disponibilité.

4.2.1 Scalabilité

La scalabilité (scalability) [DG92] est le fait de maintenir les performances d'un système quand le volume de données stockées varie. Une structure de donnée scalable est caractérisée par le fait que :

- Le temps d'insertion ou de recherche d'une donnée est indépendant du nombre d'éléments stockés, il est plus ou moins constant;
- Elle peut prendre en charge n'importe quelle quantité de donnée donc il n'y a aucune limite théorique a partir de laquelle les performances se dégradent ;
- En outre, il est souhaitable que sa capacité change de manière élégante et flexible (gracefully) sans avoir recours à une réorganisation totale ;

4.2.2 Distribution

Il arrive dans certaines situations que la quantité de données manipulée dépasse de loin la capacité de stockage et de traitement d'une seule station de travail : même si la capacité de stockage d'un ordinateur est largement suffisante, sa puissance de calcul est généralement insuffisante pour manipuler (recherche, traitement,...) une telle quantité de données. Une solution consiste à employer une structure de stockage qui distribue les données entre plusieurs nœuds (sites) du réseau. Ce qui offre au fichier une capacité de stockage théoriquement illimitée et permet d'éviter les points d'accumulation qui risquent de saturer le réseau d'interconnexion.

Une autre solution consiste à utiliser des schémas de partitionnement par hachage, qui partitionnent l'ensemble de données en fragments, chaque site stocke un ou plusieurs fragments. D'autre schémas utilisent un répertoire d'accès centralisé, visité avant chaque insertion ou recherche d'un enregistrement [Kar97]. Ce répertoire devient rapidement un point d'accumulation quand il reçoit les requêtes des différents sites pour les traiter et les redistribuer.

4.2.3 Disponibilité

Par (haute) disponibilité « high availability » on sous-entend que si un noeud du réseau tombe en panne, le fonctionnement global du système ne doit pas être affecté. les services doivent, donc, être assurés 24*7 [LMRS00, LS99].

La haute disponibilité est nécessaire dans les systèmes ou l'information est très pertinente et le temps de lecture d'une copie de sauvegarde (backup) n'est pas supporté : télécommunication, banques, transactions en ligne, ect.[Kar97].

Plusieurs structures à haute disponibilité ont été proposées. Parmi les quelles on trouve : RAID et ses variantes [Kar97, LMRS00, LS99], les schémas LH* à haute disponibilité.

Un autre concept lié à la haut-disponibilité est celui de la fiabilité (reliabilité), c'est la probabilité que toutes les données soient disponibles. les travaux de recherche mené dans le laboratoire CERIA sous la direction du professeur **W. litwin** ont permis d'étendre la définition de la disponibilité en proposant des schémas a :

- **k-disponibilité :** tous les articles sont accessibles même si n ($n \leq k$) sites sont en panne en même temps: LH^*_M [LN96], LH^*_S [LN95] et LH^*_G [LMRS00, LR97].

- **haute-disponibilité incrémentale (scalable high availability) :** puisque le nombre de cases d'un fichier augmente avec le temps, la faibilité du système diminue. cette approche permet d'éviter la diminution de la fiabilité quand la taille du fichier croit : LH^*_{SA} [LMR98], LH^*_{RS} [LS99].

Ces caractéristiques doivent assurer aux SDDS des performances inconnues des structures de données classiques.

4.3. Classification des SDDS

Ces dernières années une nouvelle classe de structures de données est apparue spécialement conçue pour un environnement distribué (multi computers). Ces structures de données portent le nom SDDS : Scalable Distributed Data Structure.

Les SDDS est un gestionnaire prototype scalable et distribue conçue pour des multiordinateurs wintel une SDDS permet une répartition scalable de donnée tout particulièrement en RAM distribue.

Dans un premier temps, des SDDS basées sur le hachage ont été proposées tels que LH* et ses variantes, DDH et EH*, celles-ci constituent une généralisation des algorithmes de hachage classiques. Cependant, dans le cas où le fichier doit supporter un parcours transversal de tous les enregistrements, les structures de données ordonnées telles que les arbres B offrent de bonnes performances par rapport aux techniques de hachage qui ne préservent pas l'ordre. Sur cette base, une autre famille de SDDS a été proposée pour construire des fichiers distribués qui préservent l'ordre des enregistrements. Citons à cet égard : RP* et DRT. En plus de ses deux catégories d'autre SDDS ont été proposée.

Selon l'organisation des données et le mécanisme qui permet d'y accéder les SDDS peuvent être classée en quatre catégories :

- **SDDS basées sur le hachage :** elles constituent une extension des schémas de hachage classiques sur les multiordinateurs : LH* (Distributed Linear Hashing) [KLR96, LNS93, LNS96], DDH (Distributed Dynamic Hashing) [Dev93], EH*

(Distributed Extensible hashing). IH* (Distributed Interpolation-based hashing) [BZ02].

- **SDDS pour les fichiers ordonnés :** elles étendent les structures ordonnées traditionnelles B-arbres, Arbres de recherche binaires) aux environnements distribués : RP* [LNS94], DRT [KW94], DRT* et arbre B+ distribué [PN00.b].

- **SDDS pour les fichiers multi-attributs :** elles constituent, essentiellement, une adaptation distribuée des k–d-arbres : k-RP* [LNS96.b] et k-DRT [PN00a&b].

- **SDDS a haute-disponibilité :** elles sont conçue pour assurer aux systèmes SDDS la continuité de fonctionner de manière transparente, quand un ou plusieurs de ses sites serveurs tombent en panne. ces SDDS font généralement appel a certains principes de redondance et de récupération de donnée tel que : Reed-solomon Code, le mirroring le striping et le grouping [LMR98, LMRS00, LR97, LS99].

5. Travaux sur SDDS

Dans un système de fichiers distribue (DFS pour Distributed File System), un fichier doit être stocké sur plusieurs sites. Cette distribution doit être transparente aux utilisateurs. Plusieurs approches ont été proposées pour atteindre cet objectif. Dans la plupart des DFS, chaque fichier réside entièrement sur un site, ce qui engendre des limitations en terme de capacité de stockage et de performances d'accès.

Une évolution naturelle est de distribuer le fichier sur plusieurs sites. Cette idée a abouti au début à une classe de DFS avec des schémas de partitionnement statiques. Ces schémas utilisent certains critères pour distribuer le fichier entre les serveurs. Une fois la distribution est établie, les critères de distribution ainsi que le nombre de sites restent statiques durant toute la durée de vie du fichier, bien que les mises a jour dans le fichier rendent ces paramètres moins optimaux. Pour les remplacer, il faut redistribuer le fichier et enlever l'ancienne copie. Parmi ces schémas, on peut citer :

* round-robin ou les articles du fichier sont distribués par rotation sur les sites quand ils sont insérés;

* partitionnement par hachage (hashed-declustring) dans lequel les articles sont affectés aux nœuds en utilisant une fonction de hachage;

* partitionnement par intervalle (range-partitioning) dans lequel l'ensemble des clés est divisé en intervalles, chaque intervalle est ensuite, affecté à un nœud;

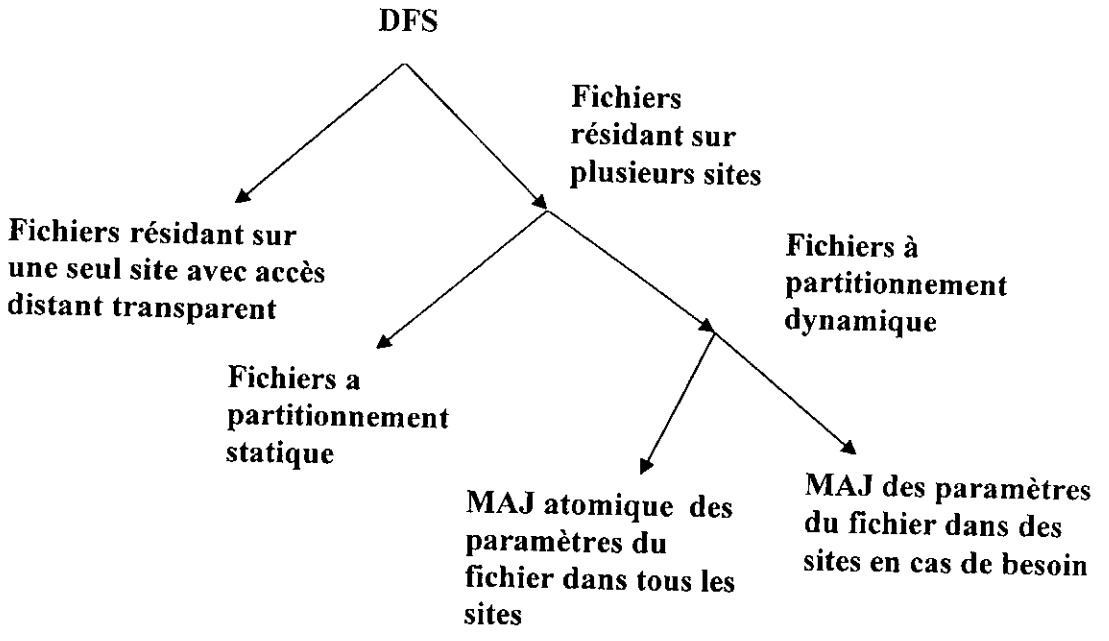


Figure I .3 : Typologie des système de fichier distribués [Ben00]

Ces schémas de données sont tous statiques, c-a-d le critère de distribution ne changera pas avec le temps le prix a payer est que le fichier ne peut dépasser la capacité (des sites) initialement allouée [Kar97, LNS96].

Pour dépasser ces limitations, des schémas de partitionnement dynamique viennent d'apparaître. Le premier de ces schémas été le hachage linéaire distribué (**DLH** pour distributed linéaire hashing) ce schéma a été conçu pour des machines multiprocesseurs fortement couplés avec une mémoire partagée (tightly coupled multiprocessors with shared memory).

Dans le DLH, le fichier est stocké en mémoire vive globale du système multiprocesseurs, ses paramètres sont rangés dans la mémoire cache locale de chaque processeur .les caches sont rafraîchies sélectivement quand une erreur d'adressage est commise ou par une mise a jour atomique de toutes les caches a un certain point durant l'évolution du fichier. On a montré, dans [LNS96, Kar97], que **DTH** donne de bons résultats par rapport au LH lorsque le taux des insertions dans le fichier est très élevé.

Précisions dans ce sens qu'un schéma utilisant des mises à jour atomiques peut être efficace dans des environnements fortement couplés. Cependant, il empêche un fichier distribué de s'étaler sur un grand nombre de sites.

5.1. Les SDDS basées sur le hachage

Les structures de données distribués et scalables (SDDS) se proposent de distribuer les données sur plusieurs nœuds de stockage. Au début des recherches, des SDDS basées sur le hachage ont été proposées, le hachage repose sur la division d'un fichier en n paquets de taille fixe, citons :

LH* (scalable distributed LH) Proposer en 1993 par **w.litwin** [KLR97, LNS93, LNS96], cette SDDS est basée sur le hachage linéaire, LH* constitue une révolution dans le domaine de structure de donnée est une démonstration de la faisabilité de l'approche SDDS.

DDH (distributed dynamic hashing) de **r.dévine** [Dev93]. Cette SDDS est basées sur le hachage dynamique. Le hachage dynamique (proposer par **P.A .Larson**) est une adaptation du hachage classique à la dynamique des fichiers,

Les versions de base du LH* et du DDH ont presque le même facteur de chargement. Le principal avantage du DDH est qu'aucune case ne stocke des enregistrements en débordement et les éclatements sont réalisés de manière autonome. Donc, on n'a pas besoin d'un site coordinateur comme dans le LH*.

Dans des travaux de recherche plus récents, **Vingralek** et **al**. [BVW95] ont étendu le LH* et le DDH pour mieux contrôler la charge du fichier. Leur but était de
Minimiser le nombre de serveurs en gardant toute fois les mêmes performances d'accès ceci en utilisant une stratégie d'équilibrage de charge (load balancing) plus flexible Le contrôleur du fichier (file advisor) gère le taux de chargement des serveurs en permettant a un site d'héberger plusieurs cases et de faire migrer, en cas de surcharge, quelques unes d'entre elles vers un autre site, par opposition au LH* et DDH ou un site gère une case seulement.

EH* (hachage extensible) Proposer par **V.Hilford** et **al**. Dans, cette SDDS est basées sur le hachage extensible (EH) de **P.Fagin.** Le schéma **EH*** consiste a distribué les donnée entre plusieurs serveur, des client autonome accède simultanément au case du fichier EH* permet une bonne utilisation de l'espace de stockage est offre un mécanisme de traitement de requête très efficace.

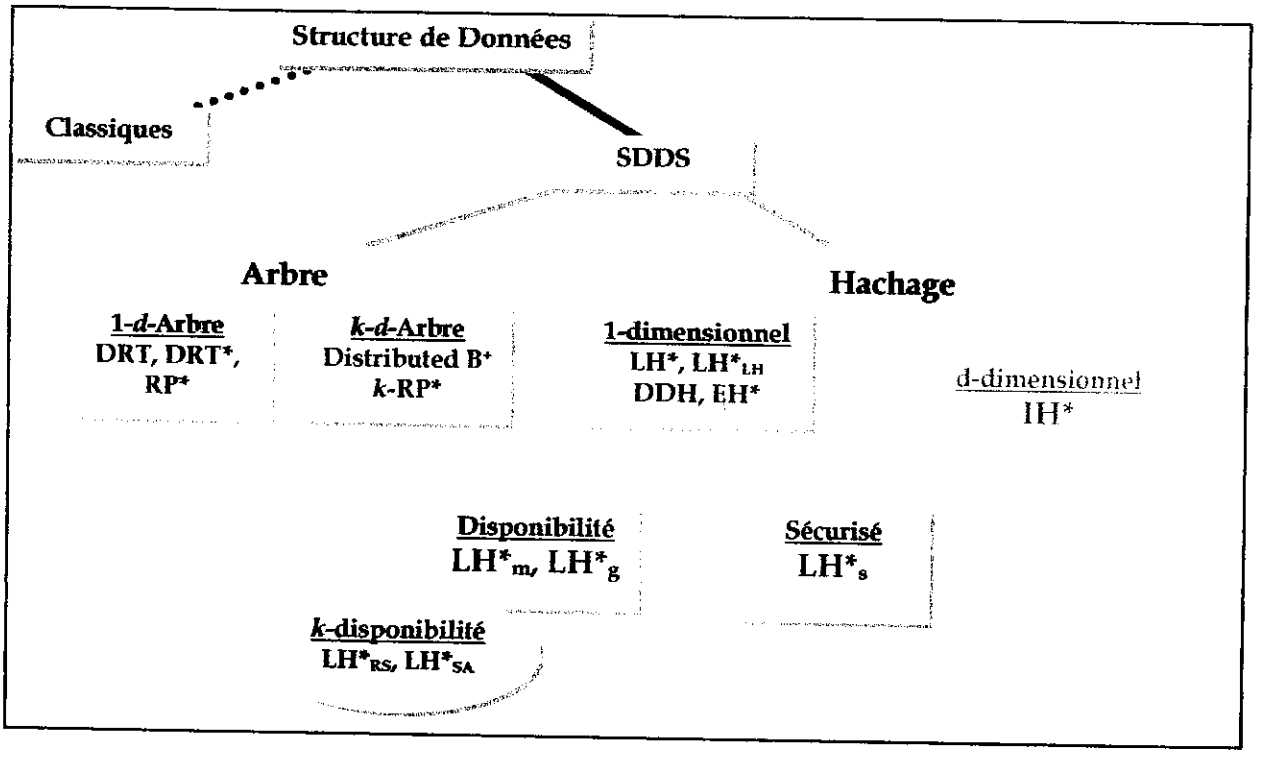


Figure I.4 : Aperçu des différentes variantes des SDDS [BB04]

5.2 SDDS Ordonnées

Les SDDS ordonnées sont apparues pour palier aux inconvénients de celles basées sur le hachage qui ne préserve pas l'ordre des enregistrements, les structures de données ordonnées tel que arbre-B sont plus rapide si le fichier doit supporter des parcours transversaux des enregistrements ou encore la recherche de l'enregistrement le plus proche selon une norme bien définie.

Par mis les SDDS ordonnée on propose :

RP* (range partitionning) : proposé par **W.Litwin** et **al.** [LNS94], les SDDS RP* constituent une généralisation des algorithmes de hachage. Leur but est d'offrir un fichier distribué qui préserve l'ordre des enregistrements. RP* regroupe trois schémas : RP*N, RP*C et RP*S [LNS94].

DRT (distributed random search tree): propose par **B .CROLL** et **al.** [KW94] pour l'accès au donnée a partir des clients et des serveurs, DRT ce repose sur une version distribue des arbres binaires de recherche (binary search tree).

DRT et RP* permettent de construire des fichiers plus larges et plus rapides par rapport aux structures de données traditionnelles [Kar97, LNS96].

Une très récente approche de recherche sur les SDDS vise a obtenir de bonnes Performances dans les cas les plus défavorable (good worst-case Performance) .RBST

(relaxed balanced search tree) et BDST (balanced distributed search tree) [PN00] constituent les premières SDDS résultant de ces travaux.

DRT* est une généralisation de la méthode DRT, et ce par l'adaptation d'une stratégie d'équilibrage de charge qui lui permet de garder des performances acceptables dans les cas défavorables. Le but de DRT* est de minimiser le coût de communication est ceci en adoptant une technique paresseuse de mise à jour des images serveurs. Dans ce sens les ajustements des arbres locaux ne sont effectués qu'en cas d'erreur d'adressage.

5.3 SDDS multidimensionnelles

Dans le domaine du multidimensionnel, plusieurs SDDS ont été développé. Citons :

K-RP*s : cette SDDS est une extension de la SDDS **RP*** au fichier multi-attributs cette SDDS utilise essentiellement des messages point a point. Dans cette SDDS on trouve des serveurs pour le stockage de donnée (serveurs buket) et des serveurs de gestion des erreurs d'adressage (serveurs index) [LNS96.b].

k-DRT [PN00.b] : cette SDDS est une extension multidimensionnelle de la SDDS DRT. Cependant il faut noté de cette famille de SDDS constitue une extension des k-d structure (k-d-trees) aux environnements distribués.

On présente dans ce mémoire une SDDS multidimensionnelle basée sur le hachage appelée **IH*.** Un fichier IH* peut débuter sur une seule machine (serveur) et s'étendre dynamiquement par insertion sur un nombre quelconque de machines. Ce fichier est manipulé par plusieurs clients répartis sur le multiordinateur.

6. Architecture Client-Serveur

L'architecture Client/Serveur est un concept logiciel qui ne date pas d'aujourd'hui : dès l'apparition des réseaux, les développeurs ont pensé à répartir l'exécution d'une application entre plusieurs (au moins 2) machines connectées par le réseau.

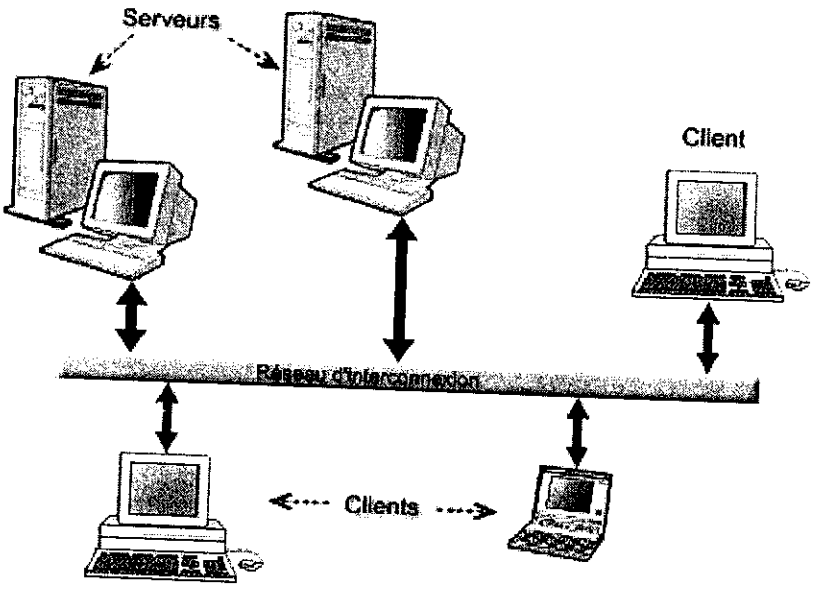


Figure I.5: Organisation d'applications Client/Serveur

L'idée de l'architecture client serveur est simple :

- ✓ le serveur est une machine qui met à disposition des services accessibles par d'autres machines sur le réseau (les clients). En général le serveur est à l'écoute des demandes des clients.
- ✓ le client, est une machine dont le principal rôle est parfois un traitement local des données, mais surtout la présentation graphique des données. En général c'est le client qui a l'initiative de la communication avec le serveur.
- ✓ le réseau transporte les données avec le protocole IP pour l'adressage et avec l'un des protocoles TCP ou UDP pour le transport. UDP est un protocole de transport plus simple que TCP car il ne fournit aucune garantie sur la fiabilité de la transmission. L'application doit prendre les mesures appropriées pour détecter et corriger les éventuelles erreurs de transmission.

6.1 Architecture interne du serveur

Les programmes client/serveur sont construits au niveau de la couche application pour permettre leur exécution sur toute plate forme supportant TCP/IP. Il existe, toutefois, trois schémas possibles pour l'architecture d'un serveur :

6.1.1 Serveur itératif

En mode connecté (TCP), le programme serveur suit les étapes de l'algorithme suivant :

Algorithme ServeurItératif
Début
1- Création du socket Local.
2- Affectation du socket à un numéro de port spécifique.
3- Définition de la taille de la file d'attente.
4- Attente d'une connexion.
5- Lecture et traitement de la requête.
6- Envoi de la réponse.
7- Fermeture de la connexion (cas TCP).
8- Aller à 4.
Fin

Dans le cas des messages UDP, le programme du serveur ne comporte pas l'étape de connexion et le réglage de la taille de la file d'attente.

6.1.2 Serveur parallèle

L'algorithme précédent est légèrement modifié afin d'obtenir un parallélisme dans le traitement des requêtes :

Algorithme ServeurParallèle
Début
1- Création du socket Local.
2- Affectation du socket à un numéro de port spécifique.
3- Attente d'une connexion.

4- Lancement d'un processus ou thread asynchrone pour la lecture et traitement de la requête. Le thread se charge de transmettre la réponse et de fermer la connexion (cas d'un message TCP).

5- Aller à 3.

Fin

6.1.3 Serveur semi-parallèle (preforked)

Cette technique consiste à créer plusieurs threads (processus légers) au démarrage du programme serveur. Les requêtes des clients seront ensuite distribuées sur les files d'attentes associées à chaque thread. Ces threads sont souvent appelés threads de travail (Worker Threads).

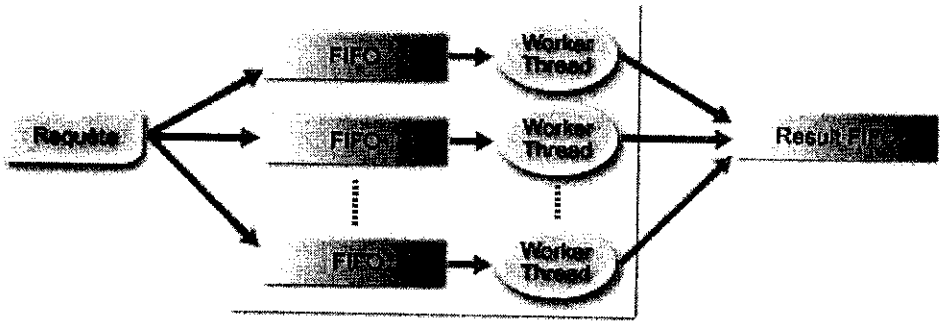


Figure I.6 : Schéma général d'un serveur semi-parallèle

7. conclusion

Dans ce chapitre, on a présenté la nécessité d'une structure de donnée qui exploite les caractéristiques des multiordinateurs et ça vue du volume de données échangées entre les applications. Nous avons présenté les SDDS qui sont venues pour répondre à ce besoin, on a donné leur définition, leurs caractéristiques et leur classification.

Nous avons présenté aussi les différentes technologies d'un serveur SDDS.

Dans le chapitre suivant, on présente le hachage par interpolation, sa fonction d'accès, ainsi que les différentes opérations d'insertion, de suppression, et de mise à jour. Puis on présente les éléments de bases du IH*[Bur83], ainsi que ses mécanismes d'éclatement et d'adressage.

Chapitre II

HACHAGE PAR INTERPOLATION DISTRIBUE ET SCALABLE

Hachage par Interpolation Distribué et Scalable IH*

1. Introduction

Si les structure de donnée unidimensionnelles demeurent toujours importantes, les nouveaux systèmes de gestion de données (SIG, imagerie,...) ont besoin des nouvelles méthodes d'accès qui supportent efficacement les opérations spatiales sur les fichiers multi-attributs volumineux.

Une nouvelle classe de structures de données, appelée SDDS est apparue cette dernière décennie pour offrir une solution efficace à cette situation et ouvrir de nouvelles perspectives pour la gestion de fichiers dynamiques volumineux.

Les SDDS constituent une famille de structure de donnée définie spécifiquement pour les multiordinateurs. elles stockent les données sur des sites désignés serveurs et d'autres sites appelés clients y accèdent. les sites clients gardent des paramètres pour calcul de l'adresse des fichiers sur les sites serveurs ces paramètres constituent l'image du client sur le fichier SDDS [SDDS].

Les données d'une SDDS résident en mémoire distribuée du multiordinateur. En effet, le temps d'accès des données est plus court que celui aux données stockées sur disque. Cette nouvelle structure dispose aussi du traitement parallèle, ce qui fait l'augmentation de la taille des données ne détériore pas les performances d'accès. ces

caractéristiques doivent assurer aux SDDS des performances de traitement inconnues des structures de données classiques[SDDS].

De nos jours plusieurs SDDS sont développées : **LH*** (une structure de donnée distribuée et scalable a base du hachage linéaire de **W.litwin** [KLR 96, LNS93, LNS96] **RP*** (une version distribuée et scalable des arbres B+ [LNS94], k-RP* (une Version multidimensionnelle du RP*) [LNS96.b], DRT [KW94], k-DRT [PN00.a&b], etc.

La SDDS présentée dans ce chapitre, appelée **IH***(scalable distributed interpolation-based hashing), est la première SDDS multidimensionnelle basée sur le hachage.

IH* constitue, a la fois, une adaptation du hachage par interpolation de **W. A. Burkhard** [Bur83] aux environnements distribuées et une introduction de la notion d'ordre et de l'aspect multidimensionnelle a la SDDS **LH*** de **W.Litwin** [LNS93, LNS96].

En ce qui suit on présente le hachage par interpolation avec sa fonction d'accès et ses déférentes opérations (insertion, suppression, recherche), puis on cite les éléments de bases de la SDDS IH*, ainsi les mécanisme d'éclatement et d'adressage.

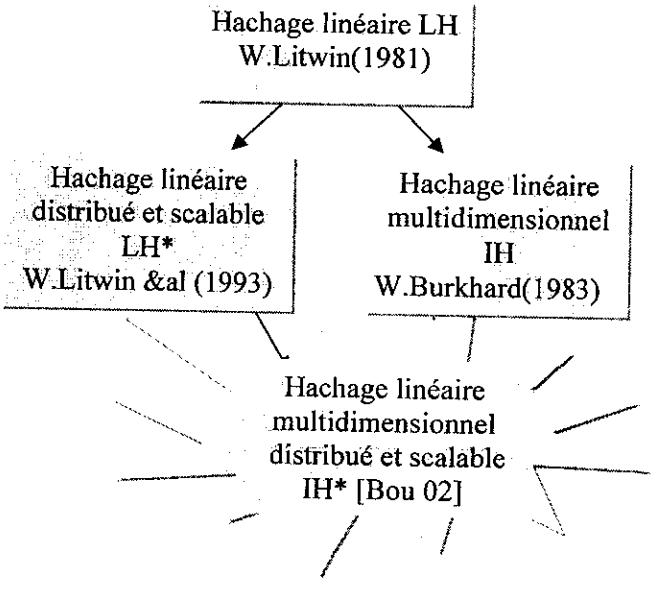


Figure II. 1: la nouvelle SDDS IH*

2. Hachage par interpolation

Le hachage par interpolation (HI pour interpolation-based hashing) de **W.A.Burkhard [BUR 83]** est une adaptation du hachage linéaire de **W.Litwin** qui préserve l'ordre des clés. La technique est conçue pour gérer des grands fichiers dynamiques et multi clés, elle n'utilise pratiquement pas d'index. Le temps d'accès est au voisinage de 1. le chargement du fichier est contrôlable par l'utilisateur. Dans ce qui suit, nous rappelons quelques éléments de base qui nous permettent de présenter la méthode. nous donnerons également la fonction de division utilisée.

2.1 Définition et Notation

Définition de la Fonction de projection (fonction shuffle) : soit k point de K

$$S(k) = \Sigma_{i\geq 1} \Sigma_{1\leq j\leq d} k_j^i \; 2^{-d.i+j+d} = \Sigma_{i\geq 1} \Sigma_{1\leq j\leq d} k_j^i \; 2^{(1-i)d+j}$$

Pour toute clé **k** de **K**

$$S(k) = \Sigma_{1\leq j\leq d} S(P_j(k))$$

Remarque :

$\mathbf{P_j(k)}$ est la clé pour laquelle la j-eme composant est égale a $\mathbf{k_j}$ de **K** et dont les (d-1) autres composants sont nulles:

$$P_j(k) = (0,\ldots\ldots,0.\ k_j,\ 0,\ldots..,0)$$

Définition de la projection π de]0,1[dans N : pour tout entier naturel i

$$\pi_i : [0,1[\rightarrow \{0, 1,\ldots.2^{i-1}\} \qquad (1)$$

$$\pi_i(x) = [2^i.x]$$

Il est facile de voir que :

$$\begin{cases} \pi_{i+1}(x) = 2^i . \pi_i(x) \\ \pi_{i+1}(x) = 2^i . \pi_i(x) + 1 \end{cases}$$

2.1.1 Fonction de hachage

Le fichier est un ensemble d'informations de même nature. l'article (ou enregistrement) constitue l'unité du fichier, les articles sont généralement groupes en Cases (bucket en anglais) sur une mémoire secondaire usuellement le disque. chaque case peut contenir un même nombre d'articles noté **b**, appelé capacité, l'article est composé de plusieurs attributs. l'un d'entre eux appelée clé permet l'identifier.

Soient $\mathbf{h_0}, \mathbf{h_1}$ ..... des fonctions de division

$$\mathbf{h_i} \ : \quad \mathbf{K} \rightarrow \{0, 1, 2 \ldots . 2^i\text{-}1\}$$

La fonction $\mathbf{h_i}$ définit $\mathbf{2^i}$ classes d'équivalences ou blocs. chaque bloc R_j^i, $(0 \leq j < 2^i)$ est définit comme suit :

$$R_j^i = \{k \mid (\pi_i S(k)) = j\}$$

Bien entendu, l'ensemble des région définit une partition dans **K**, c'est-à-dire :

$$\begin{cases} R_j^i \cap R_{j'}^i = \phi & j \neq j' \\ \cup_{0 \leq j < 2^i} (R_j^i) = \mathbf{K} \end{cases}$$

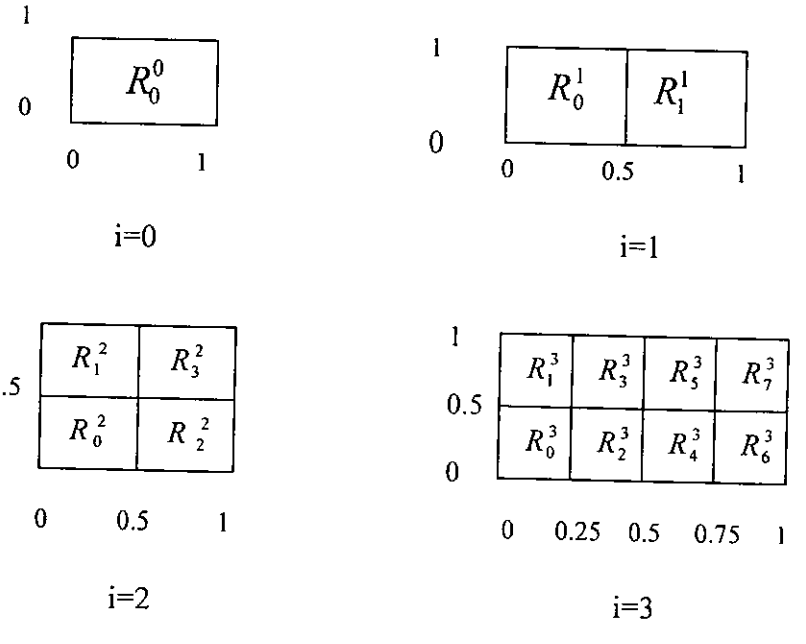


Figure II .2 : les régions d'un fichier IH pour (d=2)

A chaque fonction $\mathbf{h_i}$ est associé un ensemble de régions R_j^i, j variant de 0 a 2^i-1. le passage de $\mathbf{h_i}$ a $\mathbf{h_{i+1}}$ divise chaque région R_j^i en deux régions distinctes R_{2j}^{i+1} et R_{2j+1}^{i+1} on a la relation:

$$R_j^i = R_{2j}^{i+1} \cup R_{2j+1}^{i+1}$$

De ces propriétés découlent la fonction de divisions utilisées dans le hachage par interpolation :

$$\begin{cases} \mathbf{h_{i+1}} = \mathbf{h_i(k)} & \mathbf{k} \in R_{2j}^{i+1} \\ \mathbf{h_{i+1}} = \mathbf{h_i(k)} + \mathbf{2^i} & \mathbf{k} \in R_{2j+1}^{i+1} \end{cases} \qquad (2)$$

Ce qui veut dire que lors d'un éclatement de case, les plus petites clés dans R_j^i selon l'ordre $\leq_s$ **k** restent dans la case $\mathbf{h_i(k)}$ et les autres sont transférées dans la nouvelle case $\mathbf{h_i(k)} + \mathbf{2^i}$.

2.1.2 Constructions de la Fonction d'accès (fonction de division)

En fait, il ne s'agit pas de construire une fonctions d'accès mais plutôt, une séquence de fonctions.

La suite des fonctions **H** est donc obtenue par la suite des fonctions composées (**boπoS**) d'ou :

$$\mathbf{h_i} = (\mathbf{b} \circ \pi_i \circ \mathbf{S})$$

2.2 Opérations

En ce qui concerne le IH, nous savons maintenant comment attribuer une adresse a une classe de clés (dite région), et surtout comment augmenter le nombre des adresses en éclatant les régions.

Il nous faut maintenant voir comment appliquer les différentes opérations d'insertion, de suppression et mise a jour sur la base de ce schéma de structure de données.

Notons que les opérations d'insertion, de suppression et de mise a jour concernent généralement, un seul enregistrement, a part l'aspect multidimensionnel, ses opération sont presque identique a leurs homologues dans le schéma LH.

Dans les paragraphes qui suivent, ce présente les mécanismes d'exécution des différentes opérations classiques, les actions associées (éclatement et regroupement).

2.2.1 Eclatement d'une case

Les insertions se font normalement jusqu'au moment ou une case primaire est saturée, c'est la première collision. Il est alors nécessaire d'étendre l'espace des adresses. Pour ce faire, il faut augmenter le nombre de régions. c'est la scinder en deux régions disjointes.

```
Algorithme Split
Début
                n □n+2^i
                pour toute clé dans la case n faire
                    Si   h_{i+1}(h) =n   alors
                      déplacer k vers la case n
                  Fsi
                FPour
                    n □ n+1
                Si  n=2^i  alors
                  n□0
                  i□i+1
                Fsi
Fin
```

2.2.2 Recherche simple

Lorsque la partition de k est telle que toutes les région sont du même ordre (i), la fonction d'accès utilisée est h_i l'adresse de la case associée a la région contenant la clé k est égale a h_i (k).

Il est clair que l'éclatement incrémentiel implique la coexistence de régions d'ordre i et de région d'ordre (i+1). selon le principe du IH, si k appartient a la région R_j^i et que

celle-ci a été éclatée, alors il faut déterminer si k appartient a R_{2j}^{i+1} ou a R_{2j+1}^{i+1}. en d'autres termes, l'adresse (a) de la case contenant k est :

- h_i (k) si h_i correspond a une case non encore éclatée (h_i (k) ≥ n);
- h_i (k) dans le cas contraire (h_i <n);

```
Algorithme access(clé k)
Début
        a ← h_i (k)
        Si a <n alors
                a←h_i(k)
         fsi
        Retourner a
Fin
```

2.2.3 Insertion d'une clé

L'insertion d'un enregistrement identifie par le vecteur clé k dans le fichier IH Consiste à déterminer l'adresse de la case ou doit être inséré cet enregistrement et Résoudre les éventuels problèmes de collision.

L'adresse (a) de la case correspondant a la clé k est déterminée en utilisant L'algorithme (access). Pendant l'opération d'insertion deux cas de figure se présenter :

- ☒ la case (a) n'est pas saturée (pas de collision). dans ce cas, la clé y est insérée;
- ☒ la case (a) est saturé (il y a collision): dans ce cas l'étape de résolution de collision vient résoudre ce problème;

Comme dans le LH, l'éclatement du fichier peut suivre l'une des stratégies :

Sans contrôle d'éclatement (uncontrolled Split) :

Chaque fois qu'une collision aura lieu fichier subira une extension d'une case.

Avec un contrôle d'éclatement (controlled Split):

dans ce cas la collision ne suffi pas seule pour une déclencher un éclatement, il faut aussi que le taux de chargement du fichier dépasse un certain seuil maximum fixé

Auparavant par l'utilisateur, et au-delà duquel les performances d'accès plus Acceptables.

D'ou l'algorithme d'insertion suivant :

```
Algorithme insert(clé k)
Début
        a ← access (k)
        inserer(a, k)
        Si collision dans a alors
            Si éclatement non contrôlé alors
                    Split()
            Sinon
                Si taux de chargement ≤ seuil maximum alors
                 fsi
            fsi
        fsi
Fin
```

2.2.4 Suppression

L'éclatement permet une extension de l'espace des adresses lorsque c'est nécessaire. le groupage, opération inverse, va permettre quant a elle de rétrécir cet espace.

Il est évident que le groupage de deux région R_j^i et $R_{j'}^i$ n'est possible que si (j'=j+1), la région obtenue sera $R_{[j/2]}^{i-1}$. en termes de case, on dit que le groupage de deux cases n'est possible que s'il y a eu au moins un éclatement (i>0), et il concerne les cases qui résultent du éclatement.

Selon le principe du LH, les cases concernées par le groupage sont les cases d'adresse (n-1) et (n-1+2i) si n est supérieur a 0, c-a-d la partition de k comprend des régions d'ordre i des régions d'ordre (i+1). si n est nul, alors la partition de k est composée uniquement de région d'ordre (i>0). le groupage concernera donc les cases (2^i) et (2i -1). dans ce cas le groupage fait ressortir dans la partition de k une région d'ordre (i-1), d'ou la nécessité de mettre a jour la valeur de i, en plus de la valeur de n qui est remise a jour dans tous les cas. il évident que le groupage doit restaurer la partition de k dans l'état précédant le dernière éclatement.

L'opération de groupage dans le IH est réalisée par l'algorithme (shrink):

```
Algorithme Shrink
Début
        Si  i>0 alors
            n←n-1
            Si  n<1  alors
                i←i-1
                n□2^i
             fsi
        fsi
        n'← n+2^i
        pour  toute clé k dans la case n'  faire
                déplacer k vers case n
         fpour
fin
```

Contrairement a l'éclatement, l'opération de groupe n'est déclenchée que lorsque le taux de chargement du fichier de devient inférieur a un certain seuil minimum fixé auparavant par l'utilisateur. L'algorithme (delete) résume le principe de l'opération de suppression dans un fichier IH :

```
Algorithme Delete(clé)
Début
            a←access(k)
            Si  k existe dans a  alors
                Suprimer k de a
            fsi
            Si  taux de chargement du fichier < seuil minimum alors
                Shrink()
            fsi
fin
```

En ce qui suit on présente les éléments de bases du IH*, en l'occurrence l'organisation d'un fichier IH*, les mécanismes d'éclatement et d'adressage dans la SDDS IH*.

3. Hachage par Interpolation Distribue et Scalable

3.1 Structure du fichier IH*

Comme dans le IH classique, un fichier IH* est constitué d'un ensemble d'enregistrements identifies chacun par vecteur clé dimensionnel k= $(k_1....k_d)$ $(d\geq 1)$ et une partie non-clés organisées en n attributs $(n \geq 1)$. les enregistrements sont rangés dans des cases de capacité b, (b>>1). chaque case couvre une région R_j^i dans le Schéma de base **IH** ou (i) le niveau de la Case et (j) est son numéro. (j) varie de 0 a m où m est le nombre de cases actuellement Présentes dans le fichier **IH***. Les numéros des cases sont considérés comme des adresses logiques de celles-ci.

L'extension du **IH** a **IH*** consiste a mettre chaque case du fichier **IH** sur un Serveur différent du multiordinateur. Ces cases sont stockées en mémoire Centrale du multi ordinateur à raison d'une case par serveur. Elles peuvent bien Être sur disque. les informations de gestion de chaque case **IH*** (adresse, Niveau, handle, dimension,....) Sont stockées au niveau de son entête.

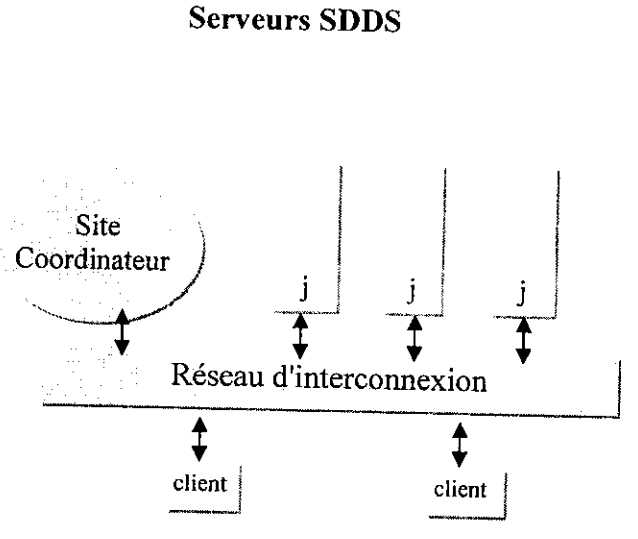


Figure II. 3 : Structure du fichier IH*

L'image exacte du fichier (n, i), (i étant le niveau du fichier et n est l'adresse du prochain serveur à éclater), est maintenue dans un site particulier appelé site coordinateur. Dans Ce qui suit, l'abréviation "CoS" dérivée du mot anglais (Coordinator Site) sera utilisée pour désigner le site coordinateur.

Le CoS gère entre autres les éclatements et les contractions du fichier IH*. il maintiennent également la table complété des adresse physiques des serveurs qui peuvent héberger des cases du fichier IH*.

Chaque client de la SDDS maintient une image locale (n', i') : i' étant la valeur présumée du niveau du fichier et n'est la position de n initialement (i' = n' = 0).

3.2 FONCTION DE HACHAGE

Dans le cas toutes cases du fichier auraient le même niveau (i), la fonction de Hachage (h_i) est utilisée pour associer a un enregistrement identifie par un vecteur clé=(k_1....k_d)

l'adresse (a) du serveur ou il doit être logé. la fonction (h_i) est définie comme suit [Bur83]:

Fonction h_i (k)
debut
Retourner ($b0\prod_i 0s$)(k)
fin.

Sous l'effet des éclatements, des cases de différents niveaux peuvent coexister dans le même fichier : (j' = i) pour les cases non encore éclatées et (j'=j+1) pour les cases éclatées ou nouvellement crées. dans ce cas, l'algorithme de hachage (access) est utilisé pour associer a l'enregistrement identifié par le vecteur clé k l'adresse (a) du serveur ou il doit se trouver (a=0,....,2+n-1).

Algorithme access (clé k)
debut
a ← (h_i)
Si a<n' **alors**
a ← h_{i+1}(k)
fsi
retourner a
fin.

Comme dans LH*, un client qui veut accéder a une cases du fichier exécute (access) en utilisant, bien sur, ses paramètres locaux (n', i'). cette technique peut induire, éventuellement , des erreurs d'adressage. cela n'est pas du a l'algorithme d'adressage mais allons détailler le mécanisme d'adressage du IH* en exposant les problèmes rencontrés et les solution proposées.

3.3 EXPANSION DU FICHIER

Mise a part l'aspect distribué, n fichier IH* se comporte comme un fichier IH. il consiste initialement en la case R_0^0. l'image exacte de fichier est dans ce cas (n=0,i=0). Quand la case R_0^0 déborde, elle est éclaté. la case R_1^1 est donc crée et les nouveaux paramètre du fichier deviennent (n=0, i=1) de la prochaine collision la case R_1^1 est de nouveau éclatée donnant naissance R_1^2 le pointeur d'éclatement pointe maintenant sur la case R_1^1 et (n=1, i=1) devient la nouvelle image du fichier etc.

Notons que selon le principe hachage linéaire, l'éclatement concerne toujours la case R_n^i son éclatement donnera naissance aux deux cases : R_{2j}^{i+1} et R_{2j+1}^{i+1} durant cette opération :

- En moyenne, la moitié des enregistrements de la case mère seront transférés vers la case fille;
- la portée de la case R_n^i va être scinder, après éclatement, en deux régions a portées égales R_{2j}^{i+1} et R_{2j+1}^{i+1};

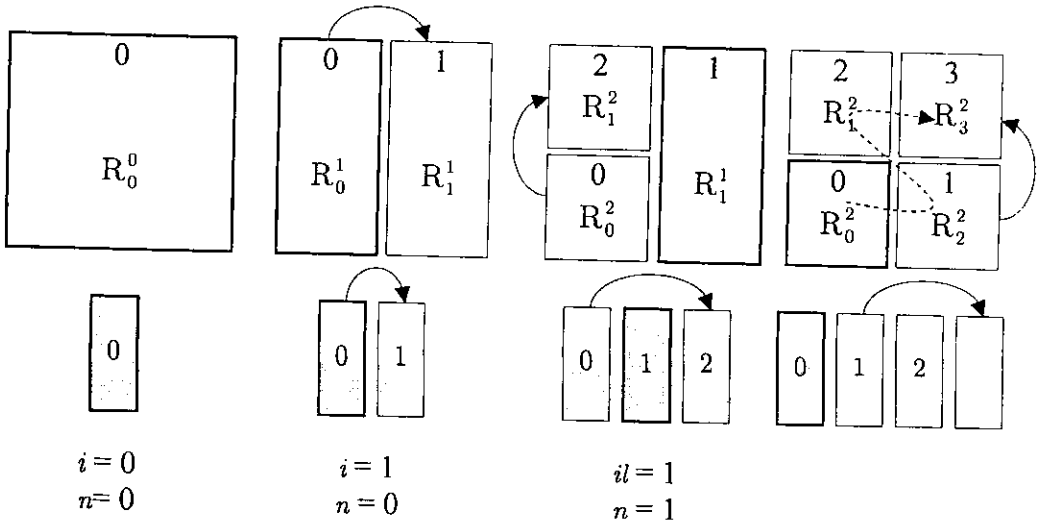


figure II.4 : Expansion du fichier IH*

3.4 ECLATEMENT

Selon le principe du hachage linéaire, le fichier IH* étend a travers le mouvement linéaire du pointeur d'éclatement (n) et ceci par éclatement des régions une par une et dans l'ordre (j=0,1,..... 2^i +n-1).

Remarque que les éclatements forment deux séries indépendantes :

- Une série de période fixe (T=d) par rapport a la dimension : éclatement périodique de la 1^{eme}, 2^{eme},..., d-eme dimension; puis 1^{eme} ,2^{eme},....
- Une série de périodes variable ($T=2^i$) par rapport aux numéros des cases : les éclatement suivent cet ordre : 0;0,1;0,1,2,3;0,1,...6,7;0,1.....2^i-1; .... etc .

L'éclatement des cases d'un fichier IH* peut être contrôle (il se produit lorsque le taux de chargement global du fichier atteint un certain seuil maximal τ fixé au paravant) ou non Contrôlé (il se produit a chaque collision).

3.4.1 Distribution de l'algorithme d'éclatement

Dans le schéma de base IH [Bur 83], l'algorithme (split) est utilisé pour effectuer des éclatements. Cet algorithme s'écrit comme suit :

```
Algorithme Split
Début
        n □n+2^i
        pour toute clé k dans la case n faire
                si    h_{i+1}(k) =n      alors
                      déplacer k vers la case n
                Fsi
        FPour

        n  □ n+1
        Si   n=2^i  alors
              n□0
              i□i+1
        Fsi
Fin
```

Du fait stockage distribué d'un fichier IH*, l'application de (split) dans un environnement distribue pose le problème du maintien de la cohérence de limage exacte au niveau du CoS et les images locales des clients.

A fin de résoudre ce problème, nous avons proposé une solution qui consiste à scinder l'algorithme (split) en deux algorithmes mutuellement indépendants :

- Le premier algorithme est executé par le serveur (n) puisque c'est lui qui héberge la case à éclater. ce serveur s'occupe de l'initialisation et du transfert des enregistrements vers la nouvelle case d'adresse (2^i +n). les étapes de cette opération sont résumées dans l'algorithme (SplitServer) :

Algorithme SpiltServer
Début
1- créer la case $n+2^j$ avec comme niveau (j+1);
2- éclate la case (n) en utilisant comme fonction de hachage h_{j+1} (les objets correspondant sont envoyés vers la nouvelle case);
3- mettre à jour $j \leftarrow j+1$;
4- confier l'éclatement au CoS;
Fin

- Le deuxième algorithme (CoSImage Adjustement) est utilisé par le CoS pour maintenir l'image exacte du fichier :

Algorithme SCImageAdjustement
Début
$n \leftarrow n + 1$
Si $n \geq 2^i$ **Alors**
$n \leftarrow 0$
$i \leftarrow i + 1$
Fin

Les détails du protocole d'éclatement seront donnés dans le chapitre suivant consacré entièrement a la conception d'un protocole de communication de la SDDS IH*.

3.4.2 Eclatements contrôlé et non contrôlé

L'algorithme d'éclatement utilisé dans le schéma de base IH prend en charge les deux modes d'éclatement, à savoir, contrôlé et non contrôlé.

a. Eclatement non contrôlé

Dans ce cas, il suffit que le CoS reçoive un message de débordement du serveur qui a subi une collision (pas nécessairement celui pointé par n). Le CoS envoie alors un ordre d'éclatement au serveur (n). en recevant cet ordre, ce dernier initialise le serveur d'adresse (+n) et lui envoie les enregistrements correspondants. des qu'il finit, il envoi message d'acquittement au CoS pour lui permettre de calculer la nouvelle image (n, i).

b. Eclatement contrôlé

Dans ce cas mode d'éclatement, le CoS a besoin de connaître le taux de chargement exact du fichier a fin de décider si un éclatement est nécessaire ou non. ce qui n'est pas évident a priori, puisque les opérations d'insertion et de suppression ne transitent pas par le CoS. Donc comment permettre au CoS connaître le taux de chargement du fichier?

La solution déterministe consiste a informer le CoS chaque fois qu'il y a insertion ou suppression dans le fichier. ce qui va a l'encontre du 1^{ere} principe des SDDS puisqu'elle augmente le nombre de message dans le réseau et rend le CoS un point d'accumulation.

A fin de palier a cet inconvénient, une solution probabiliste est proposée par **W.Litwin** [LNS96] pour la SDDS LH*.cette solution consiste a estimer le taux de chargement du fichier en se basant sur les informations venant du serveur ayant subi une collision :

Soit **S** ce serveur, **b** sa capacité, **x** le nombre d'objets qu'il contient et τ le seuil maximal de chargement, S envoi un message informant le CoS d'une situation de collision ce dernier calcule le taux de chargement du fichier en utilisant **b** et **x** s'il vérifie que le taux estimé est supérieur a τ, il déclenche l'éclatement sinon rien ne va se passer.

L'algorithme suivant (controledsplit) est celui utilisé dans LH* pour estimer le taux de chargement du fichier LH*.

```
Algorithme ControledSplit
Début
        d ← x/b
        Si a> n ou  a>2^i alors
                d ← d*2
        fsi
        t← (2^i*d)/(2^i+n)
        Si t >τ alors
                envoyer un ordre d'éclatement au serveur(n)
        Sinon
                /* pas d'éclatement pour le moment */
        fsi
Fin
```

Précisons que dans le premier prototype de la SDDS IH*, nous avons utilisé la formule du LH* pour estimer le taux du chargement du fichier. l'étude d'une nouvelle formule plus optimale peut faire l'objet de futurs travaux de recherche.

3.5 ADRESSAGE

Les cases d'un fichier **IH*** sont accédées par plusieurs clients simultanément. chaque client maintient une image locale (n',i') qui n'est pas nécessairement identique a celle du CoS (typiquement inconnus par les autres machines). initialement, (n'=0) et (i'=0). cette image n'est mise a jour que lorsque le client manipule le fichier.

La SDDS IH* adopte un mécanisme d'adressage qui repose sur trois principes: le calcul de l'adressage par le client, la vérification de la destination par le serveur et la mise a jour de l'image du client en cas d'erreur d'adressage.

3.5.1 Calcul de l'adresse par le client

Le client envoie sa requête (insertion, suppression,...) relative au vecteur clé $k=(k_1,k_2,....k_d)$ au serveur (a). l'adresse (a) est calculée en utilisant l'algorithme (access) appliqué a (n',i') au lieu de (n,i) :

```
Algorithme access (clé k)
Début
        a ← h_{i+1} (k)
        Si a <n' alors
                a←h_{i+1}(k)
         fsi
        retourner a
Fin
```

3.5.2 Les opérations dans IH*

Les opérations de recherche, d'insertion et de mise a jour sont identiques aux opérations IH qu'on a parlé déjà :

La recherche : l'adresse (a) de la case contenant k est :

- h_i (k) si h_i correspond a une case non encore éclatée (h_i (k) ≥ n);
- h_i (k) dans le cas contraire (h_i <n);

L'insertion : pour insérer une clé k il faut déterminer l'adresse doit être insérer cette dernière et résoudre les problème de collision.

La suppression : le groupage opération inverse de l'éclatement permet de rétrécir l'extension de l'espace des adresses du aux éclatements.

3.6 REQUETES A INTERVALLES PARALLELES

Une requête parallèle Q est une opération lancée par un client a plusieurs serveurs d'un fichier F .les exécutions de Q chaque serveur sont mutuellement indépendantes. Un serveur SDDS ne répond que s'il reçoit une copie de Q .par défaut, si Q est une recherche, la réponse contiendra les objets satisfaisant Q .notons que pour certaines techniques d'arrêt, la requête Q peut exiger une réponse de la part de chaque serveur, même si le résultat est vide .cela peut être pour le client pour s'assurer que tous les serveurs concernés par la requête Q ont bien répondu.

Une particularité de toutes les SDDS y compris IH*, est qu'un client ne reconnaît que l'ensemble des serveurs figurant dans son image locale. si un serveur est crée après la dernière mise a jour de cette image. ses descendants seront inconnus par le client.

Néanmoins, cela n'empêche pas qu'un copie de la requête a intervalles Q leur Parviendra.

4. CONCLUSION

Présenté dans ce chapitre le hachage par interpolation. Plus précisément sa fonction d'accès, ainsi que les différentes opération d'insertion, de suppression et de mise à jour sur la base de ce schéma de structures de données.

En plus, présenté une nouvelle structure de données distribuée et scalable conçue spécialement pour les multiordinateurs. cette SDDS appelée IH*, constitue une adaptation du hachage par interpolation classique aux environnements distribués, elle peut être également considérée comme une extension multidimensionnelle de la SDDS LH* avec préservation de l'ordre des enregistrements.

On a présenté également les élément de base de cette SDDS, en l'occurrence l'organisation d'un fichier IH, les mécanisme d'éclatement et d'adressage dans la SDDS IH*.

CHAPITRE III

Protocole de communication pour la SDDS IH* et l'architecture du système

Chapitre II

Protocole de Communication Pour la SDDS IH* et l'architecture du système

Le but de ce chapitre est de proposer un protocole de communication pour la SDDS IH* et de programmer un premier prototype.

Dans cette version, il été amené à simplifier certains aspects du protocole.

1. TRAITEMENT DES MESSAGES IH*

Les différents types de messages utilisés dans le protocole de communication IH* sont organisés suivant une hiérarchie.

Les deux grandes classes de messages sont les messages de données utilisés pour le transfert des données et les messages de service utilisés pour des préavis de certaines transactions et pour les commandes d'administration du système. Les différents types de messages dans la SDDS IH* sont représentés sur la figure suivante :

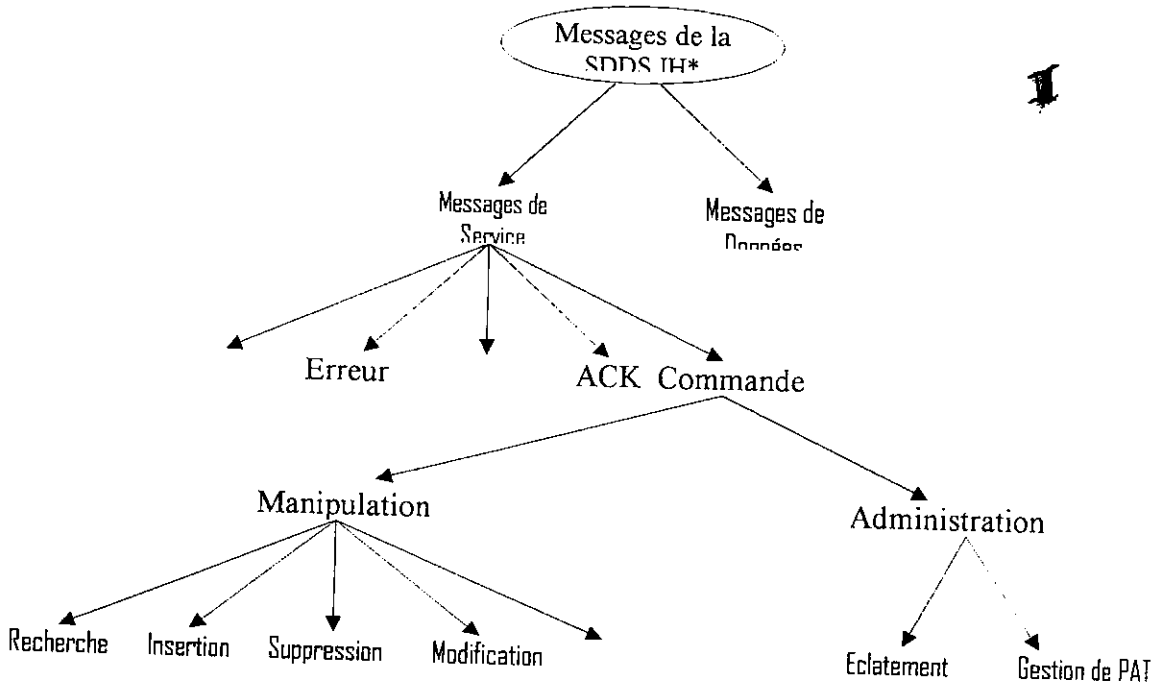


Figure III.1 : Types de messages dans la SDDS IH*[BB04]

1.1. Messages de données

Pour pouvoir transférer des données d'une manière plus efficace, il semble important de pouvoir choisir le protocole de transport de données le plus adéquat.

Le transfert de donnée se fait dans plusieurs sens :

- Client - serveur : pour les opérations d'insertion et de modification;
- Serveur - client : réponse d'une requête de recherche simple;
- Serveur - serveur : en cas d'éclatement d'une case ou en cas de redirection d'une requête mal adressée;

Dans notre prototype le protocole UDP qui ce charge du transférer les données

1.2 Messages de service

Les messages de service englobent les différentes commandes fonctionnelles ou administratives (préavis d'établissement de connexion, des accusés de réception,...), des informations sur l'émetteur, la commande à exécuter (insertion, recherche, suppression, éclatement, ...) et voire les arguments de cette commande.

Dans notre prototype, les messages de service sont acheminés en utilisant le protocole UDP du fait de la taille relativement petite de ce type de messages ainsi que la rapidité et la simplicité du protocole UDP.

1.2.1 Messages d'accusé de réception

Un message d'accuser de réception contenant l'adresse et le niveau du serveur ayant exécuté la commande, est retourné au client, le client se servira de ce massage pour s'assurer du bon déroulement de sa commande. Il peut également utiliser pour mettre a jour son image locale sur fichier manipulé.

2. PROTOCOLE DE COMMUNICATION

Dans la SDDS IH* les opérations d'insertion, de modification de suppression et de recherche simple se déroulent pratiquement de la même façon. Elles suivent toutes la séquence suivante :

1- Rechercher le serveur correct
2- Exécuter l'opération correspondante

Dans ce qui suit, on présente le protocole de recherche simple. Puis les spécificités des autres opérations.

2.1. Protocole de recherche simple

Le client, qui désire rechercher un enregistrement, identifié par le vecteur clé $k=(k_1, k_2,\ldots, k_d)$, est encapsulée dans un datagramme UDP.

Au début le client utilise l'algorithme (Access) pour déterminer l'adresse (a) du serveur susceptible de contenir l'enregistrement k, ce calcul est basés sur son image locale.

Le client détermine, ensuite, a partir de (a) l'adresse physique et le port UDP de ce serveur et lui envoie la commande de recherche. Nous supposons a ce niveau que le client connais les adresses physiques (adresse IP et port UDP) de tous les serveurs du fichier.

En recevant le message de commande de recherche, le serveur (a) vérifier s'il est le bon récipient (la clé k appartient a sa portée).

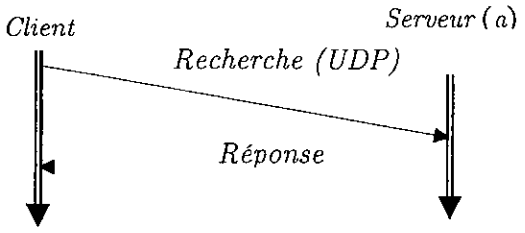


Figure III.2 : Recherche simple

Si le serveur (a) est bien adresser (Figure III.2), il exécute la requête du client et lui retourne la réponse appropriée.

2.2 Protocole d'insertion / modification

Le protocole d'insertion d'un enregistrement par le client suit pratiquement le même schéma du protocole de recherche simple.

Pour ces deux opérations un message d'accusé de contenant l'adresse et le niveau de serveur ayant exécuté la commande, est retourné au client. Le client ce servira de ce message pour s'assurer du bon déroulement de sa commande.

Dans le cas ou l'insertion porte sur un enregistrement qui existe déjà dans le fichier, le serveur ayant pris en charge la commande peut retourner un message d'erreur au client.

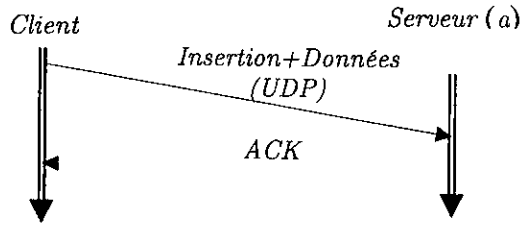


Figure III.3 : Protocole d'insertion et de mise à jour

2.3 Protocole de suppression

Le protocole de suppression est identique à celui de l'insertion. Seulement que pour la présente opération, seul le vecteur clé identifiant l'enregistrement a supprimer sera envoyer avec le message de commande, si l'enregistrement qu'on veut supprimer

n'existe pas dans le fichier, le serveur retournera un message d'erreur au client lui informant de la situation.

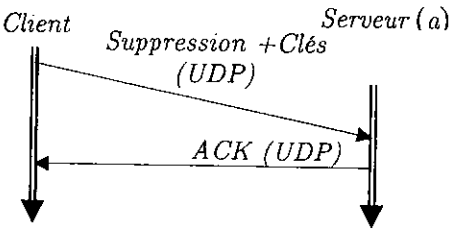


Figure III.4 : Protocole de suppression

3. Protocole d'éclatement

Un fichier IH* est subdivisé en cases implémentées sur plusieurs sites. Chaque site héberge une seule case. cette dernière a un maximum d'enregistrements qu'elle peut contenir. le fichier s'étend graduellement sur l'ensemble des sites au fur et à mesure que les cases éclatent en créant de nouvelles cases sur de nouveaux sites.

C'est à la suite d'une insertion qui fait déborder un serveur SDDS (m) que l'éclatement est déclenché. rappelons que selon l'algorithme IH*,le serveur qui éclate n'est pas forcement celui qui déborde mais celui qui est pointe par le pointeur d'éclatement (n).

Le serveur (m) envoi un avis de débordement au CoS lui informant d'une situation de collision.

Indépendamment du mode d'éclatement, la recherche d'un nouveau site sur lequel devra s'effectuer l'éclatement peut être soit à la charge du serveur (n) soit à la charge du CoS

Pour cela, en présente trois stratégies proposées initialement pour les SDDS RP* et LH*: liste des serveurs, avec site coordinateur [LNS96.a] et réseau de contrat

L'étude des problèmes liés à l'application de ces stratégies à la SDDS IH * a permis de développer une nouvelle stratégie qui profite des avantages de ses stratégies sans pour autant hériter de leurs inconvénients.

3.1 Liste des serveurs

dans cette stratégie la table des adresse des serveur devant participer à l'hébergement des cases du fichier IH* est déterminée à l'avance au niveau de chaque machine serveur.

Lorsque le serveur S (d'adresse n) reçoit un ordre d'éclatement de la part du CoS, il choisit un serveur dans sa table d'allocation de sites locale (SAT, pour site Allocation table) et le contacte pour lui demander d'héberger la nouvelle case. si le serveur ainsi contacté est capable d'héberger la nouvelle case, il envoi un message de service indiquant q'il est prêt à recevoir les données devant migrer.

Si le serveur S ne reçoit pas de message d'acceptation jusqu'à l'expiration d'un time-out, il en déduit que le serveur qui la contacter n'est pas disponible et contacte alors un autre site pris de sa SAT. ce processus boucle jusqu'à la réception d'un message d'acceptation d'hébergement de la part d'un serveur S'.

le serveur S envoi alors au serveur S' un message d'initialisation .A la réception de ce message, le serveur S' se prépare à recevoir la moitié de la case S et accuse réception de la demande d'éclatement .

A l'arrivée de l'accusé de réception, un canal de données est ouvert entre les serveurs S et S' et le transfert des enregistrement est effectué par des messages de transfert de Données, l'acquittement est obligatoire pour chaque paquet reçu afin de permettre au Serveur S de supprimer sur son site les données déjà envoyées en toute sécurité.

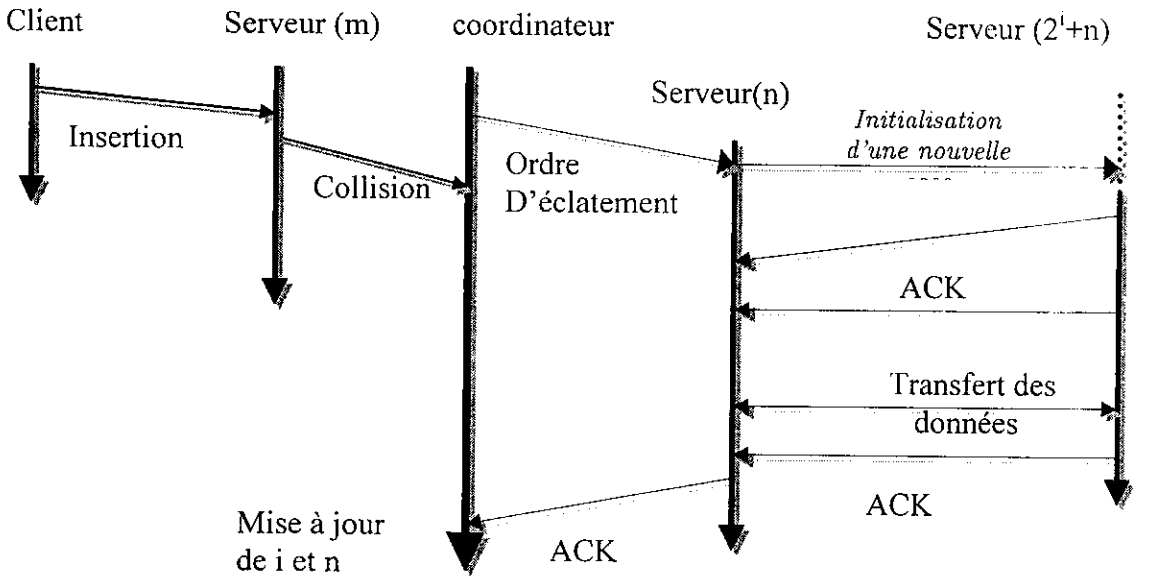


Figure III.5 : Protocole d'éclatement-Liste des serveurs

A la constitution de la nouvelle case d'adresse (2^i+n) au niveau du serveur S', le serveur S informe le site coordinateur du bon déroulement de l'éclatement pour qu'il mettre à jour les paramètres du fichier (n, i).

Comme avantage, cette politique d'éclatement est facile à mettre en œuvre. Cependant, l'ajout de nouveaux serveurs au fichier IH* peut poser le problème de mise à jour globale des SAT au niveau des serveurs SDDS. En effet, la table d'allocation de serveur doit être mise à jour de façon synchrone.

D'un autre cote, le serveur qui désire éclate risque de perde beaucoup de temps avant de trouver un serveur libre compte tenu des délais d'attente cumulés.

3.2 Site coordinateur

Une amélioration de la liste des serveurs au niveau de chaque site consiste à disposer au niveau des sites coordinateur d'une table des adresses des serveurs complètes. les autres machines de la SDDS (client et serveurs) garde une partie seulement de cette table. les règle de gestion de ces table sont comme suit :

- La table d'un client contient les adresses physiques des serveurs inclus dans son image sur le fichier;
- La table d'un serveur contient uniquement les adresses physiques de ses serveurs descendants. de ce fait, un serveur S n'a pas besoin de connaître les adresses des serveurs situés avant lui ni celles des serveur hors de sa porté [BZ02].

A partir de cette règle, on peut voir que seule le serveur 0 qui contient la table complète des adresse (c'est en même temps la première case est la racine du fichier) Ce qui lui permet de jouer le rôle du site coordinateur dans le cas ou ce dernier tombe en panne.

Apres réception du message de débordement, le CoS détermine un site disponible dans sa PAT et lui envoi un préavis d'éclatement. le site désigné se prépare alors pour recevoir les donnés et informe le coordinateur qu'il est prêt à recevoir des données. Le CoS met à jour sa PAT et transmet au serveur S un message d'ordre d'éclatement muni de l'adresse du serveur S' ou doit être logée la nouvelle case d'adresse (2^i+n).

pour des raison de performance, ce message peut également être muni des adresse des descendants de S qui ont été crées depuis son dernier éclatement .

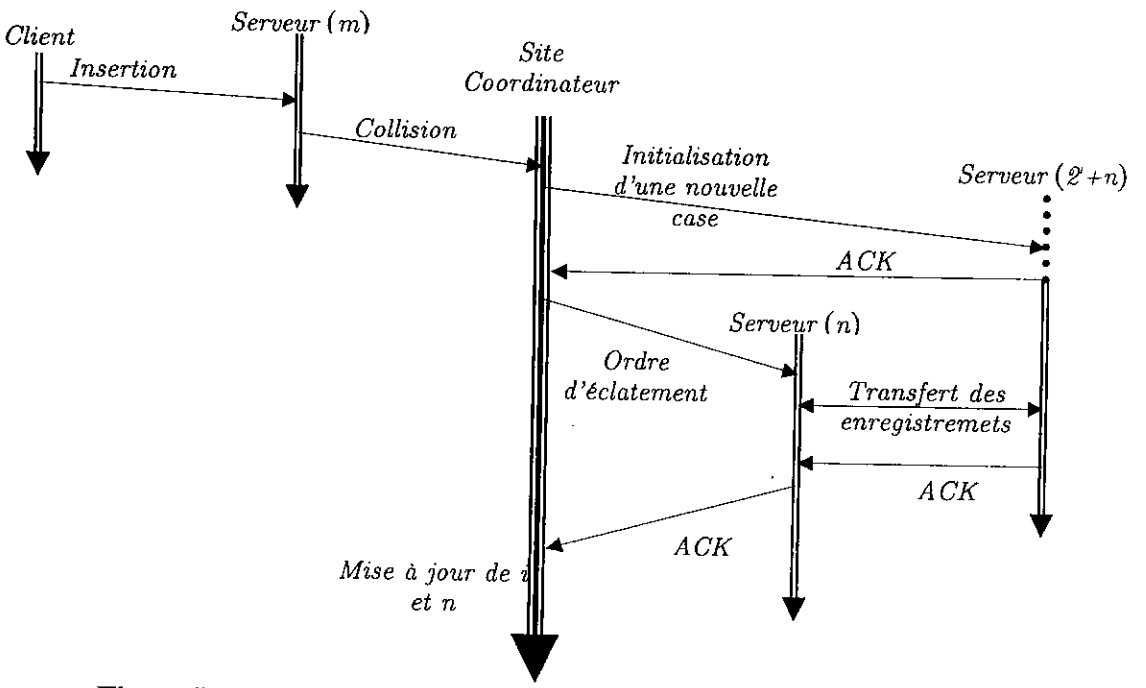


Figure III.6 : Protocole d'éclatement - site coordinateur

Des la réception de ce message, un canal de données est établie entre le serveur

devant éclater S et le serveur d'accueil S' et le transfert est ainsi effectué.

A la fin, un message d'acquittement est envoyé au CoS lui permettant de savoir que L'éclatement s'est correctement terminé. en recevant ce message, le CoS met à Jour les paramètres du fichier IH*.

Comme avantage, cette stratégie :

- Elimine le problème de mise à jour synchrone des SAT locales en cas d'évolution du nombre de serveur.
- Consomme moins d'espace mémoire puisque un serveur ne garde que les adresses de ses descendants .

En contre partie :

- Cette solution nécessite un site coordinateur qui peut facilement devenir un goulot d'étranglement dans le cas ou le fichier subit des éclatements très fréquents, ce qui détériore les performances du système SDDS.
- Si le CoS tombe en panne, le système se trouve alors paralysé et la reprise après résolution de la panne n'est pas toujours évidente SDDS.

Précisons que c'est bien ce protocole qui a été implémenté dans notre prototype.

3.3. Réseaux de contrat

Dans cette strategie, il n'existe pas de table d'allocation de sites ni de site coordinateur. le serveur qui reçoit un ordre d'éclatement envoie une demande de site disponible vers le groupe multicast du fichier IH*, dans lequel il précise la capacité désirée dans la case à créer. le serveur capable de satisfaire la demande d'hébergement renvoi au serveur S un message de candidature à l'hébergement en précisant son adresse parmi les messages de candidature reçu, le serveur S choisit un serveur S' (le premier par exemple) et lui envoie un message d'initialisation. a la réception d'un accusé d'initialisation, le serveur S ouvre un canal de données pour la migration des enregistrements vers la nouvelle case S'. Après réception d'un accusé de réception de données, le serveur S rajoute l'adresse de S' dans sa table d'allocation de sites locale [BB04].

Dans cette technique, les serveurs ne connaissent pas toutes les adresses physiques des autres serveurs. il appartient donc à chaque machine de collecter les adresses physiques chaque fois que cela est possible.

Comme avantages, cette solution évite l'emploi d'un site coordinateur. Elle constitue, également, un meilleur chois en cas d'évolution du nombre de serveurs car elle rend le système plus souple.

Cependant, le nombre de messages dans le réseau augmente légèrement à cause des messages multicast et du passage des adresses physiques lors des éclatements. de plus, Les machines auront souvent recours aux messages multicast pour rechercher un site libre ou déterminer les adresses qu'elles ne connaissent pas.

3.4 Protocole d'éclatement du IH*

Apres une étude approfondie des propriétés de la SDDS IH* et une analyse Minutieuse des précédentes stratégies d'éclatement nous avons pu Développer une nouvelle stratégie d'éclatement qui combine les précédentes stratégies en optimisant certains de leurs aspects.

En recevant un message de débordement, le CoS contact un site disponible Dans sa PAT en lui envoyant un message d'initialisation à l'éclatement Contenant l'adresse physique du site S qui doit éclater. Le site S' qui a reçu Le message d'initialisation et qui est capable d'héberger la nouvelle case se Prépare à la préparation de données.

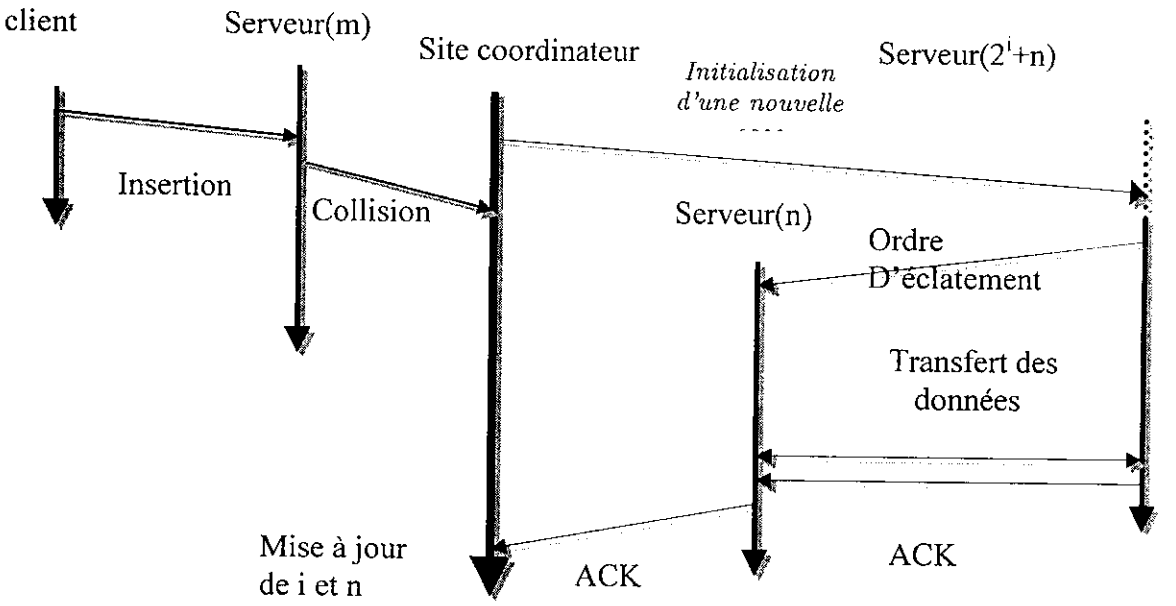


Figure III.7 : nouveau protocole d'éclatement du IH*

Contrairement à la stratégie avec site coordinateur, le serveur S' envoi, dans Ce cas, un ordre d'éclatement au serveur S, sous l'autorité du CoS.

A la réception de ce message, un canal de données est etablie entre le Serveur à éclater S et le serveur d'accueil S' et le transfert des Enregistrements est ainsi effectué.

A la fin du transfert, le serveur S envoi un message d'acquittement au CoS Lui informant du bon déroulement de l'opération d'éclatement.

Comme avantage cette stratégie permet de réduire de (-1) le nombre de Messages utilisés par les deux premières stratégies. de plus elle permet de charger le CoS des taches de choix et d'initialisation de la case D'accueil. c'est le serveur S qui s'en occupe puisque c'est lui qui possède Toutes les informations nécessaires et suffisantes pour le faire.

4. L'architecture du système

L'architecture du système est faite principalement pour que chaque machine d'un multiordinateur participe activement à la gestion des fichiers de la SDDS. A l'opposer, l'architecture la plus répandue est composée d'un ou plusieurs serveurs de forte puissance (processeur et mémoire) qui dessert un réseau local [CLR95].

Pour cela en présente une architecture constituée de plusieurs processus coopératifs mais fonctionnellement indépendants :

- Client ;
- Serveurs;
- Site coordinateur;

Cette séparation offre plusieurs avantages. Il est ainsi possible d'offrir plusieurs qualités de service et diverses fonctionnalités par assemblage des modules. Ainsi l'expérimentation ou la gestion s'en trouve considérablement simplifiée.

D'un point de vue purement technique, cette modularité permet de dégager précisément le rôle de chaque composante du système, et ainsi de comprendre les relations exactes entre serveurs. Il n'est pas négligeable non plus de rappeler qu'une telle séparation facilite grandement la réflexion et l'implantation a l'intérieur d'une équipe.

Dans cette partie nous détaillerons l'architecture interne et le rôle de chacune des applications SDDS, sans pour autant rentrer dans des considérations algorithmiques, en faisant, toute fois, ressortir leurs récupérations vis-à-vis des performances de la disponibilité et de la fiabilité. Ainsi que les interaction générales et qu'ils ont entre eux.

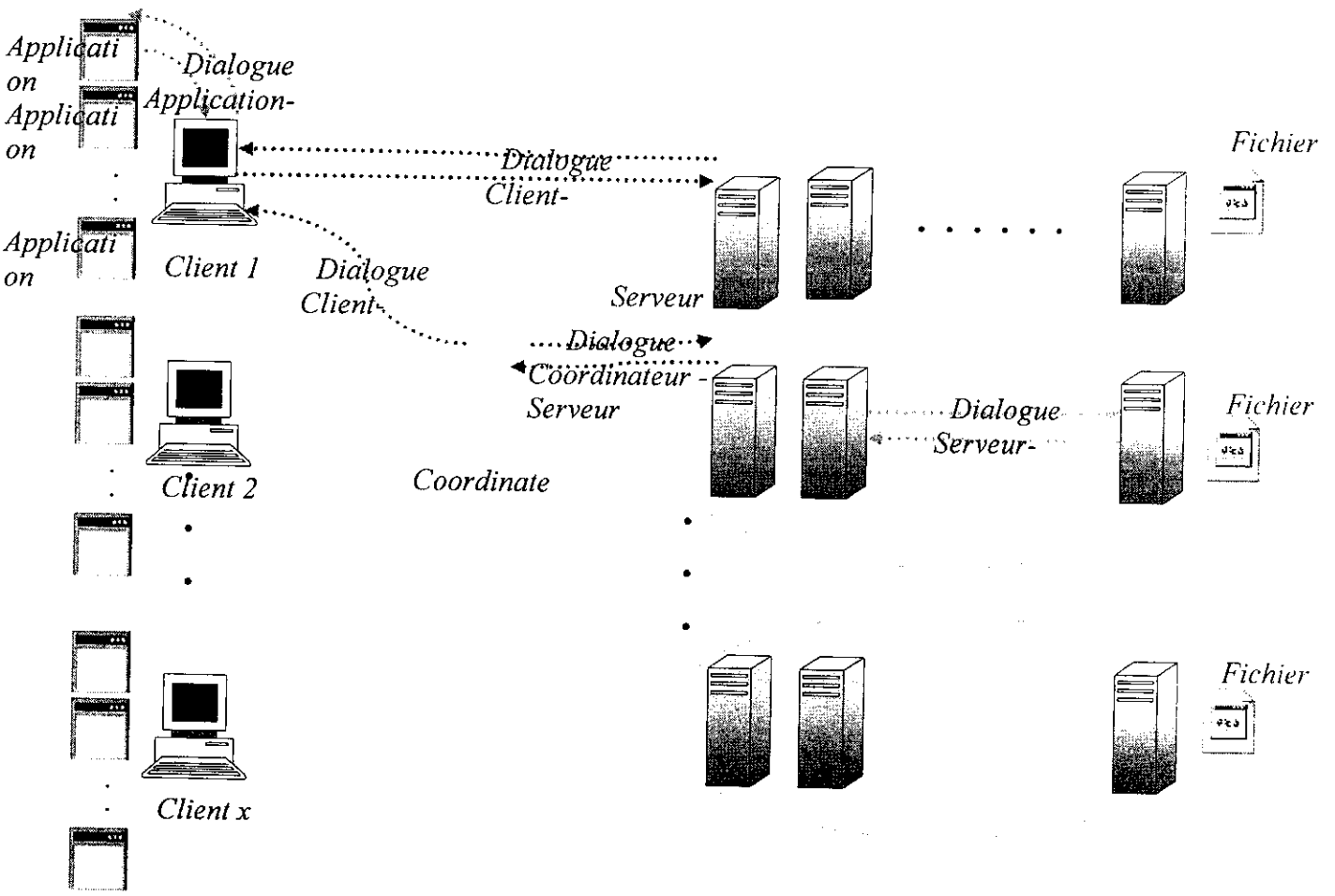


Figure III.8 : Architecture Générale

4.1. SERVEUR SDDS

Le serveur SDDS a pour charge le stockage de façon persistante des données des fichiers SDDS. Il constitue l'entité virtuelle qui représente une case du fichier IH*. Il regroupe, également, tous les modules permettant de gérer le stockage et les accès concurrents aux données.

Le serveur SDDS dispose d'un port d'écoute UDP. Ce port sert, a recevoir les requête des clients SDDS et les message de services.

D'un autre coté, cette voie UDP est utilisée pour envoyer des réponses vers les clients demandeurs ou des messages de service vers le CoS.

Le serveur inclut une zone mémoire qui constitue la case IH* proprement dite.

4.2. Client SDDS

Le client est la machine du système qui s'occupe de l'envoi des requêtes (insertion, recherche ...).

Le client calcule l'adresse du serveur auquel doit être envoyée la requête. Ce calcul dépend évidement sur l'image-client correspondante au handle d'un fichier concerné par la requête. A cet effet, le client dispose d'une table d'accès au fichiers local (FAT pour File Access Table) qui fournit la correspondance entre le handle d'un fichier IH* et ses paramètres de hachage (n', i').

Une fois le numéro du serveur, au quelle doit être envoyée la requête, est déterminé, le client procède a la résolution de l'adresse physique (*@IP, Port UDP*) de la machine sur laquelle est lancé ce serveur en adoptant le même mécanisme de résolution des adresses physique utilisé par le serveur. Le client envoi, ensuite, sa requête a la machine identifiée par (*@IP, UDP*) en utilisant le protocole UDP. le client reste à l'écoute sur un port UDP des réponses des serveurs.

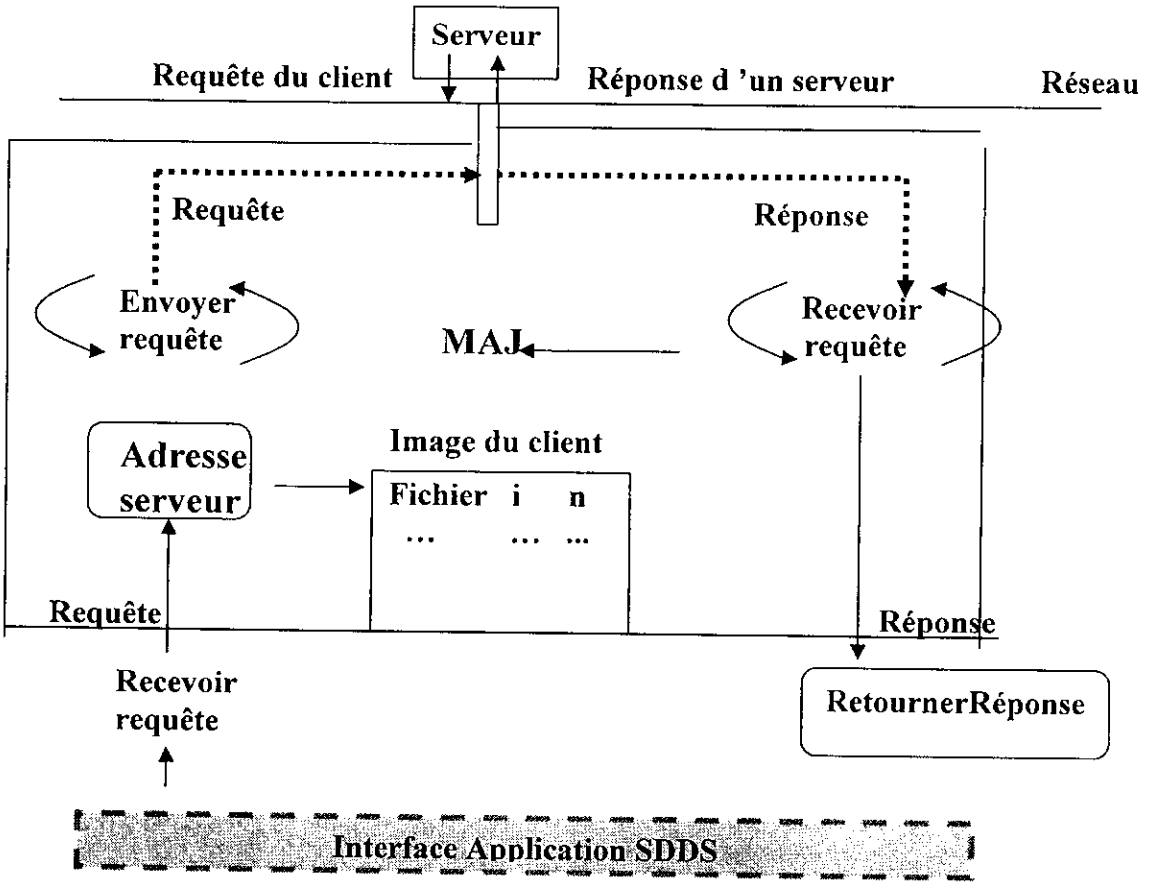


Figure III.9 : Architecture d'un client/serveur

4.3. Site de Coordinateur

Sachant que les serveurs ignorent le numéro du prochain serveur a éclater, il doit y avoir pour cela un serveur spécialisé qui sait déterminer avec exactitude la case (n) qui doit éclater au moment d'une collision. Ce serveur nommé « site coordinateur » peut être l'une des cases du fichier, la plupart du temps c'est le serveur d'adresse 0 puisqu'il est le premier a être crée [LNS96, KLR96, Ben00].

Le coordinateur garde les paramètres (n, i) du fichier. Il est, surtout, responsable du déclenchement des éclatement, c-a-d de décider quand et quelle case doit éclater puisque les case en débordement envoient des message de collision au coordinateur qui est libre d'en tenir compte ou pas, selon la charge du fichier et le mode d'éclatement adopté.

Le site coordinateur de notre prototype adopte le mécanisme d'éclatement décrit dans (& III 3.2) et pour repérer les serveurs du fichier, il utilise table des adresses physiques dynamiques.

Théoriquement, chaque fichier SDDS a son propre coordinateur qui est l'un de ses serveurs (le serveur 0 par exemple), mais dans un souci de simplicité nous avons préféré écrire un coordinateur a part au lieu de rajouter ses algorithmes au niveau d'un serveur IH* afin qu'il peut assurer la fonction de coordination.

De ce fait, chaque fichier possède un site coordinateur à part et un ensemble de serveurs pour le stockage de ses cases.

5. Réalisation d'un serveur pour la SDDS IH*

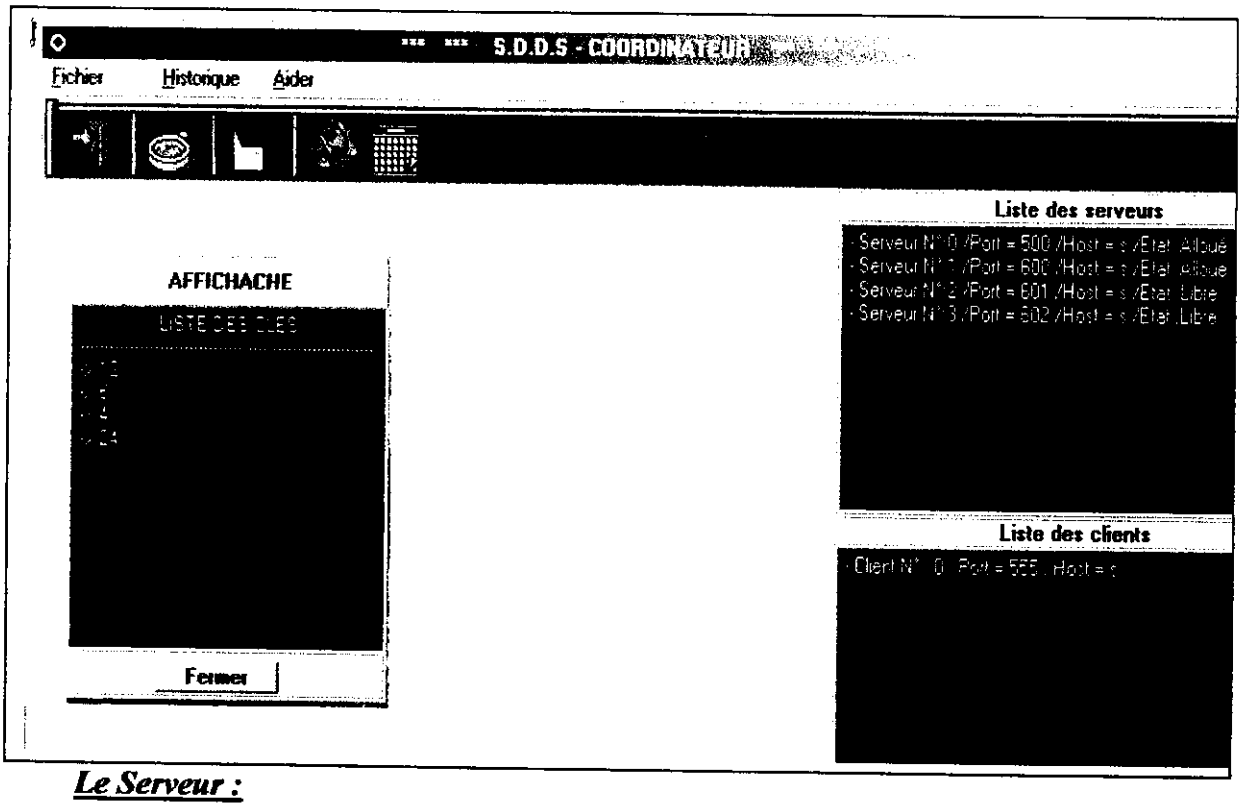
Notre projet consiste de trois applications différentes et qu'ils sont tous coopèrent pour le bon fonctionnement de leurs exécutions. Ces applications sont ***le coordinateur, le client, le serveur***

Ainsi pour le bon fonctionnement du projet, ce dernier doit tourner sur un réseau local et un ordre de lancement des applications doit être respecté. Premièrement, on lance le ***coordinateur*** puis ***le Serveur ou bien le client peut import.***

Maintenant, on va détailler les interfaces de ces applications suivant l'ordre de leurs lancement.

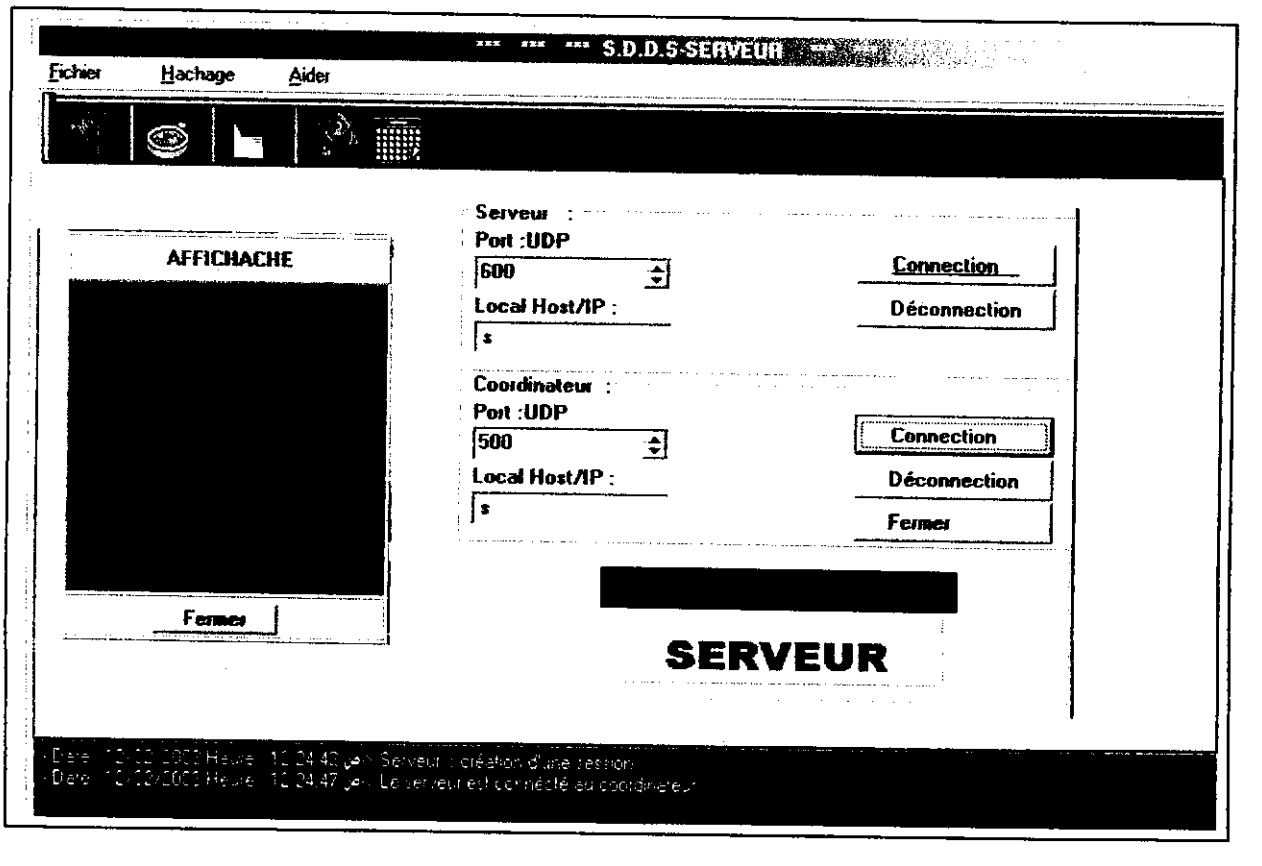
Le Coordinateur :

Le ***Coordinateur*** est l'application qui contient les informations (liste des serveur connectes, liste des clients, table d'affichage…..) et c'est lui qui décide le moment d'éclatement et la case (un serveur) qui doit éclater. Il contient
aussi la liste des machines libres. Le ***Coordinateur*** a la forme suivante :

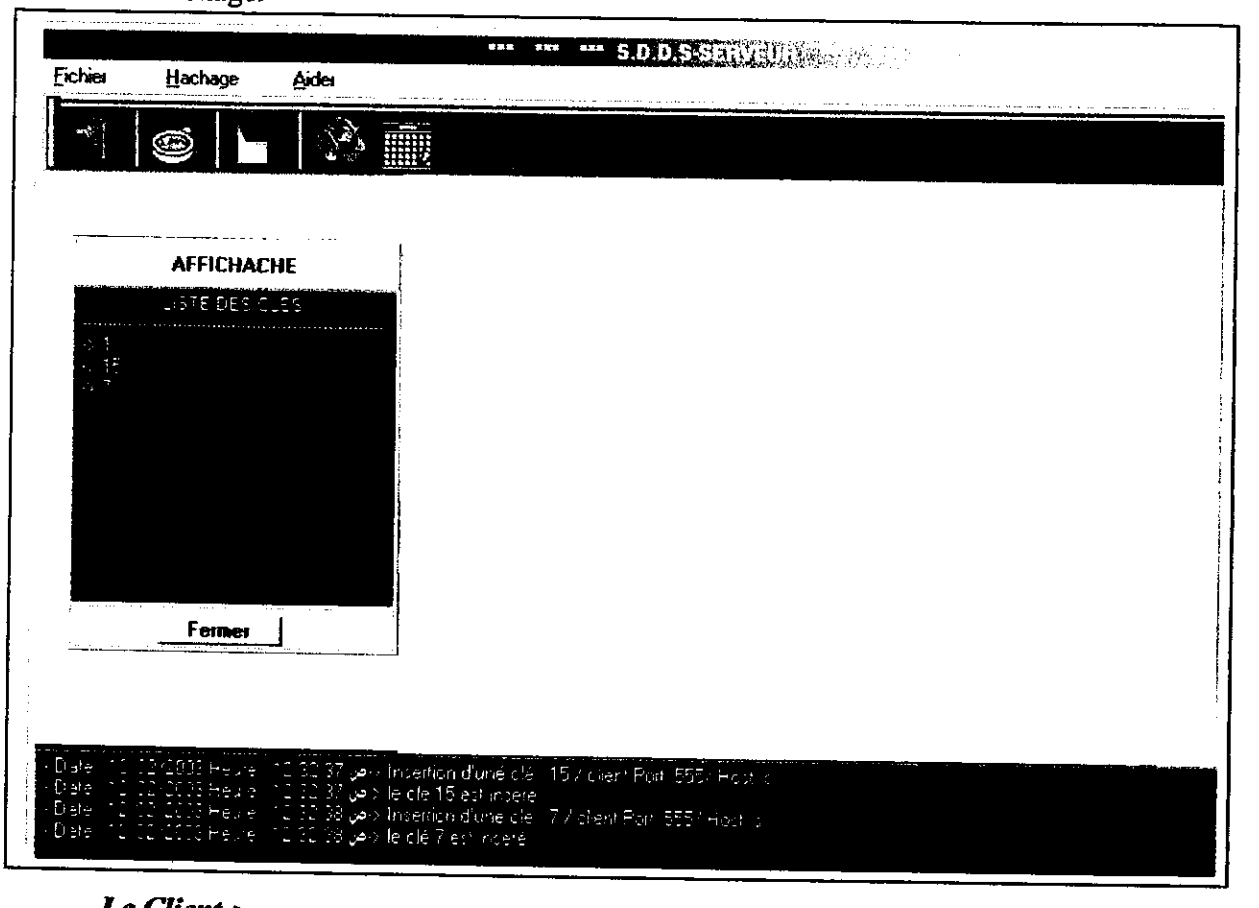


Le Serveur :

Le serveur se connecte avec le coordinateur en appuyant sur le bouton connexion et en spécifiant son adresse et son port. Et peut se déconnecter au coordinateur en cliquant sur le bouton déconnexion.

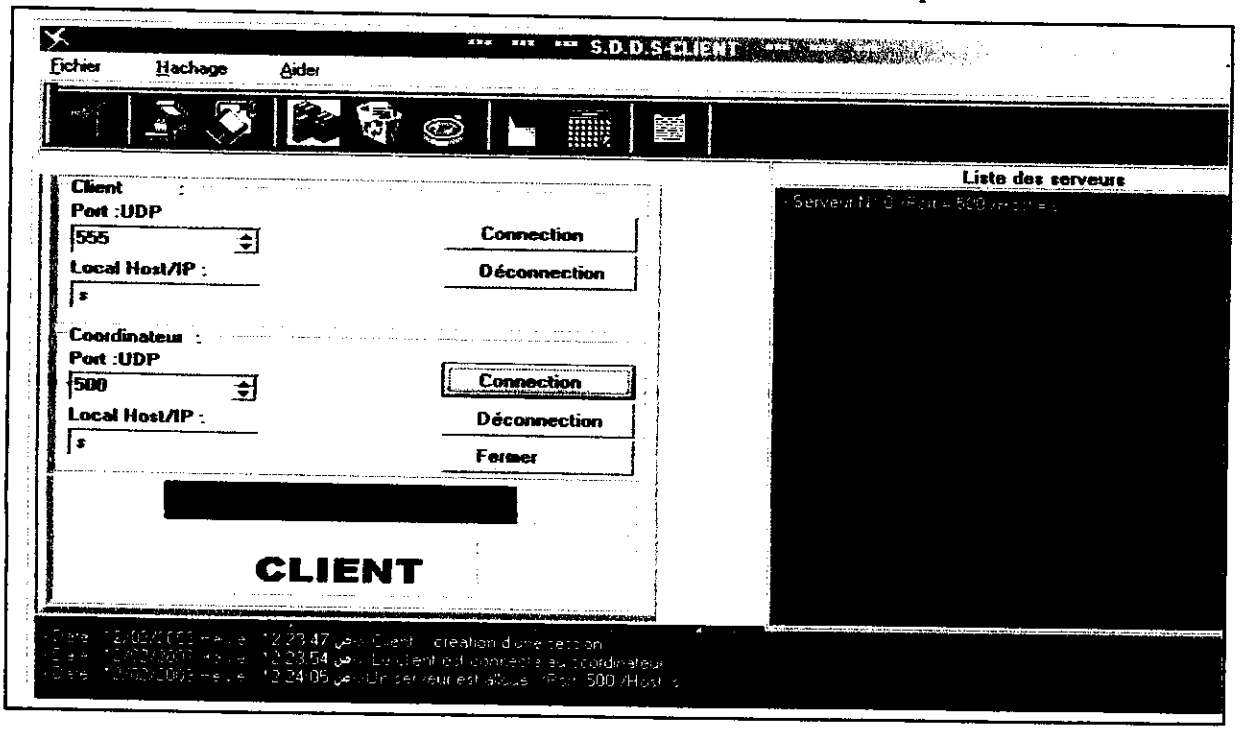


Le ***Serveur*** est l'application qui reçoit les opérations (insertion, recherche,...) émises par le client. C'est le serveur qui stocke les clés pour cela il possède une table d'affichage.

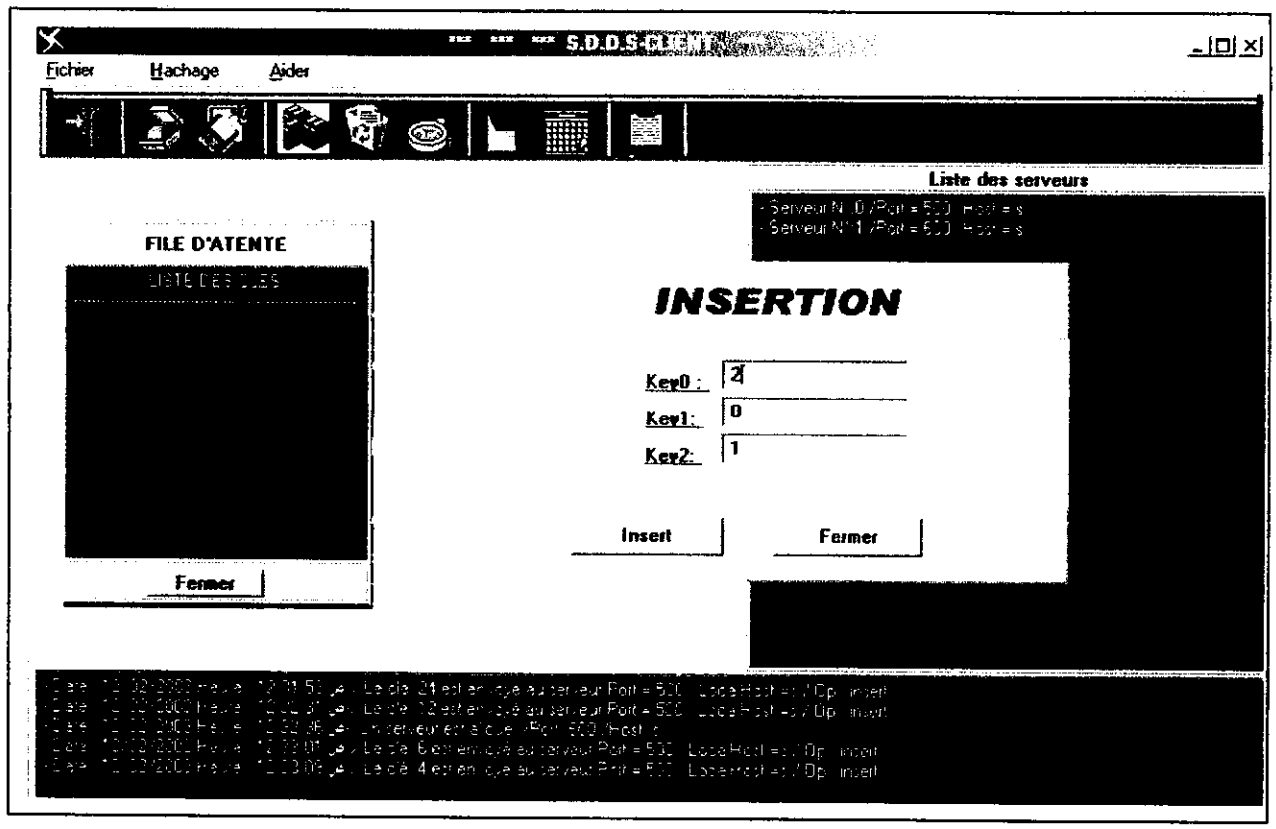


Le Client :

Le client se connecte avec le coordinateur, en cliquant sur le bouton connexion, en cliquant sur le bouton connexion, en spécifiant son adresse et son port.



Le client est une application qui contient des boutons pour les opérations d'insertion de suppression...etc. pour insérer une clé on click sur le bouton insérer, une boite de dialogue apparaît qui contient des cases pour insérer la clé.



6. Conclusion

Tout au long de ce chapitre on présente les différents types de messages manipulés, et on décrit les protocoles de différentes opérations sur un fichier IH*(insertion, modification,....), ainsi que les différentes stratégies d'éclatement.

On a aussi présenté l'architecture interne des machines SDDS. Cette architecture basée sur trois processus coopératifs fonctionnellement indépendant qui rend facile l'extension de leurs fonctionnalités est permet une large portabilité vers différents systèmes.

Pour l'implémentation nous étions amené à utiliser certaines techniques avancées de programmation telles que les Windows socket.

A la fin de ce chapitre on présente les différentes interfaces de la réalisation d'un serveur pour la SDDS IH*.

CONCLUSION & PERSPECTIVES

Conclusion et perspectives

Nous avons eu comme objectif la conception d'un serveur pour la SDDS IH*. Cette étude nous a amené à examiner et à comparer les performances des différentes méthodes d'organisation pour mieux orienter notre choix.

Le travail de recherche bibliographique nous a permis d'approfondir nos connaissances sur les systèmes distribués, les principes des multiordinateurs et des SDDS. Il nous a permis, également, d'étudier les techniques de programmation avancée sous WINDOWS. La maquette a été réalisée en C++ sous WINDOWS.

Le modèle client / serveur est particulièrement adapté aux environnements informatiques distribuées. Il est aussi admis, actuellement, que les mémoires valent de moins en moins cher et les réseaux sont plus en plus rapides. Toutes ces raisons ont favorisé la pertinence des multiordinateurs qui offrent des performances théoriques de traitement impossibles pour les systèmes classiques. Les SDDSs permettent l'exploitation de ces capacités. Elles offrent la possibilité d'utiliser des données de quantité, théoriquement non limitée, distribuées sur un réseau dont l'étendue et la capacité de stockage sont supposées illimitées.

Ce projet contribue à démontrer la faisabilité des SDDSs qui offrent de nouvelles possibilités pour de nombreuses applications. Plusieurs problèmes qui font l'objet de recherches sur les systèmes distribués méritent une étude approfondie. Il s'agit notamment de l'application des SDDSs aux bases de données multimédia où les volumes de données sont très importants.

Comme perspectives nous proposons :

- La comparaison de la méthode IH* avec d'autres SDDSs afin de situer ses performances.
- La conception d'un schéma pour l'organisation interne d'une case SDDS.
- La conception d'un mécanisme efficace pour la sauvegarde des informations des serveurs SDDS sur disque dans le but de leur permettre de retrouver leur état.
- La conception d'une solution offrant une disponibilité des informations des serveurs après les pannes.

REFERENCES BIBLIOGRAPHIQUES

Bibliographies

On s'est basé dans notre mémoire beaucoup sur la thèse de Mr boukhelef. Dj

[ACP95] **Anderson.T, Culler.D, E. and Patterson D:** a case for now (network of workstations). IEEE Micro, 12(1):54-64, February 1995.

[BB04] **S. Belaid, A.Boularias :** Implémentation et étude des performances d'une SDDS CTH*sous Windows 2000, mémoire d'ingénieur, Institut national de formation en informatique (I.N.I) Oued-Smar Alger 2004.

[Ben00] **F. Bennour :** un gestionnaire de structure de données distribuées et scalables pour les multiordinateurs windows : application a la fragmentation par hachage, these de doctorat de l'université parix IX dauphine. Soutenue le 30 juin 2000.

[Bou02] **Boukhelef. Dj :** extension du hachage par interpolation aux environnements distribues IH*, thèse de magister en informatique Institut national de formation en informatique (I.N.I) Oued-Smar Alger, février 2000.

[BUR 83] **B.A.Burkhard** –interpolation –based index maintenance .proc .of the 2 symposium on principe of databases systems .PODS 1983.pp.76-88.

[BVW95] **Y.Beritbart , R .Vingralek & G. Weikum**–load contrôle in distributed file structure. technicale report 254-95, departement of computer science, university of KENTUCKY, lexington , KY. february 1995.

[BZ02] **D.Boukhelef & D.E.Zegour** -IH* : A New Hash-Based Multidimensional SDDS Workshop on Distributed Data & Structures –WDAS-2002 Université Parix IX Dauphine .France.20-23 mars 2002.

[C94] **Culler,D. NOW**: Towards Everyday Supercomputing on a Network of Workstations. EECS Tech. Rep. UC Berkeley, to app.

[CLR95] **P.Chevochot, J.P.lesot & F.Roger**-systeme de gestion de fichier distribue : migration et replication dynamique comme approche a l'amelioration des performance. Rapport de stage. Université de Pierre et marie curie. Paris VI. 26 September 1995.

[Dev93] **R.Devine** – desgin and implementation of DDH :a distributed dynamic hashing algorithm .4 intel. conf. on foundation of data organizations and algotithms (FODO-93), chicago (oct.1993). lecture notes in computer science, springer verlag (publication).1993.

[DG92] **D.G.Dewitt, J.N. Gray:** parallel data base systems: the future of high performance data base sytems, communication of the ACM vol. 35 N° 6, june 1992, pages 85-98.

[kar97] **J.S.Kalsson** –a scalable distributed data structure for A Parallel data Server. Muster thesin n°. 609. department of computer science and information science, linkoping university, February 11, 1997.

[Kar98] **J.S.Kalsson** –a scalable distributed data structure for high –performance spatia access. intl. conference on foundations of data organization, FODO 98.pp37-46, kobe, japan, nouvember 1998.

[KLR96] **Jonas. S. Kalsson, Witold Litwin, and Tore Rich:** LH*th: A Scalable High performace Data Structure for Switched Multicomputer- in Avdance in Database Technology-EDBT96, page 573-791, Avignon, France, March 1996. Springer.

[KNU73] **Knuth. D.E.** The Art of Computer Programming, Sorting and Searching Vol 3. Addison Wesly Publishing Company, Reading, Massachusetts 1973.

[KW94] **Kroll, B, Widmayer**, P. Distributing a Search Tree Among a Growing Number of Processors. To app. at ACM-SIGMOD Int. Conf. On Management of Data, 1994.

[L95] **Litwin,W:** redundant arrays of LH* files for high a valaibility and security. Techn note GERM paris 9 & distributed inf, techn. Dep. HPL PALO Alto, sep.1995.

[LNS93] **Litwin.W, Neimat, M-A. Schneider**, D. LH*: Linear Hashing for Distributed Files. ACM-SIGMOD Int. Conf. On Management of Data, Washington D.C.p.327-336, may, 1993.

[LNS93.a] **Litwin.W, Neimat, M-A. Schneider**, D. LH*: Linear Hashing for Distributed Files. ACM-transaction on Database systems. Dec 1993.

[LNS93.b] **Litwin,W, Neimat, M-A.Schneider**,D. LH*: Linear Hashing for Distributed Files. ACM-SIGMOD Int. Conf. On Management of Data, 1993.

[LNS94] **Litwin.W, Neimat, M-A.Schneider**, D. RP*: A Family of Order Preserving Scalable Distributed Data Structures. HPL-DTD-94-012, (Feb. 1994).

[LNS96] **Litwin.W, Neimat, M-A. Schneider, D.** K-RP*s: Scalable Distributed Data Structure. for high–performance multi-attribute acces. PDIS1996. PP.120-131, 1996.

[LNS96.a] **Litwin.W, M-A Neimat.** LH*s: a high-availability and high-security Scalable Distributed Data Structure. IEEE Workshop on Research Issues in Data Engineering. IEEE Press, 1997 (to app.).

[LNS96.b] **Litwin.W, M-A Neimat & D. Schneider** – K-RP*s: Scalable Distributed Data Structure .for high –performance multi-attribute access. PDIS1996. PP. 120-131. 1996.

[LMR98] **Litwin.W, L.Menon & T.Risch** – LH*schemes with scalable availability IBM almaden research report RJ10121(91937). may, 1998.

[LN95] **Litwin,W, Neimat. K-RP***: a family of high performance multi-attribute scalable distributed data structure. To upp. In IEEE Intl. conf. on par. & distr; systems, PDIS-96, (Des. 1995).

[LN96] **Litwin,W. Neimat. A.M.** High a velebility LH* schemes with merorring, intl. conf on cooperating systems, Brussels, IEEE presse 1996.

[LR97] **Litwin.W & T.Risch** –LH^*_g high –a velabilité scalable distributed data structure through record grouping. CERIA technicale reporte. university parix XI dauphine. may, 1997. (submitted for bublication).

[LRMS00] **Litwin.W, J.Menon, T. Risch &T. Schwarz** –issues for scalable available LH* schemes with record grouping. distributed data structures DIMCAS, may 1999. carleton scientific, (pub 2000).

[LS99] **Litwin.W, T, SCHWARTZ, S. j:** LH*RS, a High Availability Scalable Distributed Data Structure Using Reed Solomon Codes. Rapport technique U paris IX Dauphine 1999.

[Ndi98] **Y.Ben Abdelkader Ndaye**- mise en euvre de l'architecture de communication des SDDS RP^*_n et RP^*_c . mémoire de DEA. Departement informatique université cheikh anta diop de dakar. 27 juin.

[PN00] **A. Di Pasquale & E.Nardelli**–fully dynamic balanced and distributed search trees with logarithmic costes. in proceeding of workshop en

distributed data and stucture (WDAS99), pp.73-90, princeton, Nj, carleton scientific, may 1999.

[PN00.a&b] **A.Di Pasqual & E.Nardelli** – scalable distributed data structure: a surevey , in proceeding of workshop en distributed data and structure (WDAS2000), L Aquila, carleton scientific. june 2000.

[PN00.b] **A. Di Pasqual & E.Nardelli**– scalable distributed data structure: a surevey, in proceeding of workshop en distributed data and stucture (WDAS2000), L. Aquila, carleton scientific. june 2000.

[SDDS] Scalable Distributed Data Structure a new class of data structure for multicoputers .site web de ceria.dauphine.fr/SDDS- bibliograhie.

[Y98] **Yakham Ben Abdel Ndiaye:** mise en oeuvre de l'architecture de communication des Structures de Données et Scalable RP*N et RP*C- Memoire de DEA, université de Dakar-segegal.

PROGRAMMATION RESEAU: TCP/IP & SOCKETS

[Ben01] **B.Beej**-Beej's Guide to Network Programming Using Internet Sockets.Revision Version 2.1.0 May 3, 2001.

[Bes94] **L.Besaw**-Berkeley UNIX Systeme Calls and Interprocess Communication. CS 640 Fall 1994. BCD Socket Reference. Jannuary, 1987 Revised, September 1987, Jannuary 1991 by Marvin Solomon.

[JRZ94] **A. Jazouli, N. E. Radi &T. Zghal**-Programmation réseau sur TCP/IP : l'interface des sockets. E. N. S. I. M. A. G. 1993 / 1994.

ANNEXE

HACHAGE PAR INTERPOLATION

Les structures de fichiers modernes basées sur le hachage dynamique s'avèrent très performantes. Dans ces nouvelles structures, les insertions et les suppressions se font pratiquement sans dégradation de performance. Leur découverte est une avance majeure dans l'étude des structures de données. La plupart de ces structures utilisent un espace intermédiaire pour l'accès aux données.

Le hachage dynamique peut être vu comme une généralisation du hachage classique dans lequel :

- la fonction de hachage est dynamique, c'est-à-dire qu'elle peut varier.
- l'espace des adresses possible n'est plus limité.

Le hachage par interpolation (HI pour interpolation-based hashing) de **W.A.Burkhard [BUR 83]** est une adaptation du hachage linéaire de **W.Litwin** qui préserve l'ordre des clés. La technique est conçue pour gérer des grands fichiers dynamiques et multi clés, elle n'utilise pratiquement pas d'index. Le temps d'accès est au voisinage de 1. Le chargement du fichier est contrôlable par l'utilisateur. Dans ce qui suit, nous rappelons quelques éléments de base qui nous permettent de présenter la méthode. Nous donnerons également la fonction de division utilisée.

1. Définition et notations

L'ensemble de clés **K** est une partie de l'ensemble d-dimentionnel $[0,1[^d$.

$$K = [0,1[^d = [0,1[\times [0,1[\times \ldots\ldots [0,1[\times$$

L'intervalle $\mathbf{I_j}$ est l'ensemble des clés x appartenant à l'intervalle

$x \in \mathbf{I_j} = [\mathbf{min_j}, \mathbf{max_j}[\quad \Leftrightarrow \quad \mathbf{min_i} \leq \mathbf{x} < \mathbf{max_j}.$

Une région R de K est le produit de d-intervalles :

$$\mathbf{R} = [min_1, max_1[\times [min_2, max_2[\times \ldots\ldots [min_d, max_d[.$$

Remarque : L'espace des clé K est lui-même une région

Un enregistrement est identifie par un vecteur clé d-dimensionnel $k=(k_1, k_2, \ldots, k_d)$ ou chaque k_j $(1 \leq j \leq d)$ prend ces valeurs dans un domaine ordonné $D_j = [0, 1[$.

1.1 Ordre linéaire sur K

On utilise un ordre linéaire sur **K** noté $\leq_s$ pour ranger et retrouver des articles dans le fichier. cet ordre est défini comme suit :

Si $\mathbf{k_1}$ et $\mathbf{k_2}$ sont deux points de **K**

$$\mathbf{S(k_1) \leq S(k_2) \Rightarrow k_1 \leq_s k_2}$$

S est une fonction de projection de l'espace d-dimensionnel $\mathbf{K} = [\mathbf{0},\mathbf{1}[^d$. sur l'espace unidimensionnel[0, 1[.

S est définie comme suit :

$$S(k) = \Sigma_{i\geq 1} \ \Sigma_{1 \leq j \leq d} \ k_j^i \ 2^{-d.i+j+d} = \Sigma_{i\geq 1} \ \Sigma_{1 \leq j \leq d} \ k_j^i \ 2^{(1-i)d+j}$$

Les k_j^i représentent les coefficients de la représentation binaire de la composante k_j

$$\begin{cases} k_j = k_j^1 \ 2^{-1} + k_j^2 \ 2^{-2} + \ldots\ldots + k_j^i \ 2^{-i} + \ldots\ldots \\ \text{ou} \\ k_j = \Sigma_{i\geq 1} \ k_j^i \ 2^{-i} \end{cases}$$

Enfin, les k_j^i $(1 \leq j \leq d)$ désignent les composantes d'un vecteur de **K**. on observe les propriétés suivantes sur **K**:

- $0 \leq S(k) < 1$
- $k_1 = k_2 \Rightarrow S(k_1) = S(k_2)$

Définissons maintenant l'opération $\mathbf{P_j}$ $(1 \leq j \leq d)$ sur une clé **k** de **K** comme suit :

$$P_j(k) = (0,\ldots\ldots,0.\ k_j,\ 0,\ldots..,0)$$

$\mathbf{P_j(k)}$ est donc la clé pour laquelle la j-eme composant est égale a $\mathbf{k_j}$ de **K** et dont les (d-1) autres composants sont nulles.

Pour toute clé **k** de **K**

$$\mathbf{S(k) = \Sigma_{1 \leq j \leq d} S(P_j(k))}$$

1.2 Fonction π sur N

On définit une fonction notée π_i qui associe a tout réel dans L'intervalle **[0 ,1[**un entier Compris entre 0 et 2^{i-1}. i étant un entier naturel.

$$\pi_i : [0,1[\rightarrow \{0, 1,\ldots 2^{i-1}\}$$

$$\pi_i(x) = [2^i . x]$$

[x] désigne la partie entière de x.

Il est facile de voir que :

$$\begin{cases} \pi_{i+1}(x) = 2^i . \pi_i(x) \\ \pi_{i+1}(x) = 2^i . \pi_i(x) + 1 \end{cases}$$

1.3 Relation d'équivalence sur K

Au moyen de la fonction composée **(π0 S)** on définit une relation d'équivalence sur **K** de la manière suivante.

$$k_1 = k_2 \Leftrightarrow \pi_i(S(k_1)) = \pi_i(S(k_2))$$

Pour tout entier naturel i, on dit que π_{i+1} raffine π_i .le nombre de classes définies par π_{i+1} est le double du nombre de classes définies par π_i le nombre de classes d'équivalence de π_i est égal a 2^i .ces classes d'équivalence sont d'égal volume.

1.4 Constructions de la Fonction d'accès (fonction de division) :

En fait, il ne s'agit pas de construire une fonctions d'accès mais plutôt, une séquence de fonctions. Dans ce qui suit, on montre la construction de h_i. remarquons d'abord que :

$$\begin{cases} \pi_{i+1}(S(k)) = 2\,\pi_i(S(k)) = 2j & k \in R_{2j}^{i+1} \\ \pi_{i+1}(S(k)) = 2\pi_i(S((k)) = 2j+1 & k \in R_{2j+1}^{i+1} \end{cases}$$

Comme nous voulons construire des fonctions h_i qui soient de la forme (2), on introduit alors la fonction de récurrence suivante :

$$\begin{cases} \mathbf{b(0 , n) = 0} & \text{pour tout n} \\ \mathbf{b(i+1 ,n) =b(i , n/2)} & \text{pour tout n pair} \\ \mathbf{b(i+1, n) = b(i , [n/2]) + 2^i} & \text{pour tout n impair} \end{cases}$$

Pour tout entier **i** et **j** ($0 \leq j < 2^i$),

$$\mathbf{b(i+1, \pi_{i+1}(S(k)))} = \begin{cases} \mathbf{b(i , \pi_i (S (k)))} & \mathbf{k} \in R_{2j}^{i+1} \\ \mathbf{b(i, \pi_i (S (k)))+2^i} & \mathbf{k} \in R_{2j+1}^{i+1} \end{cases}$$

Preuve: voire [Aid90, Bur83]

La suite des fonctions **H** est donc obtenue par la suite des fonctions composées **(boπoS)** d'ou :

$$\mathbf{h_i = (b \circ \pi_i \circ S)}$$

Remarque : l'espace des clés peut être vue comme une région de l'espace des réel construit par interpolation élément par élément des déferent dimension d'où le nom de la méthode « hachage par interpolation ».

2. OPERATIONS

L'éclatement d'une région R_j^i donne lieu aux régions R_{2j}^{i+1} et R_{2j+1}^{i+1} nous dirons que R_j^i est une région d'ordre i (R_0^0 est la région d'ordre 0). de plus si k est diviser en j régions de même ordre, alors $j=2^i$. supposons maintenant que k soit divisé en j régions et que nous soyons contraint d'étendre l'espace des adresse (situation de collision). dans ce cas, le IH adopte la même solution que le LH, c-a-d on étend de (+1) la plage des adresses et ce en éclatant les régions R_j^i une par une et dans l'ordre.

Rappelons que cette solution, nécessite l'entretien de deux variable: n et i qui servent a décrire a tout moment la partition de k. en effet, i servira a décrire le niveau de régions définissant la partition, n est le numéro de la prochaine case a éclater .

Plus précisément, l'éclatement provoque l'extension de (+1) de l'espace adressable, et concerne toujours la case (n). soit R_j^i cette case, l'éclatement de la case (n) revient a créer une nouvelle case d'adresse ($n'=n+2^i$), et a transférer les enregistrements correspondants vers cette case.

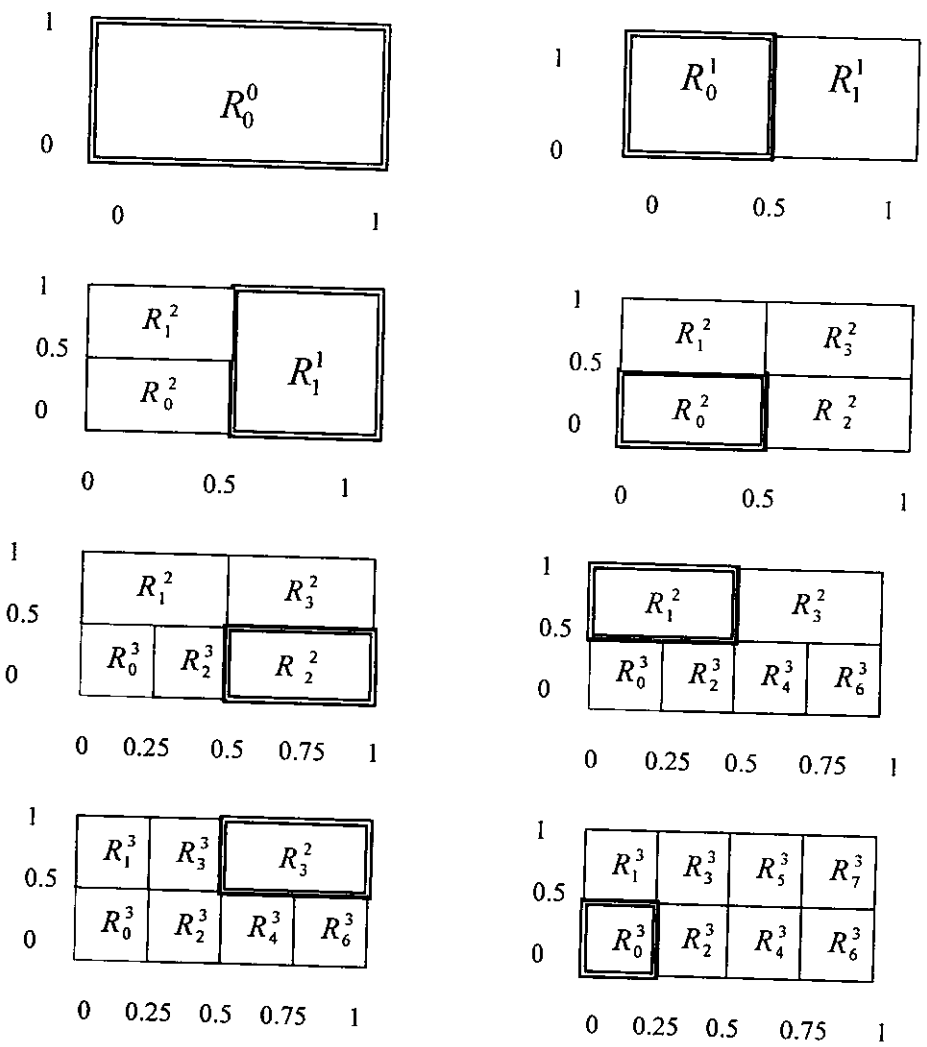


Figure A.1 : Séquence Déclatement dans un Fichier IH (d=2)

Maintenant, si n se trouve être l'adresse de la case associée a la dernière de niveau i, c-a-d, ($n=2^i$), alors après l'éclatement de la case (n), toutes les régions formant la partition de k deviennent toutes de niveau (i+1). Dans ce cas, i est incrémenté de (+1) et n est remis à 0. dans les autre cas, n est incrémenté de (+1) tandis que i reste inchangé.

Rappelons que la projection d'une clé multidimensionnelle vers sa représentation unidimensionnelle consiste à la concaténation des bits, pris alternativement des représentations binaires (en puissances négatives de 2) des composantes de cette clé.

La région R_j^i contient toutes les clés telles que les i premiers **bits** de leurs projections S, sont identiques. lors de la définition d'une région, le critère d'appartenance d'une clé k a une région R_j^i était l'égalité $\pi_i(S(k))=j$. remarquons que,selon (2), la projection π a pour effet de valider les i premiers bits de la représentation de S (k).

L'éclatement d'une région revient, donc, a distinguer deux classe de clés au sein de cette même région, et ceci sur la base du (i+1) –eme bit de la représentation binaire de leurs projections par S. l'éclatement de R_j^i fera donc apparaître :

- La région R_{2j}^{i+1} qui contient des clés telles que ce bit est a 0;
- La région R_{2j+1}^{i+1} contenant les clés telles que ce bit est a 1 ;

Selon la définition de S, il est facile de vérifier que ce bit appartient à la j-eme composante de la multidimensionnelle tel que (j=1+ (i mod d)).

Architecture Client-Serveur

L'architecture Client/Serveur est un concept logiciel qui ne date pas d'aujourd'hui : dès l'apparition des réseaux, les développeurs ont pensé à répartir l'exécution d'une application entre plusieurs (au moins 2) machines connectées par le réseau.

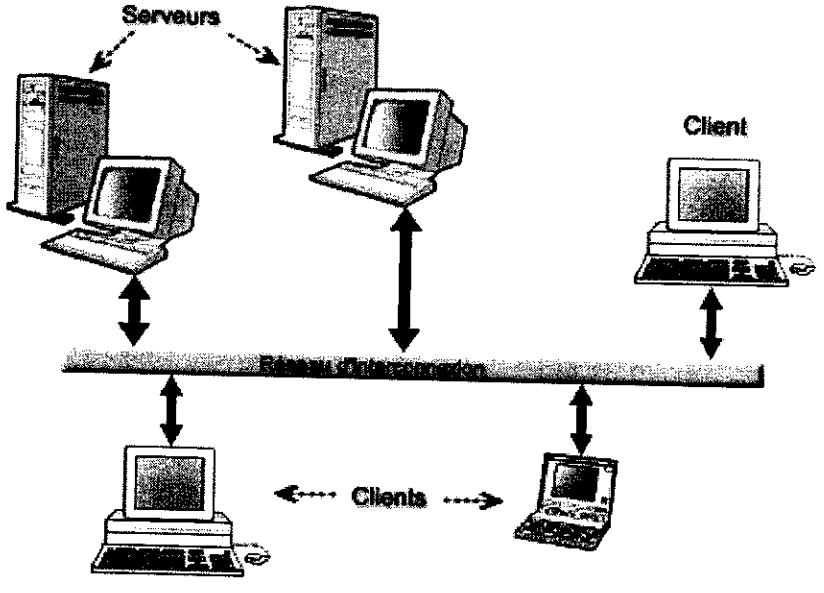


Figure B.1: Organisation d'applications Client/Serveur

L'idée de l'architecture client serveur est simple :

- ✓ le serveur est une machine qui met à disposition des services accessibles par d'autres machines sur le réseau (les clients). En général le serveur est à l'écoute des demandes des clients.

- ✓ le client, est une machine dont le principal rôle est parfois un traitement local des données, mais surtout la présentation graphique des données. En général c'est le client qui a l'initiative de la communication avec le serveur.

- ✓ le réseau transporte les données avec le protocole IP pour l'adressage et avec l'un des protocoles TCP ou UDP pour le transport. UDP est un protocole de transport plus simple que TCP car il ne fournit aucune garantie sur la fiabilité de la transmission. L'application doit prendre les mesures appropriées pour détecter et corriger les éventuelles erreurs de transmission.

1. LE SERVEUR

La nature des services fournis par le serveur est très variée : gestion d'une base de données, consultation d'une boite aux lettres (serveur *mail*), serveur de fichiers,...

La spécificité du serveur est bien sur de posséder le(s) logiciel(s) correspondants aux services, mais aussi de posséder les ressources matérielles adaptées à l'exécution de services. Par exemple un serveur de base de données est équipé d'un SGBRD (Système de Gestion de Bases de Données) et possède certainement une capacité de stockage disque en relation avec la volumétrie des données prévues.

L'optimisation des ressources matérielles du serveur est un soucis capital : si le serveur doit fournir des services à de nombreux clients nécessitant des traitements importants il devra posséder une CPU et de la RAM en conséquence. Par contre le serveur est en général déchargé des taches de présentation graphique : il n'a pas besoin d'être équipé d'une carte graphique ou d'un moniteur de hautes performances.

Sur le serveur, chaque service accessible via TCP/IP est associé à un numéro de port (port number). La machine serveur possède un mécanisme logiciel permettant de lancer un processus serveur lorsqu'un client contacte le serveur sur un numéro de port donné.

Le choix d'un service de la machine serveur est fixé par le couple : **numéro de port/protocole** qui figure dans les paquets émis par le client. Le protocole peut être :

- TCP associé à IP : TCP/IP (mode de transport dit connecté)
- UDP associé à IP : UPD/IP (mode de transport non connecté)

2. LE CLIENT

La machine cliente est surtout dimensionnée en termes de capacités graphique : carte graphique et écran. Avec l'augmentation croissante de la puissance des postes clients (PC, Macintosh, stations, ...) on trouve de plus en plus de clients qui effectuent localement (avec leurs ressources CPU et RAM) une partie du traitement des données (c'est le cas par exemple des serveurs WEB qui font exécuter des applications JAVA (on parle d'applets JAVA) sur le poste client).

3. LES NUMEROS DE PROTOCOLE ET DE PORT

La communication entre une machine cliente et une machine serveur se fait par l'intermédiaire de processus qui s'exécutent sur chacune des machines. Lorsque le processus client désire contacter un service particulier du serveur, il doit envoyer un paquet TCP ou UDP sur un port du serveur correspondant, identifié par un numéro.

Les serveurs "**écoutent**" les requêtes en observant le trafic sur un port particulier qui leur est dédié. Les clients doivent donc savoir sur quel port ils doivent envoyer leurs requêtes.

Ainsi la liaison entre un client et un serveur est complètement identifiée pour un protocole donné, par l'ensemble des nombres :

<protocole ; adresse IP client ; numéro de port émetteur ;
adresse IP serveur ; numéro port demandé >

4. COMMUNICATION ENTRE CLIENTS ET SERVEUR

Il existe deux modes de communications entre machines : avec et sans connexion.

4.1 Mode avec connexion

La caractéristique de base du mode avec connexion est qu'il est nécessaire de mettre en place une liaison dédiée entre les deux machines avant de commencer à communiquer. Concrètement, cela signifie qu'une communication dans ce mode se fait en trois étapes :

- Etablissement d'une connexion : au cours de cette phase, la machine émettrice va envoyer à la machine réceptrice un bloc d'informations pour demander la mise en place d'une connexion. La machine réceptrice va alors renvoyer un bloc spécifiant si elle est en mesure d'accepter cette liaison. Si c'est le cas, d'autres blocs seront encore échangés avant la communication proprement dite, afin de spécifier les paramètres de la connexion.
- Transfert des données : c'est la communication proprement dite.
- Fermeture de la connexion.

Un exemple de communication avec connexion est le réseau téléphonique. En ce qui concerne TCP/IP, le protocole TCP est un exemple de protocole orienté connexion.

- La liaison est sous contrôle car ses extrémités sont connues, et des paramètres de connexion ont été définis, permettant par exemple de limiter les débits à des niveaux acceptables pour les deux parties.
- Lorsqu'une connexion a été établie, l'émetteur est sûr que le destinataire est présent, ce qui assure que les données envoyées iront bien quelque part.
- L'inconvénient principal du mode avec connexion est la lourdeur. Même s'il y a très peu de données à envoyer, il est nécessaire d'établir la connexion et de définir ses paramètres.
- Comme une connexion s'établit entre deux machines, si on veut réaliser une communication multipoints (une machine communique avec plusieurs machines), il faudra ouvrir autant de connexions qu'il y a de machines à atteindre.

4.2 Mode sans connexion

C'est l'inverse du mode avec connexion : une machine désirant communiquer avec une autre peut envoyer ses données sans avoir au préalable contacté la machine réceptrice, et donc sans s'assurer que celle-ci est en fonction. La machine émettrice doit cependant connaître à l'avance les caractéristiques des machines avec lesquelles elle veut communiquer.

- C'est une procédure beaucoup plus facile à mettre en oeuvre qu'avec une connexion.
- La communication est moins bien contrôlée : une machine peut par exemple recevoir d'un seul coup un grand nombre de données provenant de plusieurs machines différentes.

- Une machine qui envoie des données n'est pas sûre que la machine réceptrice est en fonction.

Le mode sans connexion est donc approprié pour l'envoi de données courtes, ou ne nécessitant pas la présence d'un récepteur en fonction. Pour ce qui est de TCP/IP, les protocoles IP et UDP, par exemple, sont des protocoles sans connexion.

5. ARCHITECTURE INTERNE DU SERVEUR

Les programmes client/serveur sont construits au niveau de la couche application pour permettre leur exécution sur toute plate forme supportant TCP/IP. Il existe, toutefois, trois schémas possibles pour l'architecture d'un serveur :

5.1 Serveur itératif

En mode connecté (TCP), le programme serveur suit les étapes de l'algorithme suivant :

Algorithme ServeurItératif
Début
1- Création du socket Local.
2- Affectation du socket à un numéro de port spécifique.
3- Définition de la taille de la file d'attente.
4- Attente d'une connexion.
5- Lecture et traitement de la requête.
6- Envoi de la réponse.
7- Fermeture de la connexion (cas TCP).
8- Aller à 4.
Fin

Dans le cas des messages UDP, le programme du serveur ne comporte pas l'étape de connexion et le réglage de la taille de la file d'attente.

6.2 Serveur parallèle

L'algorithme précédent est légèrement modifié afin d'obtenir un parallélisme dans le traitement des requêtes :

Algorithme ServeurParallèle

Début

1- Création du socket Local.

2- Affectation du socket à un numéro de port spécifique.

3- Attente d'une connexion.

4- Lancement d'un processus ou thread asynchrone pour la lecture et traitement de la requête. Le thread se charge de transmettre la réponse et de fermer la connexion (cas d'un message TCP).

5- Aller à 3.

Fin

6.3 Serveur semi-parallèle (preforked)

Cette technique consiste à créer plusieurs threads (processus légers) au démarrage du programme serveur. Les requêtes des clients seront ensuite distribuées sur les files d'attentes associées à chaque thread. Ces threads sont souvent appelés threads de travail (Worker Threads).

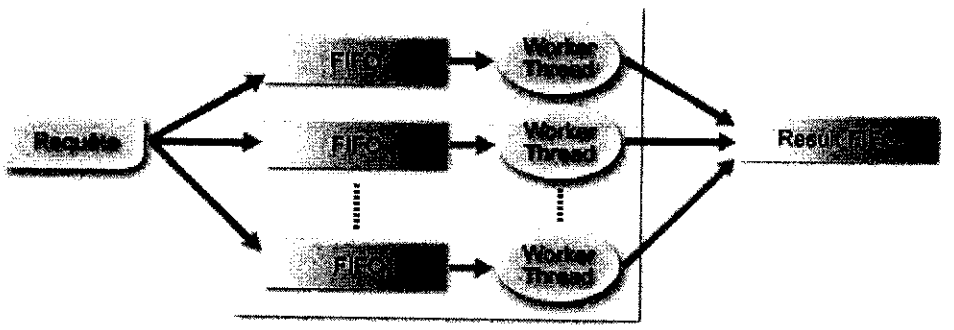


Figure B.2 : Schéma général d'un serveur semi-parallèle

Interface de Communication Par Sockets

Le concept de socket est assez difficile à appréhender. C'est Unix BSD 4.2 qui les a choisis pour accomplir les communications interprocessus (IPC). Cela veut dire qu'un socket est utilisé pour permettre aux processus de communiquer entre eux.

Formellement, un socket est un objet qu'une application communicante peut créer et utiliser comme point de communication bidirectionnel.

- Il n'est accessible au programmeur d'application que par un ensemble de primitives (création, fermeture, lecture, écriture, divers)
- L'autre application communicante peut se trouver sur la même station ou une station distante.
- L'architecture réseau sous-jacente peut être quelconque : TCP/IP, X25, Apple, IPX/SPX, etc.
- Le système d'exploitation peut-être quelconque : Unix, Windows, NT, ...

A l'origine, les sockets ont été développés pour Unix dans le contexte d'Internet TCP/IP, mais les concepteurs ont pris soin, conformément à l'esprit d'Internet, de spécifier de façon suffisamment générique pour permettre l'interopérabilité avec d'autres systèmes et architectures de réseau.

Les sockets sont un cas particulier de la notion Unix générale de fichier, qui recouvre celles de fichier ordinaire, tube, tube nommé, et donc socket. Aussi le vocabulaire est-il souvent semblable à celui utilisé pour les fichiers ordinaires : on ouvre et on ferme un socket, on lit et on écrit sur un socket : le socket est en quelque sorte un "fichier réseau".

Comme pour les fichiers, on peut utiliser le mode bloc, qu'on appelle mode datagramme, ou le mode flot d'octets (stream).

Un autre point de vue utile est de considérer le socket comme une variable structurée commune au programme de niveau application et au programme de niveau transport.

1. PORTS ET SOCKETS

La compréhension de la notion de ports et sockets est indispensable pour comprendre ce qui va suivre.

Quand un programme est lancé sur une machine, le système d'exploitation de celle-ci crée ce qu'on appelle un processus.

En simplifiant, on peut dire qu'un processus représente un programme en cours d'exécution ainsi que divers paramètres associés à cette exécution, comme l'état des registres du processeur.

Quand un processus est créé, le système d'exploitation lui attribue un numéro d'identification. Ce numéro est différent d'un système à l'autre mais aussi d'une exécution à l'autre. Le numéro de processus n'est donc pas un bon moyen d'identifier un processus de façon unique.

Or, dans la plupart des applications où deux machines doivent communiquer entre elles, il est préférable que ce soient les processus des deux machines qui échangent des données, plutôt que les machines elles-mêmes. Physiquement, cela revient au même, mais pas d'un point de vue logique. Le processus qui a créé la connexion est appelé client, et celui qui répond, serveur.

Le concept de ports et sockets offre en fait la possibilité d'identifier clairement les connexions et les processus engagés dans ces connexions, indépendamment des numéros d'identification des processus. En pratique, tout processus désirant communiquer avec un autre processus sur une autre machine, s'identifie auprès de TCP/IP par un nombre de 16 bits appelé **port**. Le port permet à TCP/IP de savoir quel processus est en train de communiquer, ce qui autorise des communications simultanées *(*multiplexage*)* de plusieurs processus.

Il existe **deux types** de ports : les ports réservés *(*Well-Known Ports*)* et les ports éphémères.

Les premiers sont des ports dont le numéro a été associé une fois pour toutes par une organisation (IANA) à un processus (serveur) particulier. Ils ont un numéro dans la gamme de 1 à 1023. Les ports les plus connus sont :

20 et 21 : FTP (File Transfer Protocol)
23 : Telnet
25 : SMTP (Simple Mail Transfer Protocol)
53 : DNS (Domain Name System)
80 : HTTP (Hyper Text Transfer Protocol)

Les seconds, dans la gamme de 1024 à 65535, peuvent être librement utilisés par les programmeurs.

2. TYPE DE SOCKETS

Le type d'une soceket définit un ensemble de propriété des communications dans lesquelles elle est impliquée.

- La fiabilité de la de la transmission
- La préservation de l'ordre des données
- La non duplication des données
- La communication en mode connecté
- La préservation des limites des messages
- L'envoi des messages "urgent"

Le type SOCK_DGRAM : Il correspond aux sockets destinées à la communication en mode non connecté pour l'envoi de datagramme de taille bornée. Les communications correspondantes ont la propriété (e). Dans le domaine Internet, le protocole sous-jacent est le protocole UDP.

Le type SOCK_STREAM : Ces sockets permettent des communications fiables en mode connecté (propriété a, b, c et d) et autorisent éventuellement les messages hors-bande (f). Le protocole sous-jacent dans le domaine Internet est TCP.

Le type SOCK_SEQPACKET : il correspond aux communications possédant les propriétés a, b, c, d et e. Il est utilisé dans le domaine XEROX NS.

Le type SOCK_ROW : Il permet l'accès aux protocoles de plus bas niveau (protocole IP). Il permet d'implanter de nouveaux protocoles.

3. DOMAINE (FAMILLE) DES SOCKETS

Il spécifie le format des adresses qui pourront être données au socket et les différents protocoles supportés pour les communication via les sockets de ce domaine.

Il existe plusieurs familles d'adresses, chacune correspondant à un protocole particulier. Les familles les plus répandues sont :

AF_UNIX	*Protocoles internes de UNIX*
AF_INET	*Protocoles Internet*
AF_NS	*Protocoles de Xerox NS*
AF_IMPLINK	*Famille spéciale pour des applications particuliers*

La structure générique

```
struct sockaddr {
  u_short  sa_family ;          /* famille d'adresse */
  char     sa_data[14] ;    /*14 octets d'adresse maximum */
} ;
```

est utilisée pour décrire les différentes primitives. Pour une application particulière, cette structure doit être remplacée par la structure correspondante du domaine de communication utilisé.

Pour le domaine UNIX (AF_UNIX), les socket sont locales au système où elles sont définies. La structure d'une adresse dans ce domaine est définie dans <sys/un.h>.

```
struct sockaddr_un {
  short    sun_family ;       /* domaine UNIX : AF_UNIX */
  char     sun_path[108] ;          /* référence UNIX */
} ;
```

Pour le domaine Internet (AF_INET), les adresses des sockets ont la structure suivante:

```
struct sockaddr_in {
  short        sin_family ; /* domaine Internet : AF_INET */
  u_short      sin_port ;          /* numéro de port */
  struct in_addr sin_addr ;        /* adresse Internet */
  char         sin_zero[8] ;       /* un champ de zéros */
} ;
```

4. LES APPELS SYSTEME

4.1. Création d'une socket

La primitive socket permet la création d'une socket. Sa forme générale est la suivante :

```
int socket (int family, int type, int protocole) ;
```

La valeur de retour est -1 en cas d'échec. le troisième paramètre est généralement 0: le système choisit alors un protocole par défaut associé au type et domaine de la socket crée.

4.2. Suppression d'une socket

Une socket est supprimée lorsque plus aucun processus ne possède de descripteur pout y accéder. Cette suppression implique la libération des entrées dans les différentes tables et des tampons alloués par le système en relation avec la socket. Cette suppression est donc le résultat d'au moins un appel à la primitive Close.

4.3. Attachement d'une socket à une adresse

Après sa création, une socket n'est accessible que par les processus qui en connaissant un descripteur. Par conséquent, sans mécanisme supplémentaire de désignation, seuls ces processus pourraient l'utiliser. La primitive bind permet de nommer une socket :

```
int bind (int sock, struct sockaddr *addr , int addrlen);
```

Le premier argument est le descripteur de socket retourné par l'appel socket. Le second argument est un pointeur sur une adresse de protocole spécifique et le troisième est la taille de cette structure d'adresse.

Pour une utilisation dans un domaine particulier, le pointeur addr pointe sur une zone dont la structure est celle d'une adresse dans ce domaine.

Le domaine UNIX : Les socket dans ce domaine ne sont destinées qu'à la communication locale. Il leur correspond donc des adresses locales qui sont des références UNIX identiques à celles des fichiers.

Le domaine Internet : L'attachement d'une adresse Internet à une socket de ce domaine nécessite la préparation d'un objet de type sock_addr_in. Cela suppose en particulier la connaissance de l'adresse de la machine locale ou le choix de la valeur INADDR_ANY et le choix d'un numéro de port.

5. COMMUNICATION EN MODE CONNECTE

Deux approches de programmation de serveurs se présentent : l'approche itérative et l'approche parallèle.

Rappelons que dans l'approche itérative, le serveur traite lui même la requête, ferme le nouveau socket puis invoque de nouveau accepte pour obtenir la demande suivante.

Dans l'approche parallèle, lorsque l'appel système accepte se termine le serveur crée un serveur fils chargé de traiter la demande (appel de fork et exec). Lorsque le fils a terminé il ferme le socket et meurt. Le serveur maître ferme quand à lui la copie du nouveau socket après avoir exécuté le fork. Il appel ensuite de nouveau accepte pour obtenir la demande suivante.

L'enchaînement des appels systèmes pour un serveur itératif et parallèle est schématisé sur la figure suivante :

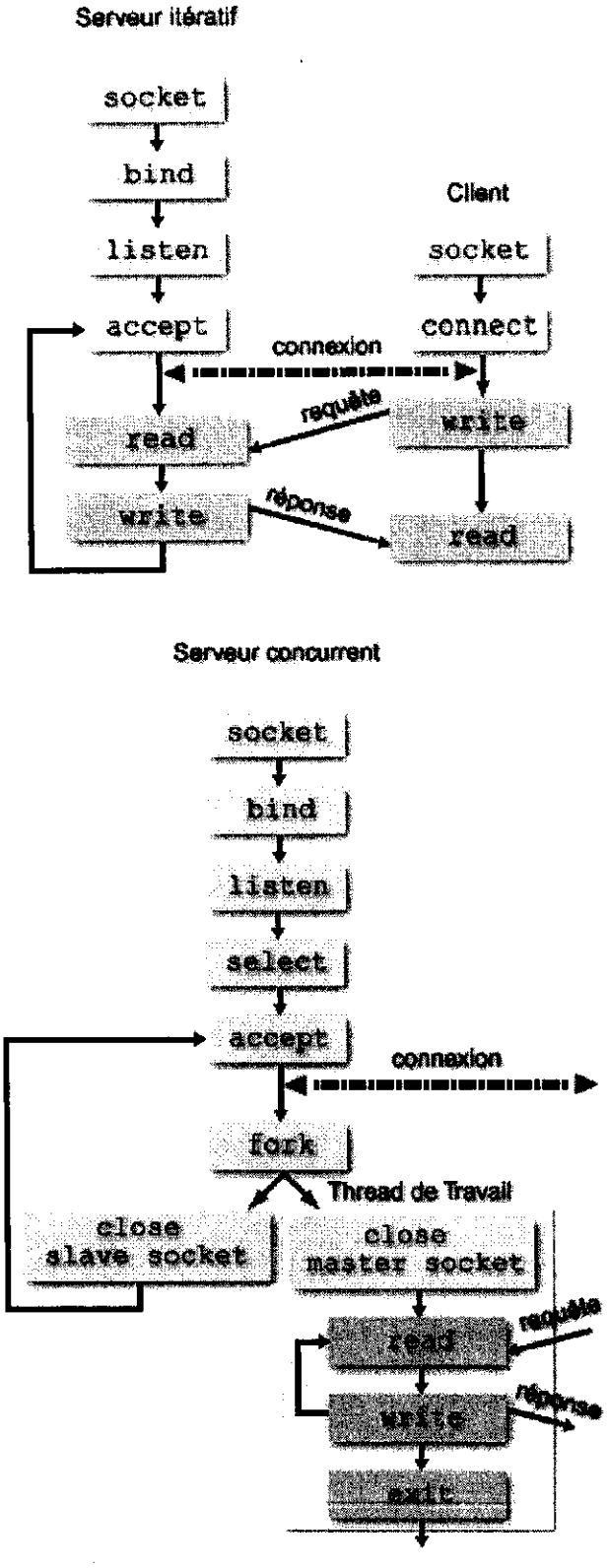


Figure C.1 : Principe d'un serveur concurrent

5.1. Attente d'une demande de connexion (listen)

```
int listen ( int sockfd, int backlog);
```

Cet appel est généralement utilisé après les appels socket et bind et juste avant les appels l'appel accept. L'argument backlog spécifie le nombre de connections à établir dans une file d'attente par le système lorsque le serveur exécute l'appel accept. Cet argument est généralement mis à 5 qui est la valeur maximale utilisée.

5.2. Connexion à une socket (connect)

Un socket est initialement créé dans l'état non connecté, ce qui signifie qu'il n'est associé à aucune destination éloignée. L'appel système connect associe de façon permanente un socket à une destination éloignée et le place dans l'état connecté.

```
int connect (int sockfd, struct sockaddr *servaddr,
        int addrlen) ;
```

Un programme d'application doit invoquer connect pour établir une connexion avant de pouvoir transférer les données via un socket de transfert fiable en mode connecté.

Les sockets utilisées avec les services de transfert en mode datagramme n'ont pas besoin d'établir une connexion avant d'être utilisés, mais procéder de la sorte interdit de transférer des données sans mentionner à chaque fois, l'adresse de destination.

5.3. Etablissement de la connexion (accept)

```
int accept (int sockfd, struct sockaddr *peer , int *addrlen);
```

Lorsqu'une requête arrive, le système enregistre l'adresse du client dans la structure *peer et la longueur de l'adresse dans *addrlen. Il crée alors un nouveau socket connecté avec la destination spécifiée par le client, et renvoi à l'appelant un descripteur de socket. Les échanges futurs avec ce client se feront donc par l'intermédiaire de ce socket.

Le socket initiale sockfd n'a donc pas de destination et reste ouvert pour accepter de futures demandes.

Tant qu'il n'y a pas de connexions le serveur se bloque sur cet appel. Lorsqu'une demande de connexion arrive, l'appel système accept se termine. Le serveur peut gérer les demandes itérativement ou simultanément :

Dans l'approche itérative, le serveur traite lui-même la requête, ferme le nouveau socket puis invoque de nouveau accept pour obtenir la demande suivante.

Dans l'approche parallèle, lorsque l'appel système accept se termine l de datagrammes e serveur crée un serveur fils chargé de traiter la demande (appel de fork et exec). Lorsque le fils a terminé, il ferme le socket et meurt. Le serveur maître ferme quant à lui la copie du nouveau socket après avoir exécuté fork. Il appel ensuite de nouveau accept pour obtenir la demande suivante.

6. LA COMMUNICATION PAR DATAGRAMMES

Dans le cas d'un protocole sans connexion, les appels système sont différents. Le client n'établit pas de connexion avec le serveur.

L'échange est réalisé par l'intermédiaire de socket du type SOCK_DGRAM. Un processus souhaitant émettre un message à destination d'un autre doit d'une part disposer d'un point de communication local et d'autre part connaître une adresse sur le système auquel appartient son interlocuteur.

Le client envoi un datagramme au serveur en utilisant l'appel système sendto. Symétriquement le serveur n'accepte pas de connexion d'un client, mais attend un datagramme d'un client avec l'appel système recvfrom. Ce dernier revoie l'adresse réseau du client, avec le datagramme, pour que le serveur puisse répondre à la bonne adresse.

L'enchaînement dans le temps des appels systèmes pour une communication en mode non connecté et pour un serveur itératif se résume par la **Figure** :

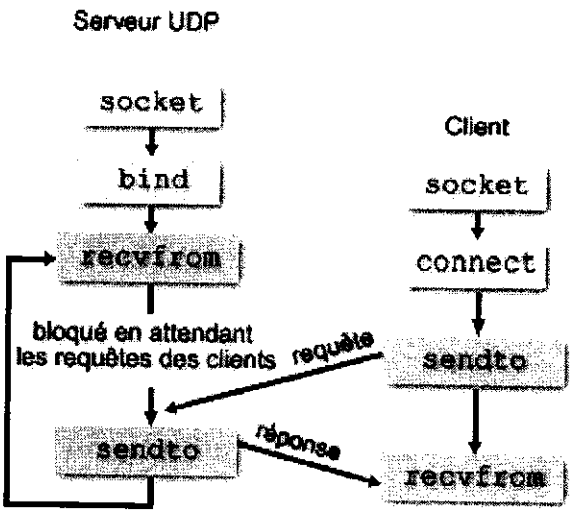


Figure C.2 : Fonctionnement d'un serveur en mode non connecté

7. ECHANGES D'INFORMATIONS SUR UN SOCKET

7.1. Emission d'informations

Une fois que le programme d'application dispose d'un socket, il peut l'utiliser afin de transférer des données. Cinq appels système sont utilisables : send, sendto, sendmsg, write et writev. send, write et writev ne sont utilisables qu'avec des sockets en mode connecté car ils ne permettent pas d'indiquer d'adresse de destination. Les différences entre ces trois appels sont mineures :

```
write ( int sockfd, char *buff, int nbytes ) ;
writev ( int sockfd, iovec *vect_E/S, int lgr_vect_E/S ) ;
int send (int sockfd, char *buff, int nbytes, int flags ) ;
```

En mode non connecté, les appels sendto et sendmsg imposent d'indiquer l'adresse de destination :

```
int sendto (int sockfd, char *buff, int nbytes, int flags,
        struct sockaddr *to, int addrlen);
```

Les quatre premiers arguments sont les mêmes que pour send, les deux derniers sont l'adresse de destination et la taille de cette adresse.

Pour les cas où on utiliserait fréquemment l'appel sendto qui nécessite beaucoup d'arguments et qui serait donc d'une utilisation trop lourde on utilise sendmsg définit comme suit :

```
int sendmsg ( int sockfd, struct struct_mesg, int flags );
```

7.2. Réception d'information

On distingue 5 appels système de réception d'information qui sont symétriques aux appels d'envoi. Pour le mode connecté on a les appelles read, readv et recv et pour le mode sans connexion on a les appelles recvfrom et recvmsg.

```
int read ( int sockfd, char *buff, int nbytes ) ;
int readv ( int sockfd, iovec *vect_E/S, int lgr_vect_E/S ) ;
int recv (int sockfd, char *buff, int nbytes, int flags ) ;
```

L'appel système recvfrom remplit les deux derniers arguments avec l'adresse de l'envoyeur.

```
int recvfrom (int sockfd, char *buff, int nbytes, int flags,
        struct sockaddr *from, int addrlen);
```

Par symétrie, il existe l'appel système recvmsg :

```
int recvmsg ( int sockfd, struct struct_mesg, int flags ) ;
```