HAL
open science

Intégration d'un client mobile à CTH

Amel Bennaceur

To cite this version:

Amel Bennaceur. Intégration d'un client mobile à CTH. Informatique ubiquitaire. 2006. hal-00850116

HAL Id: hal-00850116

https://inria.hal.science/hal-00850116

Submitted on 5 Aug 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Institut National d'Informatique

I.N.I

Mémoire de fin d'études

Pour l'obtention du diplôme d'ingénieur

d'état en informatique

Option : Systèmes Informatiques

Thème

Intégration d'un client mobile à CTH*

Réalisé par :

BENNACEUR Amel

Encadré par :

Pr. ZEGOUR (Professeur à l'INI)

En collaboration avec :

Mr. HIDOUCI (Chargé de cours à l'INI)

Année universitaire 2005 - 2006

A mes chers parents,

A mon frère Amine

Remerciements

Je tiens à remercier particulièrement Pr. DE. Zegour, professeur à l'Institut National d'Informatique pour son soutien, sa patience et sa disponibilité tout au long de cette dernière année.

Je remercie également Mr. Hidouci, chargé de cours à l'INI, pour son aide et ses conseils précieux.

Je tiens aussi à exprimer ma gratitude et ma reconnaissance à tous mes professeurs pour m'avoir transmis leur savoir durant ces cinq dernières années. Leur instruction m'apporte une base essentielle et me laisse affronter le futur avec sérénité.

Résumé

Le hachage digital compact distribué (CTH*) est une structure de données distribuée et scalable qui permet de gérer de grands volumes de données tout en maintenant des performances stables et optimales et en préservant l'ordre des enregistrements.

CTH* a été implémenté sur différents environnements distribués fixes et a permis de réaliser des temps d'accès courts et indépendants de la taille du fichier.

L'objectif de ce projet est de concevoir et de développer une infrastructure mobile prolongeant de manière naturelle la plateforme CTH* existante. La faisabilité de cette architecture sera ensuite vérifiée et les performances du prototype réalisé seront évaluées.

La plateforme ainsi obtenue intègre aussi bien des clients CTH* fixes que mobiles et offre un accès simple, rapide et efficace à une grande masse d'informations en tout lieu et à tout moment.

Le prototype CTH* peut, par la suite, être utilisé pour la mise en œuvre d'application à hautes performances ou des systèmes de gestion de bases de données à grandes capacités qui garantissent des services dynamiques, réactifs, omniprésents et sans contraintes spatio-temporelles.

Mots-clés : Hachage Digital Compact Distribué (CTH*), Structure de Données Distribuée et scalable, environnements mobiles.

Abstract

Distributed Compact Trie Hashing (CTH*) is a scalable and distributed data structure which allows the management of a large amounts of data while keeping performances steady and optimum and preserving the records order.

CTH* was implemented in different distributed and fixed environments and allowed short access time regardless of the file weight.

This project's goal is to design and develop a mobile infrastructure extending the existing CTH* platform. This architecture feasibility will be validated and the prototype performances evaluated

The CTH* platform will integrate fixed and mobile clients and offer a simple, fast and efficient access to a great deal of information anywhere and anytime.

The CTH* prototype can then, be used to implement high performance applications or very large database management system which guarantees dynamic, reactive and omnipresent services with no space or time constraints.

Key-words: Distributed Compact Trie Hashing (CTH*), Scalable Distributed Data Structure, mobile environment.

Table des matières

Introduction……1

Chapitre I : Systèmes Distribuées et Structures de Données Distribuées et Scalables……4

1 Systèmes distribués 5

1.1 Architectures matérielles 5

1.2 Architectures logiques 8

1.3 Stratégies de distribution des données 10

2 Les SDDS (Structures de Données Distribuées et Scalables) 12

2.1 Principe des SDDS 13

2.2 Caractéristiques des SDDS 14

2.3 Performances d'accès d'une SDDS 16

2.4 Classification des SDDS 16

2.5 Projets sur les SDDS 23

3 Synthèse 24

Chapitre II : Hachage Digital Compact Distribué (CTH*) ……..25

1 Hachage digital compact (Compact trie hashing) 26

1.1 Mécanisme de construction 27

1.2 Expansion de l'arbre 29

1.3 Opérations sur le fichier 31

1.4 Performances de la méthode CTH 34

2 Hachage digital compact distribué (Distributed Compact Trie Hashing) 35

2.1 Concepts 35

2.2 Expansion du fichier 37

2.3 Opérations sur le fichier 38

2.4 Mise à jour de l'image du client 44

2.5 Travaux sur CTH* 45

2.6 Plateforme existante 46
3 Synthèse 48
Chapitre III : Environnements mobiles 48
1 Schéma d'architecture des réseaux mobiles 50
2 Générations des réseaux mobiles 52
2.1 Première Génération (1G) 52
2.2 Deuxième génération (2G) 53
2.3 Evolutions de la deuxième génération (2,5G) 55
2.4 Troisième génération (3G) 56
2.5 Quatrième génération (4G) 59
3 Transmission de données dans les réseaux mobiles 61
4 Protocoles de transmission de données 64
4.1 WAP (Wireless Application Protocol) 64
4.2 i-mode 67
5 Téléphone mobile 67
5.1 Composants matériels 68
5.2 Plateforme logicielle 72
6 Limitations des environnements mobiles 76
6.1 Contraintes liées au réseau mobile 76
6.2 Contraintes liées au téléphone mobile 77
7 Synthèse 78
Chapitre IV : Conception et réalisation 78
1 Architectures 80
1.1 Architecture générale 80
1.2 Architecture du client fixe 83
1.3 Architecture du serveur 85
1.4 Architecture de coordinateur 87
1.5 Architecture du client mobile 89

1.6 Architecture du préserveur 91
2 Protocole de communication CTH* 92
2.1 Description du protocole CTH* 92
3 Implémentation et mise en œuvre 107
3.1 Environnement de développement 107
3.2 Outils techniques de mise en oeuvre 108
4 Interface de l'application 111
5 Synthèse 115

Chapitre V : Validation et tests 115

1 Environnement expérimental 117
2 Tests de validité 119
3 Tests de performances 119
Insertion 119
Requête à intervalle 123
Facteur de chargement 124
4 Synthèse 125

Conclusion 126

Bibliographie

Liste des abréviations

Liste des figures

Figure I.1	Architecture à mémoire partagée (Shared-Memory Architecture)…….	6
Figure I.2	Architecture à disques partagés (Shared-disk Architecture)……..…	6
Figure I.3	Architecture sans partage (Shared-Nothing Architecture)……..	7
Figure I.4	Architecture client/serveur…….	8
Figure I.5	Architecture Peer to Peer……	9
Figure I.6	Classification des systèmes informatiques……	9
Figure I.7	Courbes idéales de scalabilité…….	15
Figure I.8	Classification des SDDS…….	23
Figure II.1	Schéma global de CTH*…….	37
Figure II.2	Plateforme CTH* fixe…….	46
Figure III.1	Equipements et interfaces de base d'un réseau sans fil……	50
Figure III.2	Les générations de réseaux mobiles……	60
Figure III.3	Architecture d'un réseau WAP……	64
Figure III.4	Vue générale d'un téléphone mobile……	68
Figure III.5	Architecture interne d'un téléphone mobile 3G……	71
Figure III.6	Plateforme Java……	74
Figure IV.1	Architecture générale de la plateforme CTH*……	81
Figure IV.2	Architecture du client CTH* fixe……	83
Figure IV.3	Architecture du serveur CTH*……	85
Figure IV.4	Architecture du coordinateur……	87
Figure IV.5	Architecture du client mobile……	89
Figure V.1	Architecture du préserveur……..	90
Figure V.2	Architecture du réseau expérimental……	117
Figure V.3	Evolution du temps total d'insertion pour un client fixe d'image initialement vide…….....	119
Figure V.4	Temps moyen d'insertion pour un client fixe……	119

Figure V.5 Evolution du temps total d'insertion pour un client distant…… 120

Figure V.6 Temps moyen d'insertion pour un client distant……. 120

Figure V.7 Evolution du temps total d'insertion pour un client mobile…… 121

Figure V.8 Temps moyen d'insertion pour un client mobile……. 121

Figure V.9 Evolution du temps de recherche à intervalle pour un client fixe (plusieurs serveurs par machine)……..... 122

Figure V.10 Evolution du temps de recherche à intervalle pour un client fixe (un seul serveur par machine)……..... 122

Figure V.11 Evolution du temps de recherche à intervalle pour un client mobile (plusieurs serveurs par machine)…… 123

Figure V.12 Evolution du temps de recherche à intervalle pour un client mobile (un serveur par machine)……..... 123

Evolution du facteur de chargement…… 124

Introduction

Les structures de données distribuées et scalables (SDDS) sont une nouvelle méthode de répartition de données spécialement conçue pour exploiter efficacement les ressources cumulées d'un multiordinateur (c'est-à-dire un ensemble d'ordinateurs autonomes interconnectés par un réseau à haut débit). Elles adoptent une gestion dynamique des fichiers et permettent la prise en compte de grands volumes d'informations en maintenant des performances stables et optimales.

Les données d'un fichier SDDS sont réparties sur plusieurs sites de stockage appelés serveurs. Le fichier s'étend par l'éclatement des serveurs surchargés et se rétrécit par leur fusion. Cette gestion décentralisée des fichiers permet d'éviter les goulots d'étranglement et offre des performances de stockage et de temps de réponse meilleures que celles des structures de données centralisées traditionnelles.

Les serveurs SDDS sont accessibles à partir de stations indépendantes et autonomes appelées clients. Chaque client a sa propre image du fichier. Cette image n'étant pas mise à jour de manière synchrone, elle peut devenir erronée ; le client fait alors des erreurs d'adressage en envoyant une requête à un serveur non approprié. Ce serveur est capable de rediriger cette requête vers le serveur adéquat et d'envoyer au client un message d'ajustement (IAM) pour lui permettre de corriger son image.

Il existe plusieurs variantes de SDDS qui s'inspirent généralement des structures de données existantes. Nous trouvons ainsi des SDDS basées sur le principe de hachage et d'autres basées sur la techniques des arbres.

CTH* (Distributed Compact Trie Hashing) est un nouvelle structure de données distribuée et scalable qui adapte une méthode de hachage dynamique, appelée hachage

digital, aux environnements distribuées. Le hachage digital est l'une des méthodes d'accès les plus rapides pour les fichiers mono-clés ordonnées et dynamiques, il utilise une fonction de hachage lexicographique, ce qui lui permet de bénéficier de la simplicité et de l'efficacité des méthodes de hachage tout en préservant l'ordre des enregistrements.

CTH* a été implémenté en environnements distribués fixes sous Windows et sous Linux et offre une bibliothèque de fonctions puissantes qui permettent de gérer des fichiers (création, ouverture, fermeture et suppression) et de manipuler leurs données (insertion, recherche, mise à jour, suppression et requête à intervalle). Dès lors, il est intéressant d'y intégrer une infrastructure mobile prolongeant de façon naturelle la plateforme fixe.

Le projet SD²M (Structure de Données Distribué Mobile) vise l'adaptation de CTH* aux environnements mobiles et l'extension de ses fonctions à travers des terminaux mobiles, notamment par l'intégration de clients et de serveurs mobiles et par l'adoption d'une architecture pair-à-pair (P2P : Peer to Peer).

Notre travail s'inscrit dans le cadre de ce projet. Il s'agit d'instaurer un accès simple et rapide à une grande masse d'informations en tout lieu et à tout moment à travers un client CTH* mobile.

Toutefois, la diversité des plateformes de développement, l'hétérogénéité des interfaces de communication, les limitations des débits des réseaux mobiles ainsi que les contraintes en ressources des terminaux mobiles soulèvent des défis importants en matière de conception et nécessitent une réflexion sur les techniques de mise en œuvre. En effet, les difficultés inhérentes aux environnements mobiles recèlent des verrous architecturaux sous-jacents dont nous devons tenir compte lors de la réalisation de notre prototype mobile.

Enfin, la prise en compte de la plateforme CTH* fixe est indispensable pou réutiliser autant que possible le code existant et de ne pas réimplémenter une plateforme opérationnelle, efficace et éprouvée.

L'objectif de ce travail est de spécifier, concevoir et développer un client mobile s'intégrant à la plateforme CTH* existante. Une fois le prototype réalisé, nous l'analysons, testons sa faisabilité et évaluons ses performances.

Ce document est structuré en cinq chapitres complémentaires organisés en suite logique.

Le premier chapitre introduit les systèmes distribués et présente les structures de données distribuées et scalables (SDDS).

Le second se penche sur une SDDS particulière, le hachage digital compact distribué (CTH*) ; il définit ses concepts fondamentaux, présente les différents algorithmes et décrit la plateforme CTH* fixe.

Le troisième chapitre dresse un panorama des environnements mobile et étudie leurs caractéristiques afin de connaître les possibilités offertes par de tels systèmes ainsi que leurs limitations.

Le quatrième chapitre détaille la conception et l'implémentation de chaque composant de la plateforme CTH* intégrant les clients fixes et mobiles.

Le cinquième chapitre est consacré aux tests de faisabilité et aux mesures de performances. Ces expérimentations permettent de valider les choix conceptuels et architecturaux effectués et d'évaluer le prototype réalisé.

Enfin, une conclusion résume les aspects et apports du travail réalisé et évoque les perspectives qu'il ouvre.

Chapitre I

Systèmes distribués

et

Structures de Données Distribuées Salables

(SDDS)

L'évolution des architectures d'ordinateurs et des réseaux de communication ainsi que l'augmentation des masses d'informations traitées ont favorisé l'apparition de nouveaux systèmes basés sur la distribution de données sur plusieurs sites.

Ce chapitre présente les principes fondamentaux des systèmes distribués, puis décrit les concepts, les caractéristiques et les variantes d'une technique de répartition de données particulière, les SDDS (Structure de Données Distribuée et Scalable).

1 Systèmes distribués

Un système distribué est un ensemble d'ordinateurs indépendants qui apparaissent aux utilisateurs comme un seul système cohérent [TAN94].

Les tâches de communication inter-machines, les mécanismes nécessaires pour accéder aux ressources et les techniques de réplication et de récupération en cas de panne sont masqués à l'utilisateur. Ce dernier manipule donc les ressources de manière simple et facile car il considère l'ensemble du système comme une seule machine virtuelle.

Dans cette section nous décrivons les différentes architectures matérielles et les divers modèles logiques des systèmes distribués. Nous abordons également les méthodes de répartition des données dans ces systèmes.

1.1 Architectures matérielles

1.1.1 Architecture à mémoire partagée : Shared-Memory

Dans une architecture à mémoire partagée, les disques et les mémoires centrales sont physiquement partagés par tous les processeurs du système.

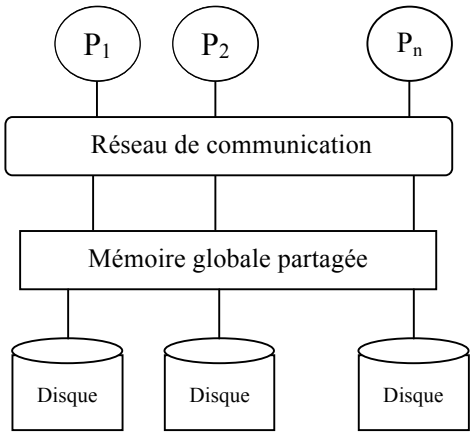


Figure I.1 Architecture à mémoire partagée (Shared-Memory Architecture)

Cette architecture présente l'intérêt de fournir un modèle de programmation simple dans la mesure où chaque processeur accède de manière transparente à la mémoire commune. L'équilibre de charge entre les processeurs est facile à réaliser et l'accès aux données est possible même en cas de panne d'un ou de plusieurs processeurs.

Cependant, les accès conflictuels peuvent créer un goulot d'étranglement et dégrader considérablement les performances du système. De plus, l'ajout d'un nouveau nœud entraîne des modifications importantes au niveau du matériel. Ce modèle est n'est donc pas extensible et assez coûteux.

1.1.2 Architecture à disques partagés : Shared-Disk

Dans l'approche à partage de disques, chaque site possède une mémoire privée (cache) et peut accéder à n'importe quel disque à travers un réseau d'interconnexion

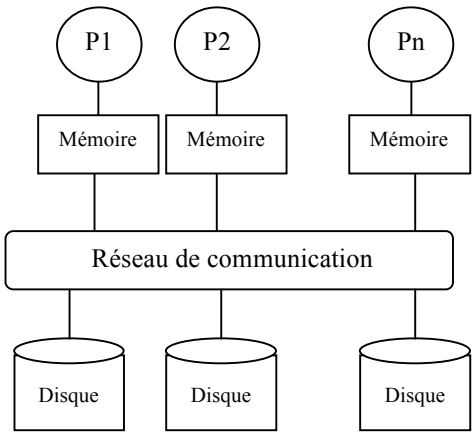


Figure I.2 Architecture à disques partagés (Shared-disk Architecture)

Chaque processeur peut ainsi copier des données du disque partagé dans son propre cache disque, ce qui augmente le niveau de parallélisme mais rend difficile le maintien de la cohérence des caches.

1.1.3 Architecture sans partage : Shared-Nothing

Dans cette approche, chaque processeur dispose de sa propre mémoire locale et de son propre espace disque.

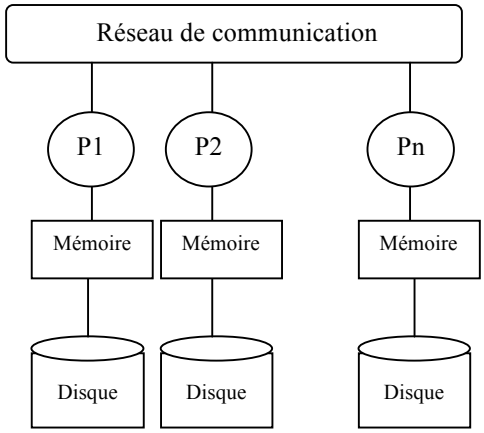


Figure I.3 Architecture sans partage (Shared-Nothing Architecture)

Ce système constitué d'un ensemble d'ordinateurs autonomes interconnectés par un réseau généralement haut débit est facilement extensible et offre un rapport prix/performances inégalé. La puissance théorique cumulée des ressources en calcul et en mémoire est impossible à atteindre pour tout autre système.

Toutefois, ces systèmes sont difficiles à administrer et à programmer et la panne d'un nœud entraîne l'indisponibilité des données contenues dans son disque.

Le mode de communication le plus approprié est l'envoi de messages. Il existe trois types de messages : les messages unicast (point à point) destinés à une seule machine, multicast (multipoint) adressés à un ensemble de nœuds appartenant au même groupe ou broadcast (diffusion) diffusés vers toute les machines du sous réseau.

1.2 Architectures logiques

1.2.1 Architecture Client/Serveur

C'est un modèle d'architecture où les programmes sont répartis entre processus clients et processus serveurs. Le client envoie une requête au serveur sachant son adresse et son port d'écoute spécifique au service demandé. Le serveur reçoit la demande, la traite puis envoie la réponse au client.

Dans un environnement purement client/serveur, les clients ne peuvent voir que le serveur. Il n'y a donc pas de dialogue ente les clients. Ces derniers peuvent être supprimés ou rajoutés sans perturber le fonctionnement du réseau ni modifier son architecture. Le réseau est donc évolutif.

Le serveur gère les ressources communes à tous les utilisateurs évitant ainsi les problèmes de cohérence et améliore la sécurité. Le serveur est donc au centre du réseau, ce qui facilité certes son administration mais rend le système très vulnérable aux pannes.

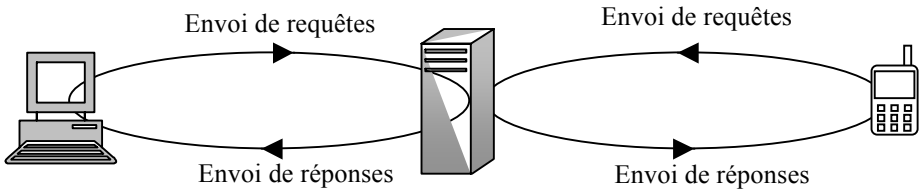


Figure I.4 Architecture client/Serveur

Il existe d'autres types d'architectures client/serveur appelées architectures multiniveaux. Dans ces systèmes, le serveur recevant la requête peut faire appel à plusieurs serveurs intermédiaires (middleware) pour la traiter.

1.2.2 Architecture Pair à Pair

Les applications pair à pair (Peer-to-Peer, P2P) travaillent dans un mode totalement décentralisé. Chaque nœud correspond directement avec un petit nombre d'utilisateurs qui eux-mêmes correspondent avec d'autres, ce qui crée un graphe de connexion (figure I.5)

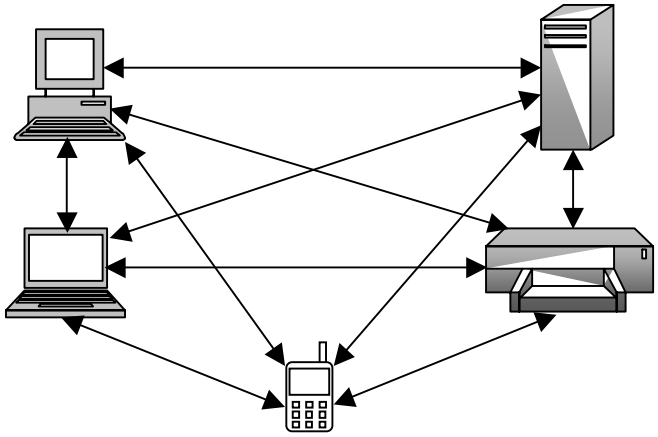


Figure I.5 Architecture Peer to Peer

La répartition des traitements et des ressources équilibre la charge entre les entités, accélère les échanges et augmente la résistance aux pannes mais rend difficiles l'administration et la sécurisation du système. De plus, lorsque le nombre de connexions augmente, le réseau est vite saturé à cause des messages envoyés dans tous les sens. En outre, une limitation importante de ce modèle est liée au fait que les entités du réseau n'ont pas toujours une adresse IP (Internet Protocol) fixe.

Outre les architectures natives (pures) complètement distribuées, il existe des architectures hybrides qui utilisent un serveur jouant le rôle d'un annuaire commun où les utilisateurs s'enregistrent et déclarent les ressources partagées.

La figure I.6 résume les différentes architectures citées ci-dessus.

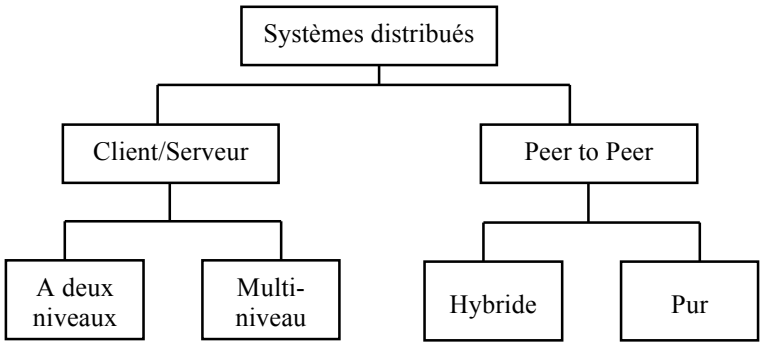


Figure I.6 Classification des systèmes informatiques

1.3 Stratégies de distribution des données

L'une des composantes essentielles d'un système distribué est le gestionnaire de fichiers distribués (DFS : Distributed File System). Il se charge de répartir les données et les programmes sur les différents sites (nœuds) du réseau offrant ainsi une plus grande disponibilité des fichiers.

La première approche de distribution utilisée dans un système de fichiers distribués est celle qui consiste à stocker chaque fichier sur un site unique. Plusieurs produits commercialisés ont adopté cette technique, en particulier NFS (Network File System) et AFS (Andrew File System).

Cette approche engendre des limitations bien connues concernant d'une part, la taille du fichier qui est limitée par l'espace disque disponible sur un seul site, et d'autre part, la scalabilité[1] des performances d'accès et la vulnérabilité aux pannes. Pour dépasser ces limitations, de nouvelles méthodes, reposant sur l'idée de distribuer le fichier sur plusieurs serveurs, sont apparues. Il en existe deux catégories : statiques et dynamiques.

1.3.1 Méthodes statiques

Les méthodes statiques, appelées aussi "schéma à fragmentation statique", se basent sur l'idée simple de la distribution des enregistrements d'un fichier sur plusieurs sites suivant un ou plusieurs critères. Dans ce qui suit, nous citons quelques-unes de ces méthodes.

Distribution par Round-Robin

La stratégie de distribution selon le "Donneur de Cartes" (Round-Robin Partitioning) est le schéma de partitionnement le plus simple ; il consiste à distribuer les enregistrements d'un fichier par rotation à travers les différents sites. Si N est le nombre de nœuds, le $i^{éme}$ article (ou unité de fragmentation) est affecté au nœud de numéro i mod N.

Elle permet ainsi de partitionner le fichier en fragments de même taille et garantit un équilibre de charge optimum. L'entrelacement des données sur les différents serveurs, augmente le parallélisme des requêtes de parcours séquentiel mais il est mal adapté aux requêtes sélectives.

[1] scalability en anglais, peut se traduire par extensibilité

Distribution par hachage

La stratégie de distribution par hachage (Hashed Partitioning) distribue les enregistrements du fichier en utilisant une fonction de hachage sur un ensemble d'attributs appelé attributs de partitionnement. La fonction de hachage retourne le numéro du serveur auquel l'article sera affecté. Cette méthode est très efficace pour le traitement des requêtes dont le prédicat implique les attributs de partitionnement.

Distribution par intervalles

La stratégie de distribution par intervalles (Range Partitioning) distribue les articles en fonction de la valeur d'un ou de plusieurs attributs. Elle consiste à diviser l'espace des valeurs des attributs sélectionnés en intervalles puis à affecter chaque intervalle à un nœud. Par la suite, chaque enregistrement t est assigné au nœud n si les valeurs des attributs spécifiés de t appartiennent à l'intervalle de n. Cette stratégie facilite le traitement des requêtes par intervalle et les opérations impliquant les attributs de partitionnement. Cependant, elle ne garantit pas que les fragments issus du partitionnement auront la même taille. De ce fait, ces derniers peuvent être disproportionnés. Dans le cas extrême, il peut exister des nœuds vides et d'autres débordés. Il n'y a donc pas d'équilibre de charge entre les nœuds.

1.3.2 Méthodes dynamiques

Suivant les schémas de partitionnement statique, un fichier ne peut être réparti que sur les sites alloués initialement. Ainsi, lorsque la distribution est établie, le nombre de sites et les critères de partitionnement sont figés durant toute la durée de vie du fichier, même si son exploitation devient non optimale par suite des mises à jour. Cette situation nécessite une redistribution du fichier puis une suppression de l'ancienne copie [DIE01].

C'est pour surmonter ces difficultés que les schémas de partitionnement dynamique ont été conçus.

DLH (Distributed Linear Hashing)

La méthode DLH [SEV90] est le premier schéma qui a été proposé pour le partitionnement dynamique. Elle s'appuie sur les architectures de multiprocesseurs faiblement couplées et à partage de mémoire. Un fichier DLH réside en mémoire centrale et ses paramètres sont stockés dans la mémoire cache locale de chaque processeur du système.

Les caches sont rafraîchis de manière sélective lorsqu'il y a une erreur d'adressage mais aussi à travers des mises à jour périodiques durant l'évolution du fichier [DIE01].

Les SDDS

Les Structures de Données Distribuées Scalables (SDDS: Scalable, Distributed Data Structures) [LIT93] ont été proposées pour gérer la distribution des données d'un fichier dans une architecture sans partage. Elles se basent sur le modèle client/serveur.

Contrairement aux systèmes de gestion de fichiers distribués traditionnels, un fichier SDDS ne comporte pas de répertoire central. De plus, il peut s'étendre d'un seul site de stockage à n'importe quel nombre de sites.

Les SDDS permettent de gérer un grand volume de données en maintenant des performances stables optimales. Plusieurs SDDS ont été développées et ont pu montrer l'efficacité de cette méthode. Ces résultats ont poussé plusieurs centres de recherche à travailler sur ce thème : Microsoft, IBM, l'université de Berkeley, etc.

2 Les SDDS (Structures de Données Distribuées et Scalables)

Le volume et la complexité des données gérées par la majorité des applications sont en perpétuelle croissance : objets complexes, graphiques, images, vidéo, sons, documents hypermédia. Les systèmes de gestion de fichiers distribués traditionnels ne peuvent plus faire face à cet accroissement continu des masses d'informations à gérer car la taille d'un fichier est limitée par l'espace disque disponible sur un seul site et le calcul d'adresse, unique et centralisé, entraîne une vulnérabilité aux pannes et peut provoquer des goulots d'étranglement. De plus, les changements de la structure du fichier exigent des mises à jour synchrones et dégrade les performances d'accès. Pour remédier à ces inconvénients, il faut utiliser des structures de données capables de tirer le meilleur parti des nouvelles architectures de machines.

Les structures de données distribuées et scalables (SDDS) ont été spécialement définies pour gérer des fichiers distribués sur un multiordinateur. Un multiordinateur est basé sur une architecture sans partage, c'est-à-dire qu'il est constitué d'un ensemble d'ordinateurs hétérogène reliés par un réseau haut débit.

Les SDDS offrent ainsi au prix d'une simple configuration matérielle, des temps d'accès beaucoup plus courts que ceux des données stockées sur les disques. De plus, elles gèrent

plus efficacement les accès parallèles et peuvent traiter des quantités de données potentiellement illimitées. Ces caractéristiques assurent aux SDDS des performances de traitement supérieures à tout autre structure de données traditionnelle.

2.1 Principe des SDDS

Un fichier SDDS réside initialement sur un seul site de stockage ; il est ensuite étendu dynamiquement à n'importe quel nombre de sites appelés nœuds ou serveurs. Contrairement aux schémas classiques où l'ajout d'un nouveau nœud nécessite une réorganisation totale du fichier, les SDDS permettent une extension du nombre de serveurs avec un coût minimal. Cette extension se fait par des éclatements de sites débordés suite à des insertions d'enregistrements. Et les enregistrements sont répartis sur les serveurs suivant la valeur de leur clé et l'état du fichier.

Les serveurs sont accessibles à partir de stations autonomes appelées clients. L'un des objectifs majeurs des SDDS est de minimiser le nombre de messages nécessaires pour accéder à un enregistrement du fichier. Typiquement, l'accès à un enregistrement nécessite deux messages : un message pour envoyer la requête et un autre pour envoyer l'enregistrement demandé.

Chaque client a sa propre image de la structure du fichier. Elle lui permet de calculer l'adresse du serveur où est supposé se trouver un enregistrement de clé c. Cette image n'est pas globalement exacte, car les mises à jour de la structure du fichier ne sont pas envoyées aux clients d'une manière synchrone. Cette stratégie permet de minimiser les coûts de communication entre clients et serveurs car une mise à jour synchrone des images deviendrait très coûteuse lorsque le nombre de clients est élevé. Un client peut alors faire une erreur d'adressage par suite d'une image devenue inexacte. Le serveur cible lui envoie alors un message correctif. Ce message est appelé message d'ajustement de l'image du client (IAM : Image Adjustment Message). Le client corrige son image pour ne pas refaire la même erreur. Parallèlement à la mise à jour de l'image du client, le serveur redirige la requête vers le serveur adéquat. Ce dernier l'exécute et s'il n'y a pas d'erreur d'adressage il envoie la réponse au client.

2.2 Caractéristiques des SDDS

Les SDDS ont plusieurs propriétés, certaines sont propres aux systèmes distribués et d'autres sont spécifiques aux SDDS. Cette section décrit les trois principales caractéristiques des SDDS.

2.2.1 La scalabilité

La scalabilité d'un système est sa capacité à supporter une montée de charge. Elle reflète son aptitude à utiliser des ressources additionnelles de manière optimale. Elle peut être résumée par deux paramètres, le facteur de rapidité et le facteur d'échelle.

Facteur de rapidité

Le facteur de rapidité ou facteur d'accélération (speed-up) mesure la diminution du temps de réponse en augmentant le nombre de nœuds pour l'exécution d'une même requête sur un même volume de données. Il est égal au rapport du temps de traitement d'une requête par un seul serveur et du temps de traitement de la même requête par n serveurs.

$$\text{Facteur de rapidité} = \frac{\text{Temps de traitement par un serveur}}{\text{Temps de traitement par n serveurs}}$$

L'idéal serait d'avoir un speed-up linéaire c'est-à-dire que lorsque les capacités du système sont multipliées par x ($x > 0$), le temps de réponse diminue d'un facteur de x [MOU04], figure I.7 a)

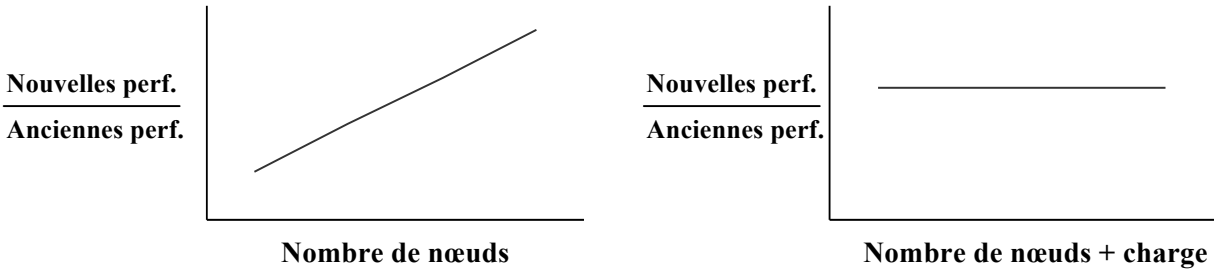
Le facteur de rapidité ne tient pas compte de la configuration du système, mais le facteur d'échelle décrit ci-dessous tient compte à la fois de la taille du problème et de la configuration matérielle.

Facteur d'échelle

Le facteur d'échelle ou facteur d'accroissement (scale-up) mesure la conservation du temps de réponse en augmentant de façon proportionnelle la charge et le nombre de processeurs, par rapport à la configuration initiale.

$$\text{Facteur d'échelle} = \frac{\text{Temps mis par un serveur pour traiter une requête}}{\text{Temps mis par n serveurs pour traiter n requêtes}}$$

L'idéal est de conserver le même temps pour traiter une charge par un processeur que n fois la charge par n processeurs. La courbe serait donc horizontale, figure I.7 b). En réalité, le temps de réponse augmente quand la charge augmente malgré l'ajout de processeurs [MOU04].



a) Facteur de rapidité b) Facteur d'échelle

Figure I.7 Courbes idéales de scalabilité

2.2.2 Distribution

Contrairement aux structures de données classiques où les données sont sauvegardées sur un seul site, les SDDS permettent la distribution du fichier sur plusieurs serveurs, autorisant ainsi la gestion de grands volumes de données. Cette stratégie permet de bénéficier de la puissance de stockage cumulée du multiordinateur et d'augmenter le niveau de parallélisme.

2.2.3 Disponibilité

Un système de stockage performant doit garantir la disponibilité des données même après une panne.

L'architecture modulaire des SDDS leur permet d'intégrer cette fonctionnalité et d'assurer l'accès aux données en cas d'échec d'un ou de plusieurs nœuds.

La solution la plus triviale de disponibilité consiste à sauvegarder les données régulièrement sur des supports magnétiques ou optiques. En cas de perte, les données sont récupérées à partir des sauvegardes, avec le risque de perte des mises à jour ayant lieu après la dernière sauvegarde. D'autres stratégies plus performantes peuvent être utilisées. Ces techniques se basent sur le principe de réplication, de fragmentation ou de groupage.

2.3 Performances d'accès d'une SDDS

Lors de la conception d'une SDDS, il est primordial de considérer les paramètres suivants :

- nombre moyen de messages par requête de recherche, insertion, mise à jour ou suppression. Il est à préciser que cette mesure est indépendante des paramètres du réseau. Il est parfois difficile de calculer le nombre moyen de messages ; nous avons donc recours au nombre de messages dans les conditions les plus défavorables ;
- convergence de vue des clients ;
- taux de remplissage : $\alpha = \frac{m}{b * N}$ où m désigne le nombre d'articles du fichier, b la capacité de la case et N le nombre de serveurs ;
- coût d'éclatement.

2.4 Classification des SDDS

Nous pouvons classer les SDDS en deux grandes familles selon la stratégie de distribution de données adoptée. La première famille est basée sur la distribution par hachage et la seconde sur les arbres.

2.4.1 SDDS basées sur le hachage

Plusieurs structures de données distribuées utilisent le hachage comme méthode d'accès à des données distribuées, notamment LH* [LIT94], EH* [HIL97] ou DDH [DEV93].

LH*

La structure de données distribuée et scalable LH* est due à Litwin, Neimat et Schneider. Elle utilise une version distribuée du hachage linéaire (LH : Linear Hashing) [LIT80].

LH est un algorithme de hachage extensible par lequel on étend l'espace d'adressage primaire d'un fichier pour éviter l'accumulation de débordements et la détérioration progressive des performances d'accès. Un fichier LH est une collection de cases, contenant chacune un ensemble d'enregistrements. Le fichier s'étend et se rétrécit de manière dynamique à travers des éclatements et des fusions de cases. L'adresse logique d'une clé C

est déterminée par le calcul de la valeur de la fonction de hachage hi(c) = c modulo N* 2i où N désigne le nombre de cases initiales, i le niveau du fichier et n un pointeur sur la prochaine case à éclater.

L'adaptation de LH aux environnements distribués consiste à affecter à chaque serveur une adresse logique, une case et un niveau. LH* n'impose pas de structure interne pour la case.

LH se base sur le principe qu'il existe un seul compteur de niveau i et un seul pointeur vers la prochaine case à éclater n. Or les contraintes des environnements distribués font que ce principe ne peut pas être respecté. D'où la nécessité d'utiliser un mécanisme qui permet de maintenir la cohérence de ces paramètres. La solution la plus répandue consiste à utiliser un site central appelé coordinateur.

Un fichier LH* peut être étendu à un nombre quelconque de clients et de serveurs en maintenant les propriétés suivantes [LIT94] :

- la taille du fichier est pratiquement illimitée, avec un facteur de chargement entre 65-95%, en fonction des paramètres du fichier.
- une insertion nécessite généralement un message, trois au pire des cas.
- une recherche utilise habituellement deux messages, quatre au plus.
- une opération parallèle sur un fichier de M cases utilise au plus 2M+1 messages, et en moyenne $\log_2 M$ messages.

Variantes de LH*

La famille LH* compte plusieurs variantes. Les principales variantes sont brièvement décrites dans ce qui suit.

LH^*_{LH}

Cette variante de LH* a été proposée par Karlesson, Litwin et Risch [KAR96]. Elle spécifie la structure interne d'une case LH* en proposant une organisation LH pour cette case. Le schéma LH^*_{LH} utilise alors deux niveaux d'indexation. Le premier est un niveau d'indexation réseau géré par l'algorithme LH* qui permet au client de trouver la case LH* pour exécuter une requête. Le deuxième niveau est un niveau d'indexation interne géré par l'algorithme LH.

Les variantes à haute disponibilité

Ces variantes permettent de préserver la disponibilité des données en cas de panne d'un ou de n serveurs. Plus la taille du fichier augmente plus n doit être grand afin de compenser la baisse de fiabilité due à l'étalement du fichier. Car, Un fichier distribué sur N nœuds a une probabilité de disponibilité égale à p^N, où p désigne la probabilité de disponibilité d'un nœud. Notons que, quand N croit, p^N converge vers zéro. Pour pallier cet inconvénient, le concept de haute disponibilité a été appliqué à LH*.

Plusieurs techniques de récupération sont actuellement connues.

1) Technique des miroirs (mirroring)

Le principe des miroirs ou haute disponibilité par réplication, découle du principe de sauvegarde. Il consiste à sauver les données sur d'autres nœuds de stockage et à définir une stratégie de propagation des mises à jour (synchrone ou asynchrone) vers les sites miroirs [MOU04]. L'atout majeur est que les sites miroirs sont fonctionnels et interrogeables. Par contre, le principal inconvénient des solutions à base de réplication de données est le coût de stockage qui est facteur du nombre de répliques.

Un exemple de l'utilisation des miroirs dans LH* est LH^*_M [LIT96a].

2) Technique de fragmentation (Stripping)

Ce schéma nécessite moins d'espace de stockage que le schéma à miroirs mais les opérations de mises à jour demandent plus de temps et utilisent plus de messages. La segmentation peut être effectuée au niveau bit, bloc ou attribut.

LH^*_S [LIT97a] est une amélioration de LH* se basant sur la fragmentation.

Un fichier LH^*_S est constitué de k+1 fichiers segments LH*, $S_1 \ldots S_{k+1}$. Le fichier segment de parité S_{k+1} permet la récupération des données perdues appartenant à un fichier segment de données.

Le schéma LH^*_S est dit 1-disponible car il tolère l'échec d'un seul nœud. Il présente également un aspect de sécurité renforcé. En effet, pour une segmentation au niveau bit, chaque fichier segment n'a qu'un bit de k bits consécutifs d'un enregistrement. En désignant par x le nombre de bits manquants, un intrus doit énumérer les 2^x possibilités pour reconstituer l'article.

La sécurité de LH^*_S est au prix d'une complexité dans la restitution et la recherche des enregistrements, puisque les manipulations du client nécessitent une coopération de tous les

segments. Ainsi, une variante de LH^*_S, se propose de faire la fragmentation au niveau attributs

3) Technique de groupage (grouping)

Plusieurs structures de données distribuées et scalables utilisant le groupage et se basant sur LH* ont été proposées. Dans ce qui suit, nous décrivons brièvement trois d'entre elles.

*LH*g*

Dans LH*g [LIT97b] chaque groupe contient un serveur de parité, calculé par XOR des serveurs de données, et permettant au groupe de survivre à l'échec d'un serveur.

Le contenu du fichier de parité permet de récupérer n'importe quel enregistrement de clé C appartenant à un groupe si, et seulement si, les enregistrements appartenant au même groupe logique que C sont disponibles.

LH^*_{RS}

LH^*_{RS} est une autre version LH* pouvant supporter la haute disponibilité par la mise en œuvre d'un calcul de parité qui repose sur les codes de Reed Solomon[2] [MOU04]. Les données de parité sont stockées au niveau de sites spécifiques appelés serveurs de parité. Un fichier LH^*_{RS} est ainsi subdivisé en groupes de parité, où un groupe est formé de m cases de données et de k cases de parité. Le schéma LH^*_{RS} est alors dit k-disponible, car il peut survivre à l'échec d'au maximum k serveurs

LH^*_{SA}

Quand la taille du fichier croit, la 1-disponibilité ou la k-disponibilté pour un entier k statique est insuffisante pour éviter la décroissance de la fiabilité. Il faut donc accroître k de manière dynamique. Le résultat est une nouvelle SDDS à disponibilité incrémentale (scalable availability schemes) : LH^*_{SA} [LIT98]. LH^*_{SA} est une technique élaborée basée sur le principe de groupage d'article (Record grouping). Chaque enregistrement de donnée est un membre de k groupes ayant comme intersection l'ensemble {c} et chaque groupe est 1-disponible. La valeur de k croît progressivement avec l'accroissement du fichier. Donc

[2] Il s'agit d'un algorithme mathématique de détection, correction et compensation des bits erronés.

quel que soit k, le fichier LH^*_{SA} est k-disponible. Le coût de stockage est fonction de la taille du fichier.

Une variante multidimensionnelle : IH*

IH* est la première structure de données distribuée et scalable basée sur le hachage linéaire multidimensionnel. IH* constitue à la fois une extension du hachage par interpolation de Burkhard [BUR83] aux environnements distribués et une introduction de la notion d'ordre et de l'aspect multidimensionnel à la SDDS LH* [LIT96b].

Hachage extensible distribué

Hilford, Bastani et Cukic ont proposé une SDDS: EH* [HIL97]. EH* distribue les articles en utilisant l'algorithme de hachage extensible (EH : Extensible Hashing) proposé par Fagin [FAG79].

DDH

Devine [DEV93] soulève quatre inconvénients du schéma de distribution de données LH* et propose la SDDS : DDH (Dynamic Distributed Hashing) se basant sur l'algorithme de hachage dynamique proposé par Larson [LAR78]. Les quatre inconvénients soulevés sont :

- LH* a été proposé tel qu'un nœud héberge une seule case, ceci oblige à concevoir des cases volumineuses, ce qui dégrade les performances.
- Le deuxième inconvénient soulevé est le problème de la perte d'un message important, tel que l'ordre d'éclatement. D'où, la nécessité de communication entre les serveurs et le coordinateur qui se traduit par une relation de dépendance réduisant l'autonomie locale des serveurs;
- Dans LH* un message arrive à sa destination au bout de deux renvois maximum. D'après Devine cela n'est pas vrai si la vitesse du client est inférieure à la vitesse des éclatements
- Enfin, le schéma d'éclatement LH* est jugé inéquitable car les éclatements se font dans un ordre déterministe. Ainsi, les cases surchargées doivent supporter un excès de charge pendant que des cases non surchargées éclatent.

Devine propose aussi une méthode de calcul des adresses beaucoup moins coûteuse que le modulo en utilisant le ET binaire (&1: bitwise AND) et l'opérateur de décalage de bits à

gauche (<<2 : bitwise left shift). Ainsi, hi(c) = c mod 2i est remplacée par hi(c) = c & ((1<< i) - 1).

2.4.2 SDDS basées sur les arbres

Le schéma de distribution RP* (acronyme de Range Partitioning) a été proposé par Litwin, Neimat et Schneider [LIT94]. Un fichier RP* est une collection de cases, contenant des enregistrements identifiés par un champ clé. Chaque case est organisée en B-arbre et un intervalle]λ,Λ] lui est associé. Cet intervalle détermine l'ensemble des clés des enregistrements appartenant à la case. Initialement, un fichier RP* ne contient qu'une seule case : la case 0. Cette dernière est d'intervalle] – ∞, + ∞ [. Le fichier évolue à travers les éclatements de cases. En effet, chaque case RP* a une capacité maximale de b enregistrements. Une case surchargée éclate. L'éclatement consiste à transférer approximativement la moitié du contenu de la case surchargée vers une nouvelle case RP*. La définition de la clé de division est stratégique pour l'équilibre de charge.

Plusieurs variantes du schéma RP* ont été développées, notamment RP^*_N, RP^*_C et RP^*_S ainsi que des variantes gérant l'accès multi-clé [LIT95] et la haute disponibilité [DIE01].

Dans la suite de cette section, nous présentons brièvement chacune de ces variantes.

RP^*_N

Le schéma RP^*_N utilise la communication multipoint pour toute manipulation du fichier. Ainsi, lorsqu'un client émet une requête, il l'envoie en multicast à tous les serveurs. Seul le serveur concerné répond en envoyant un message unicast, les autres ignorent la requête

.

RP^*_C

RP^*_C ajoute à RP^*_N une image au niveau de chaque client. Les messages unicast sont alors utilisés par le client pour envoyer des requêtes et par le serveur pour transmettre les réponses. Le multicast n'est employé que pour les redirections.

RP^*_S

RP^*_S ajoute à RP^*_C un index réparti au niveau de serveurs spécifiques appelés serveurs d'index. Cet index représente une image de la structure globale de la répartition du fichier et est transparent aux clients. Les objectifs de ce schéma sont d'éliminer l'usage du multicast pour envoyer les requêtes simples et d'accélérer la convergence de l'image du client pour réduire le nombre d'erreurs d'adressage. Cette technique fournit donc un fichier distribué, ordonné et supportant uniquement des messages unicast.

Une variante à haute disponibilité : RP^*_{RS}

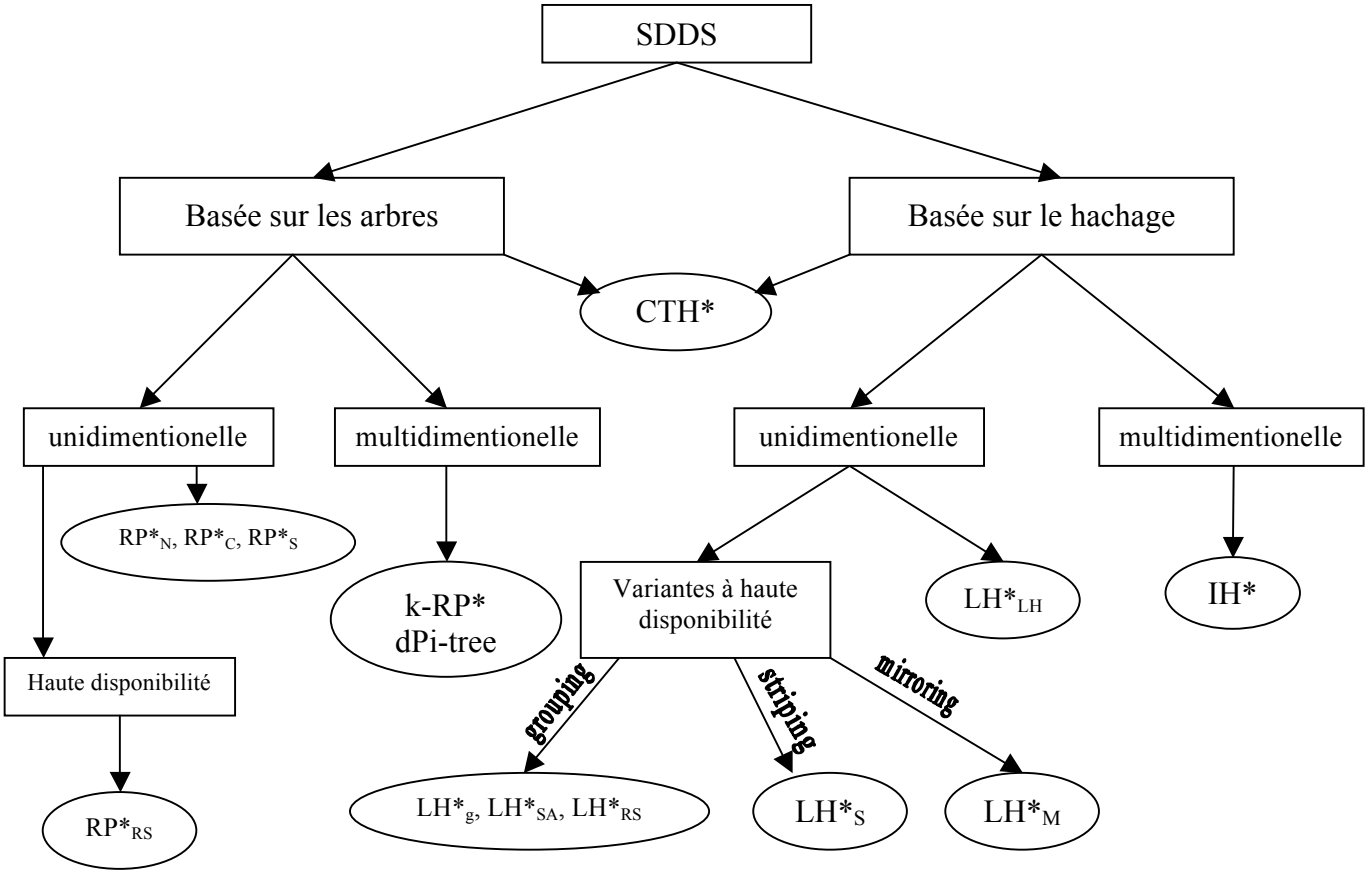
Le modèle RP* fonctionne bien mais à condition qu'il n'y ait pas de défaillances matérielles ou logicielles. RP^*_{RS} [DIE01] se propose de pallier cet inconvénient en proposant une SDDS inspirée du schéma de haute disponibilité définie pour les SDDS LH* (LH^*_{RS}).

Une variante multi-attributs : k-RP*

Cette SDDS offre la possibilité de traiter des requêtes selon plusieurs critères. C'est une adaptation des B-arbres multidimensionnels aux environnements distribués. Elle est similaire à RP* sauf qu'elle permet de traiter des requêtes contenant plusieurs clés. Pour cela un index en arbre m-aire distribué est utilisé du côté du serveur et un arbre m-aire partiel est utilisé du côté du client. Cela augmente la complexité de la mise en œuvre et diminue l'efficacité.

Il existe d'autres SDDS basées sur les arbres mais qui sont moins connues que les précédentes. Citons par exemple DRT (Distributed Random Tree) [KRO94], BDST (Balanced and Distributed Search Trees) [KRO94] et LDT (Logarithmic Distributed Search Tree) [PAS99].

Pour finir, nous résumons ces différentes catégories dans le schéma I.8.



Classification des SDDS

2.5 Projets sur les SDDS

Les performances qu'offrent les SDDS ont suscité l'intérêt de plusieurs chercheurs et ont conduit à la réalisation de plusieurs prototypes applicatifs. Les principaux travaux effectués dans ce domaine sont décrits dans cette section.

2.5.1 SDDS-2000

SDDS-2000 est un gestionnaire de Structures de Données Distribuées et Scalables (SDDS) conçu pour des multiordinateurs fonctionnant sous Windows 2000/NT/XP.

Ce projet est développé dans les laboratoires de CERIA Dauphine (France) en collaboration avec plusieurs laboratoires et universités ; notamment, l'université de Berkeley (Etats-Unis), le laboratoire EDSLAB de l'université de Linköping (Suède) et Microsoft Research.

Plusieurs projets visant le couplage de ce prototype avec des systèmes de gestion de base de données (SGBD) ont été développés ; nous trouvons ainsi AMOS-SDDS qui applique le principe des SDDS à AMOS II, un SGBD orienté objets à haute performance [NDI01] ;

et SD-SQL qui est un Système de Gestion de Bases de Données Distribuées et Scalables (SD-DBS : Scalable and Distributed Data Base System) et a été appliqué à Microsoft SQL Server [SAH04].

La dernière version de ce prototype SDDS est SDDS-2005[3].

2.5.2 Act21

Ce projet a été développé à l'Institut National d'Informatique (Algérie) [HID03]. Il visait la création d'un nouveau modèle pour des bases de données réparties ou parallèles à base d'objets actifs (acteurs) et utilisait les structures de données distribuées et scalables comme méthode de distribution. Ainsi, une nouvelle méthode SDDS a été mise au point, CTH* (Distributed Compact Trie Hashing).

3 Synthèse

Les structures de données distribuées et scalables sont utilisées pour répartir les données sur différents sites d'un système parallèle distribué tout en garantissant des performances stables et des temps d'accès indépendants du volume de données stockées. Cette solution est très attractive grâce aux exploits réalisés dans le domaine des communications.

Il existe deux familles SDDS ; la première est basée sur les le hachage et se distingue par son efficacité et sa simplicité d'implémentation ; la seconde est basée sur les arbres et permet de préserver l'ordre mais au prix d'une augmentation des temps d'accès et d'un accroissement de la complexité de réalisation.

A mi-chemin entre ces deux classes, une nouvelle structure de données distribuée et scalable appelée CTH* (Distributed Compact Trie Hashing)[4], que nous allons présenter dans le prochain chapitre, permet de bénéficier des avantages de chacune sans pour autant hériter de leurs inconvénients.

[3] Le prototype est disponible sur le site Web du CERIA http://ceria.dauphine.fr/

[4] * sert à souligner le caractère distribué de la méthode.

Chapitre II

Hachage Digital Compact Distribué

(CTH)*

Le hachage digital compact et distribué (CTH*) est une structure de données distribuée et scalable basée sur une technique de hachage dynamique, le hachage digital. Elle a été proposée par le professeur Zegour en 2001.

Cette méthode allie la simplicité et l'efficacité des SDDS basées sur le hachage à la qualité de préservation de l'ordre des SDDS basées sur les arbres.

Ce chapitre décrit la méthode de hachage digital compact puis s'intéresse à sa version distribuée, à savoir CTH*.

1 Hachage digital compact (Compact trie hashing)

Le hachage digital, proposé par le professeur W. Litwin en 1981 [LIT81], est l'une des méthodes les plus rapides pour l'accès aux fichiers monoclé, ordonnés et dynamiques. La technique utilise une fonction de hachage variable représentée par un arbre digital qui croît et décroît en fonction des insertions et suppressions. Lorsque l'arbre réside en mémoire pendant l'exploitation du fichier, un accès au plus est nécessaire pour retrouver un article. C'est souvent le cas des fichiers de taille moyenne présents sur des stations autonomes. Mais lorsque la taille du fichier est très grande, il devient intéressant d'éclater l'arbre en plusieurs sous-arbres organisés en une structure hiérarchique appelée hachage digital multiniveaux (Multilevel Trie Hashing) [ZEG94].

Il existe plusieurs manières de représenter l'arbre digital [ZEG94]. CTH permet de réduire de moitié l'espace mémoire nécessaire à la représentation de l'arbre digital en utilisant la méthode standard. Pour un environnement distribué cette option est d'autant plus intéressante qu'elle permet de diminuer la taille des parties de l'arbre transférées d'un site à l'autre. L'idée est de représenter les liens de manière implicite au détriment d'algorithmes de maintenance légèrement plus complexes que ceux de la représentation standard. La méthode CTH consomme seulement trois octets par case du fichier, ce qui permet d'adresser des millions d'articles avec un espace mémoire dérisoire.

1.1 Mécanisme de construction

Nous illustrons la technique de construction de l'arbre par un exemple :

1. A l'initialisation, l'arbre n'est constitué que d'une seule case, la case 0.

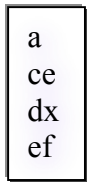
   Considérons une capacité de case de quatre enregistrements.

   Par souci de simplification, les enregistrements sont assimilés à leurs clés.

   → L'arbre : | 0

2. Les clés : a, ce, dx, ef sont insérées dans la case 0.

   → L'arbre : | 0



Case 0

3. Lors de l'insertion de la clé h, la capacité de la case 0 est dépassée. Un éclatement est alors effectué.

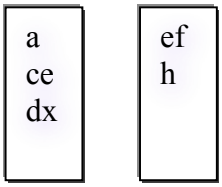
   Clé maximale ='||…'

   a ce dx ef h

   ↑

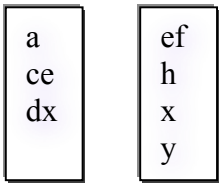
   Séquence de division : d

   → L'arbre : d 0| 1



Case 0 Case 1

4. Insertion de x et y dans la case 1 sans collision.



Case 0 Case 1

5. Lors de l'insertion de la clé kh, la capacité de la case 1 est dépassée. Un éclatement est alors effectué

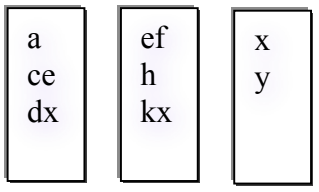
   Clé maximale ='||…'

   ef h kx x y

   ↑

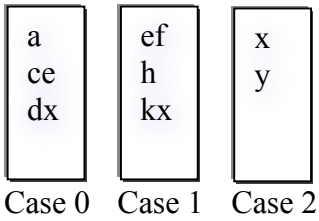
   Séquence de division : k

   → L'arbre : d 0 k 1| 2



Case 0 Case 1 Case 2

6. Insertion de la clé fe sans collision.



7. Lors de l'insertion de la clé hx, la capacité de la case 1 est dépassée. Un éclatement est effectué

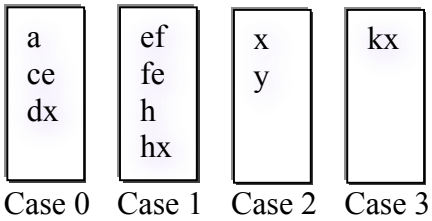
   Clé maximale ='k|…'

   ef fe h hx kx

   ↑

   Séquence de division : h

   → L'arbre : d 0 h 1 k 3| 2



8. Lors de l'insertion de la clé hy, la capacité de la case 1 est dépassée. Un éclatement est effectué

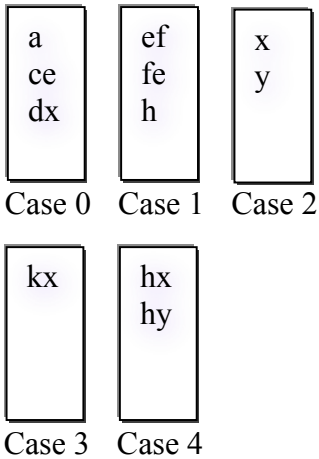
   Clé maximale ='h|…'

   ef fe h hx hy

   ↑

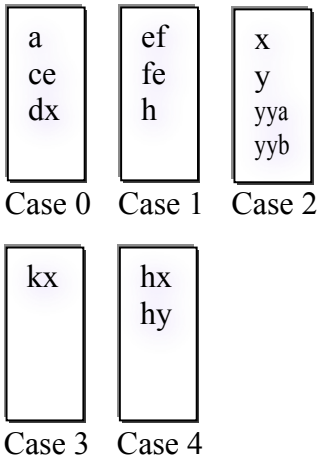
   Séquence de division : h_

   → L'arbre : d 0 h _ 1 4 k 3| 2



9. Insertion des clés yya et yyb

   → L'arbre : d 0 h _ 1 4 k 3| 2



10. Lors de l'insertion de la clé yyc, la capacité de la case 2 est dépassée. Un éclatement est effectué

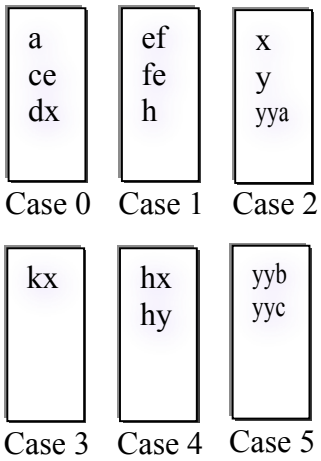
Clé maximale ='||…'

x y yya yyb yyc

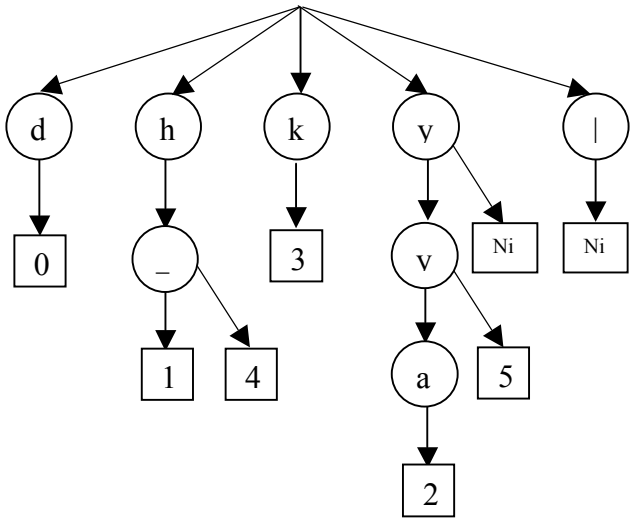
↑

Séquence de division : yya

→ L'arbre : d 0 h _ 1 4 k 3 y y a 2 5 Nil | Nil



Représentation graphique de l'arbre : d 0 h _ 1 4 k 3 y y a 2 5 Nil | Nil



1.2 Expansion de l'arbre

Il existe plusieurs variantes du hachage digital compact [SAA04]. Les améliorations de la variante de base visent à accroître le facteur de chargement en retardant le remplacement des nœuds Nil. Les algorithmes de recherche, insertion et éclatement sont modifiés en conséquence.

La variante la plus performante dans un environnement distribué, et qui sera utilisée par la suite, maximise le taux de remplissage en redirigeant les clés destinées aux nœuds Nil vers la

case non Nil la plus proche à droite. De plus, elle évite les accès inutiles en ignorant les nœuds Nil lors des éclatements.

Algorithme

1. Soit m la case à éclater.

   Former la séquence ordonnée des clés de m augmentée de la clé qui a provoqué la collision. Soit C' la clé du milieu et C" la dernière.

2. Déterminer la plus petite séquence (séquence de division) Seq des digits de C' qui permet de distinguer C' de C". Soit $C'_1C'_2.....C'_K$ cette séquence.

3. Déterminer les I premiers digits de cette séquence qui existent déjà dans l'arbre.

   Soit Ind_d : l'indice dans l'arbre du premier digit de la clé maximale C_m de m.

   Ind_m : l'indice de la case surchargée m

   Si I <> 0

   Si C_m préfixe de Seq

   Ind_d ← Ind_m

   Sinon

   Ind_d ←l'indice du premier digit de *Seq* différent dans C_m

   Fsi

   Fsi

4. Soit M la prochaine case à allouer au fichier.

   Distribuer les (b+1) clés entre l'ancienne et la nouvelle case. Les clés inférieures ou égales à Seq restent dans m, les autres migrent vers M.

   Si k – I = 1

   Remplacer m par M

   Insérer à la position Ind_d la séquence C'_km (Deux nœuds sont ajoutés)

   Fsi

   Si k- I > 1

   Insérer à la position Ind_d C'_{i+1} C'_{i+2} .....C'_k m M Nil_1 Nil_2 ..... Nil_{k-I-2}

   Dans ce cas 2(k-1) nœuds sont ajoutés

   Fsi

1.3 Opérations sur le fichier

1.3.1 Recherche

La recherche est une opération de base. Elle est le point de départ de toute opération sur le fichier. Elle consiste à trouver à partir de l'arbre digital la case supposée contenir une clé donnée. La recherche ne retourne jamais Nil[1].

Soit le prédicat et les fonctions suivantes :

feuille(nœud)	revoie vrai si nœud est une feuille.
case(nœud)	renvoie le numéro logique de la case pointée par nœud ; dans ce cas, nœud est une feuille.
digit(nœud)	renvoie le digit du nœud dans le cas où nœud est un nœud interne.
suivant(nœud)	renvoie un pointeur vers l'élément qui suit nœud dans l'arbre, Nil si nœud est le dernier élément de l'arbre.

Algorithme

```
niveau = 0
Tant que (non feuille (nœud) ou  (C>CM) ou (case (nœud)=Nil) ) Faire
    Si ( non feuille (nœud) ) Alors
            CM_niveau = digit (noeud)
            niveau = niveau + 1
    Sinon
            CM_niveau = '|'
            niveau = niveau – 1
  Fsi
  nœud = suivant (noeud)
Ftq
Retourner (case (noeud))
```

1.3.2 Insertion

L'insertion commence par l'exécution de l'algorithme de recherche afin de trouver la case m devant contenir la clé C. Si la clé n'existe pas déjà, elle est insérée dans m. Si la capacité de la case est dépassée, un éclatement est effectué.

Soit

C la clé à rechercher.

Clés(case) une fonction renvoyant l'ensemble des clés d'une case.

Capacité(case) désigne la capacité c'est-à-dire le nombre de clés que peut contenir une case.

Algorithme

```
m = Recherche(C)
Si C ∉ Clés(m)
        Insérer C dans m
        Si Capacité(m) dépassée
                Exécuter l'algorithme d'éclatement
        Fsi
Fsi
```

1.3.3 Suppression

Tout comme l'insertion, la suppression commence par l'application de l'algorithme de recherche afin de trouver la case m contenant la clé à supprimer. Suite à la suppression, le nombre de clés de m peut devenir inférieur à la moitié de sa capacité (b). Si, de plus, la somme du nombre des clés de m et de ceux de sa case sœur m_S est inférieure à la capacité b, une fusion est effectuée. La fusion consiste à regrouper les clés des deux cases sœurs dans l'une d'entre elles et à libérer l'autre. Aussi, l'arbre digital est mis à jour. Cette mise à jour consiste à éliminer un nœud interne et un nœud externe. Enfin, une fusion peut induire d'autres fusions en cascade.

Algorithme

m = Recherche(C)

Si C ∈ Clés(m)

Supprimer C de m

Soit m_S la case sœur de m

Si (nb_clés(m) < $\frac{\text{Capacité(m)}}{2}$) && (Capacité(m) + Capacité(m_S) < b)

Exécuter l'algorithme de fusion

Fsi

Fsi

L'algorithme de fusion est le suivant :

Algorithme

Soit ind_m l'indice dans l'arbre de la case m. Quatre cas sont à considérer

Cas 1 : …d m m_S c'est-à-dire non feuille(ind_m – 1) et feuille(ind_m + 1)

Après fusion, nous aurons …m… ou … m_S… selon que m_S ou m est supprimée.

Cas 2 : …d m_S m c'est-à-dire non feuille(ind_m – 2) et feuille(ind_m – 1)

Dans ce cas, nous aurons … m_S… ou …m … selon que m ou m_S est supprimée.

Cas 3 : …d m d_1 m_S … c'est-à-dire non feuille(ind_m – 1) et non feuille(ind_m + 1) et feuille(ind_m + 2) avec ind_m + 2 < taille_arbre

Dans ce cas, nous aurons …d_1 m_S…

Cas 4 : …d_1 m_S d m … c'est-à-dire non feuille(ind_m – 3) et feuille(ind_m – 2) et non feuille(ind_m – 1)

Dans ce cas, nous aurons …d m…

1.3.4 Requête à intervalle

Etant donné que l'arbre digital préserve l'ordre, une requête sur l'intervalle [C_{min}, C_{max}] consiste à rechercher la case m dont l'intervalle contient C_{min} puis à parcourir séquentiellement les cases de l'arbre jusqu'à rencontrer celle dont la clé maximale est supérieure à C_{max}.

Soient les fonctions suivantes :

suivant(m) renvoie le numéro de la case non Nil qui suit la case m dans l'arbre digital

suivant(CM) renvoie la clé maximale de la case non Nil qui suit la case dont la clé maximale est CM.

Algorithme

```
(m,CM) = Recherche(Cmin)
clé_min = Cmin
Tant que Cmax > CM
        Traiter les clés de m appartenant à l'intervalle [clé_min, CM]
        clé_min = CM
        m = suivant(m)
        CM = suivant(CM)
Ftq
```

1.4 Performances de la méthode CTH

Nous exposons dans ce paragraphe les résultats des tests mesurant les performances de CTH. [ZEG94] [SAA04]

1.4.1 Facteur de chargement

Le facteur de chargement est de l'ordre de 60-80% contre 50-70% pour l'algorithme de hachage digital de base.

1.4.2 Performances d'accès

En supposant que l'arbre digital réside en mémoire centrale, nous avons les résultats suivants:

- Puisque les clés des nœuds Nil sont redirigées vers la case la plus proche, toute opération de recherche d'une clé provoque un accès disque. Cette opération est en O(N), N étant le nombre de nœud de l'arbre digital.
- Une opération d'insertion provoque un accès disque s'il n'y a pas de collision et trois s'il y en a.
- Une opération de suppression provoque un accès disque s'il n'y a pas de fusion et trois s'il y en a.
- La requête à intervalle est facilitée car les enregistrements sont ordonnés dans les cases qui sont elle-même ordonnées dans l'arbre.

1.4.3 Espace mémoire

- Chaque case est représentée dans l'arbre digital par trois octets. Ce qui correspond à la moitié de l'espace nécessaire pour une représentation standard.
- Le taux des nœuds Nil est un peu plus élevé car ils sont ignorés lors des éclatements.

2 Hachage digital compact distribué (Distributed Compact Trie Hashing)

2.1 Concepts

CTH* est l'adaptation du hachage digital compact (CTH) aux environnements distribués. L'extension de CTH à CTH* consiste à distribuer le fichier conformément au principe des SDDS. Il se base sur le modèle client /serveur et comporte généralement trois composantes : le serveur, le client et le coordinateur.

Serveur

Le serveur peut gérer plusieurs fichiers SDDS. Pour chaque fichier, il possède une case et un arbre digital partiel qui garde trace de tous les éclatements effectués sur ce serveur.

A l'initialisation, un fichier est constitué d'une seule case appartenant au serveur 0. L'arbre digital associé est "|0". Le fichier CTH* évolue par la suite à travers des

éclatements et des fusions. Un éclatement provoque l'allocation d'une nouvelle case et par conséquent un nouveau serveur, et la distribution des clés entre l'ancien et le nouveau serveur. L'allocation du serveur se fait de façon statique ou dynamique. De plus, un serveur physique peut être associé à plusieurs serveurs logiques. Le nombre de serveurs est conceptuellement infini. Une fusion provoque le regroupement des clés de deux serveurs et la libération de l'un d'entre eux.

Client

A tout moment, un client peut rejoindre le système avec un arbre vide (|0). Cet arbre est corrigé au moyen de messages d'ajustement d'image (IAM) envoyés par les serveurs suite à des erreurs d'adressage.

Le rôle d'un client est de permettre aux applications locales de formuler des requêtes sur les fichiers SDDS. Le client peut supporter plusieurs images pour pouvoir accéder à différents fichiers SDDS. La localisation des serveurs reste invisible aux applications. En effet, les applications ignorent totalement la structure de la répartition du fichier sur le multiordinateur et font aussi abstraction des mécanismes liés au protocole de communication pour l'envoi et la réception des messages entre les clients et les serveurs.

Coordinateur

Le coordinateur est un composant important pour la plupart des variantes du hachage digital compact distribué[2]. Il maintient une table d'allocation dynamique appelée PAT (Physical Allocation Table). A chaque fois qu'un serveur rejoint le système, il s'enregistre auprès du coordinateur en lui envoyant ses paramètres (espace disponible, adresse, etc…). Après réception de ces paramètres, le coordinateur ajoute le serveur à sa PAT. Cette table lui permet de gérer les débordements et d'attribuer des adresses logiques uniques aux différents serveurs.

Le coordinateur remplit également une autre fonction, celle de serveur de nom. Le serveur de nom est un élément fondamental dans une SDDS. Il gère l'attribution des noms de fichier et assure leur unicité. Pour ce faire, il enregistre le nom et les propriétés de chaque fichier dans une table appelée FAT (File Allocation Table). Le coordinateur consulte cette table à chaque opération sur un fichier.

[2] Certaine variantes, souvent à base de multicast n'utilise pas de coordinateur.

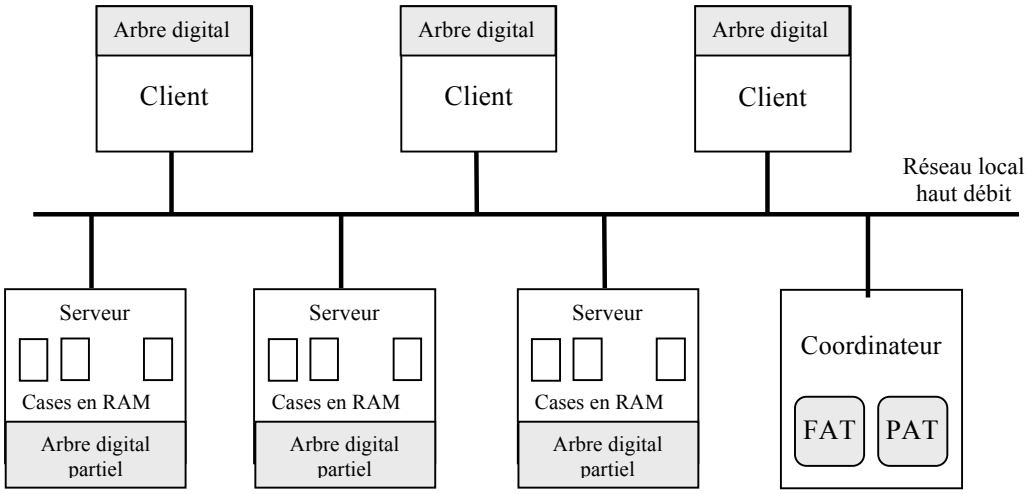


Figure II.1 Schéma global de CTH*

2.2 Expansion du fichier

L'une des propriétés les plus importantes d'un fichier SDDS est de pouvoir s'étendre dynamiquement sur un multiordinateur. Cette extension est possible grâce aux éclatements de cases. Le débordement nécessite l'allocation d'un serveur pour contenir la nouvelle case.

Le choix d'un nouveau site peut être à la charge du site en débordement ou d'un coordinateur. Dans le premier cas, le serveur peut avoir recours au multicast ou bien maintenir une liste de tous les serveurs et la consulter afin de trouver un serveur disponible. Cette dernière solution est facile à implémenter mais pose le problème de la cohérence des tables des serveurs.

Dans le second cas, le coordinateur est le seul à disposer d'une table d'allocation de serveurs PAT. Lorsqu'une collision se produit, le serveur en débordement contacte le coordinateur afin que ce dernier lui trouve un serveur capable de contenir une nouvelle case du fichier. Le coordinateur utilise sa PAT pour rechercher un serveur libre. S'il en trouve un, le coordinateur lui affecte un numéro logique, récupère ses paramètres à partir de la PAT et les renvoie au serveur débordé[3]. Si aucun serveur n'est disponible, le coordinateur envoie une réponse au serveur débordé en lui indiquant qu'aucun serveur libre n'a été trouvé. A la réception d'une réponse affirmative, le serveur exécute l'algorithme d'éclatement de CTH afin de mettre à jour son arbre et sa case et d'initialiser le nouveau serveur. Si le serveur débordé reçoit une réponse négative, la clé n'est pas insérée et le client est prévenu de l'échec de l'opération.

[3] Un serveur débordé est un serveur dont la capacité a été dépassée.

Algorithme

Soit m le serveur en débordement et CM = C_1 C_2 … C_k sa clé maximale

m contacte le coordinateur afin de trouver un serveur m' capable de contenir la nouvelle case

Si m' existe

Appliquer l'algorithme de CTH pour modifier l'arbre de m

Partager les clés entre m et m'

Initialiser l'arbre de m' avec les digit de CM suivis de m' et de (k – 1) nœuds Nil.

L'arbre de m' est : C_1 C_2 … C_k m' Nil_1 Nil_2 …Nil_{k-1}

Sinon

L'éclatement ne peut être effectué car aucun serveur n'est disponible

Fsi

2.3 Opérations sur le fichier

2.3.1 Recherche

Pour rechercher une clé C, le client utilise son arbre digital afin de trouver le serveur m contenant C. Une fois que le serveur m reçoit la requête, il vérifie si C est inférieure à sa clé maximale. Si tel est le cas, il traite la requête et renvoie au client l'enregistrement désiré ou lui indique son inexistence. Si C est supérieure à la clé maximale de m, le serveur déduit que le client a commis une erreur d'adressage et lui envoie un message d'ajustement (IAM). En même temps que la mise à jour de l'image du client, le serveur utilise son arbre digital pour rechercher l'adresse du serveur m' contenant la clé de m puis redirige la requête vers m'. A la réception de cette requête, le serveur m' refait les mêmes actions que m.

Algorithme

- **Client**

  Appliquer l'algorithme de recherche CTH. m = Recherche(C)

  Envoyer une requête de recherche de la clé C vers m

- **Serveur**

  Si C < CM

  Traiter la requête de recherche

  Envoyer la réponse au client

  Sinon

  Appliquer l'algorithme de recherche CTH. m' = Recherche(C)
  Rediriger la requête vers m'
  //

  Envoyer un IAM au client

  Fsi

2.3.2 Insertion

Pour insérer une clé C, le client commence par rechercher l'adresse du serveur où la clé doit être insérée. A la réception de la requête, le serveur vérifie si C appartient à son intervalle de traitement. Si c'est le cas, il traite la requête et revoie la réponse au client. Sinon, il la redirige vers le serveur approprié tout en envoyant un IAM au client. L'insertion d'un enregistrement peut engendrer un dépassement de capacité et donc un éclatement de la case.

Avant d'insérer une clé, le serveur vérifie qu'elle n'est associée à aucun enregistrement et qu'il existe suffisamment d'espace pour l'insérer. Dans le cas où la clé existe déjà, le serveur répond immédiatement au client pour le lui indiquer. Si l'espace libre dans la case ne suffit pas à insérer la clé, le serveur envoie une requête au coordinateur lui demandant d'allouer un nouveau serveur. Si aucun serveur n'est disponible, le serveur répond au client en précisant que la clé n'a pas pu être insérée par manque d'espace. Sinon, le serveur insère la clé, applique l'algorithme d'éclatement et envoie une réponse affirmative au client.

Algorithme

- **Client**

Appliquer l'algorithme de recherche CTH. m = Recherche(C)

Envoyer une requête d'insertion de la clé C vers m

- **Serveur**

```
Si  C < CM  (CM clé maximale du serveur m)
        Traiter la requête d'insertion
        Si  capacité dépassée
                Exécuter l'algorithme d'éclatement
        Fsi
      Envoyer la réponse au client
Sinon
        { Appliquer l'algorithme de recherche CTH.   m' = Recherche(C)
        { Rediriger la requête vers m'
        //

        Envoyer un IAM au client
Fsi
```

2.3.3 Suppression

Comme pour l'insertion, le client désirant supprimer une clé C commence par rechercher l'adresse du serveur contenant la clé. A la réception de la requête, le serveur vérifie que la clé appartient à son intervalle. Dans ce cas, il la traite et envoie la réponse au client. Sinon, il lui envoie un IAM et redirige la requête vers un autre serveur.

Pour supprimer une clé, le serveur commence par rechercher la clé. S'il ne la trouve pas, il envoie au client une réponse indiquant que la clé n'existe pas. Sinon, il la supprime et renvoie une réponse affirmative au client. A la suite d'une suppression, le nombre de clés de la case peut devenir inférieur à la moitié de sa capacité. Dans ce cas, une fusion de cases est opérée. Cette fusion peut engendrer d'autres fusions en cascade.

Algorithme

- **Client**

Appliquer l'algorithme de recherche CTH. m = Recherche(C)

Envoyer une requête de suppression de la clé C vers m

- **Serveur**

Si C < CM (CM clé maximale du serveur m)

Traiter la requête de suppression

Si $nb_clé \leq \frac{Capacité(m)}{2}$

Exécuter l'algorithme de fusion

Fsi

Envoyer la réponse au client

Sinon

$$\begin{cases} \text{Appliquer l'algorithme de recherche CTH. } m' = \text{Recherche}(C) \\ \text{Rediriger la requête vers } m' \end{cases}$$

//

Envoyer un IAM au client

Fsi

2.3.4 Requête à intervalle

Une requête à intervalle est une recherche ou une suppression de toutes les clés comprises entre deux valeurs C_{min} et C_{max}.

Pour effectuer une requête à intervalle, le client utilise son arbre digital afin de récupérer les adresses de tous les serveurs susceptibles de contenir des clés appartenant à l'intervalle [C_{min}, C_{max}]. Il leur envoie des requêtes à intervalle en précisant pour chacun l'intervalle [CM_{min}, CM_{max}] concerné. A la réception de la requête, le serveur compare l'intervalle reçu et l'intervalle de traitement et effectue le traitement approprié.

Algorithme

- **Client**

(m,CM) = Recherche(C_{min})
clé_min = C_{min}
Tant que C_{max} > CM
    Envoyer à m une requête sur l'intervalle [clé_min, CM]
    clé_min = CM
    m = suivant(m)
    CM = suivant(CM)
Ftq
Envoyer à m une requête sur l'intervalle [clé_min, C_{max}]

- **Serveur**

Le serveur m de clé maximale CM reçoit une requête à intervalle sur [CM_{min},CM_{max}]

clé_min = CM_{min}
Si CM_{min} ≤ CM
    Si CM_{max} ≤ CM
        Envoyer une réponse sur l'intervalle [CM_{min}, CM_{max}]
        Fin
    Sinon
        Envoyer une réponse sur l'intervalle[CM_{min}, CM]
        //
        Envoyer un message d'ajustement au client
        clé_min = CM
    Fsi
Sinon
    Envoyer un message d'ajustement au client
Fsi
Tant que C_{max} > CM
    Envoyer à m une requête sur l'intervalle [clé_min, CM]
    clé_min = CM
    m = suivant(m)
    CM = suivant(CM)
Ftq
Envoyer à m une requête sur l'intervalle [clé_min, C_{max}]

Le client dispose de deux stratégies pour terminer une requête à intervalle.

Stratégie déterministe. Chaque serveur précise dans sa réponse l'intervalle des clés traitées. Le client termine la requête lorsque l'union des intervalles des différentes réponses recouvre l'intervalle de la requête.

Stratégie probabiliste. Le client utilise un temporisateur ou timeout t pour collecter les réponses. Ce timeout est réinitialisé après la réception de chaque réponse. La requête est terminée lorsque t expire. Ce timeout doit être choisi de manière à réduire la probabilité de perte de réponses. Ce choix dépend non seulement des performances du réseau mais également de la vitesse de traitement des serveurs [DIE01]. Une solution consiste à choisir t égal à la somme du temps de traitement de la requête et du temps d'un aller-retour au serveur le plus éloigné. Cette solution est acceptable pour des réseaux fixes où les délais de propagation sont prévisibles mais elle est inutilisable dans un environnement mobile où les liaisons asymétriques rendent impossible le calcul des délais de propagation.

2.4 Mise à jour de l'image du client

L'image du client sur le fichier n'est pas nécessairement correcte. Elle est mise à jour de manière asynchrone suite aux erreurs d'adressage.

Soit m le serveur envoyant le message d'ajustement

Digit(b) une fonction retournant la chaîne constituée par la concaténation des digits de la branche b.

L'algorithme d'ajustement est décrit ci-après.

Algorithme

- **Serveur**

Soit CM = $C_1C_2\ldots C_k$ la clé maximale de m

CM' = $C'_1C'_2\ldots C'_{k'}$ la clé maximale de m selon l'arbre de client

N la taille du préfixe commun entre CM et CM'

Trouver dans l'arbre de m le serveur m' qui a CM' comme clé max

Trouver m" le serveur qui précède m' dans l'arbre du serveur

Envoyer au client les paramètres N, m' et la partie de l'arbre suivante : $C_{N+1}\ldots C_k$ …m''

- **Client**

Déduire CM (CM = $C'_1C'_2\ldots C'_N\ C_{N+1}\ldots C_k$)

Trouver dans l'arbre de client la première branche b qui vérifie Digit (b) +||...| ≥ CM, b = $b_1\ b_2\ \ldots b_{Taille(b)}$

Soit M la taille du préfixe commun entre CM et Digit (b)

M' la taille du préfixe commun entre CM' et CM" la clé maximale de m" selon la partie de l'arbre envoyée

Remplacer dans l'arbre digital la partie $b_{M+1}b_{M+2}\ldots b_{Taille(b)}\ldots$m par : $C_{M+1}\ldots C_K$ m…m" suivi par Taille(CM")-M'-1 nœuds Nils suivi par $C'_{M'+1}C'_{M'+2}\ \ldots\ C'_{k'}$ suivis par m'. Autrement dit remplacer $b_{M+1}\ldots$m par $C_{M+1}\ldots C_K$ m…m" $Nil_1\ Nil_2\ \ldots\ Nil_{Taille(CM")-M'-1}$ $C'_{M'+1}C'_{M'+2}\ \ldots\ C'_{k'}$ m'

*2.5 Travaux sur CTH**

Depuis la définition de ses principes fondamentaux en 2001 [ZEG04], la structure de données distribuée et scalable CTH* a fait l'objet de plusieurs mémoires d'ingénieur et de thèses de magistère.

La première version opérationnelle a été réalisée en 2002 [BIR02]. Elle se basait sur le modèle client/serveur, utilisait aussi bien des messages unicast que multicast et fonctionnait sous un environnement Linux. Une version similaire mais n'utilisant que des messages unicast a ensuite été développée [KHA04]. CTH* a aussi été adaptée à l'environnement

Windows et plusieurs améliorations ont été apportées notamment par l'introduction d'une nouvelle variante de CTH visant l'accroissement du facteur de chargement [SAA04]. Une version définissant une structure de case en T_tree a récemment été mise en œuvre [BOU05].

Des variantes se basant sur un arbre central ont également été implémentées; nous trouvons dans ce sens une version basée sur un modèle client/serveur [MER03] et une autre s'appuyant sur un modèle pair-à-pair (P2P) [SAI04].

Une variante utilisant uniquement le multicast a également été développée [AIT05].

2.6 Plateforme existante

Après avoir donné un panorama des travaux de recherche menés sur CTH*, nous décrivons dans cette section la plateforme CTH* fixe sur laquelle nous nous basons dans notre travail et telle qu'elle a été présentée par SAAD et BOULARIAS [SAA04].

La plateforme CTH* existante fournit aux applications une bibliothèque de fonctions qui leur permet de manipuler les fichiers et d'accéder aux données sans se préoccuper des mécanismes et des techniques utilisées par le gestionnaire CTH* ; les échanges de messages entre le client, le serveur et le coordinateur lui sont totalement masqués.

Un client est associé à un poste donné et gère les applications présentes sur la même machine que lui. Comme il utilise un port précis, il ne peut y avoir qu'un seul client par poste.

A chaque serveur est associé une adresse IP et deux ports, un pour l'émission et un autre pour la réception. Nous pouvons donc avoir plusieurs serveurs CTH* sur le même poste, la distinction se faisant au moyen des ports d'écoute. Le serveur peut contenir plusieurs cases CTH*, chacune appartenant à un fichier différent et associée à un numéro logique dans ce fichier. Il gère également l'ouverture, la fermeture et la suppression des cases des fichiers lui appartenant et contrôle l'accès aux données à travers les opérations de recherche, d'insertion, de mise à jour ou de suppression.

Le coordinateur gère les opérations de création, d'ouverture, de fermeture et de suppression de fichier et s'occupe de l'allocation lors des débordements.

La communication entre ces différents composants se fait à travers des messages unicast (point à point) et ne nécessite qu'un nombre réduit de messages.

La figure II.1 résume l'architecture de la plateforme CTH* existante.

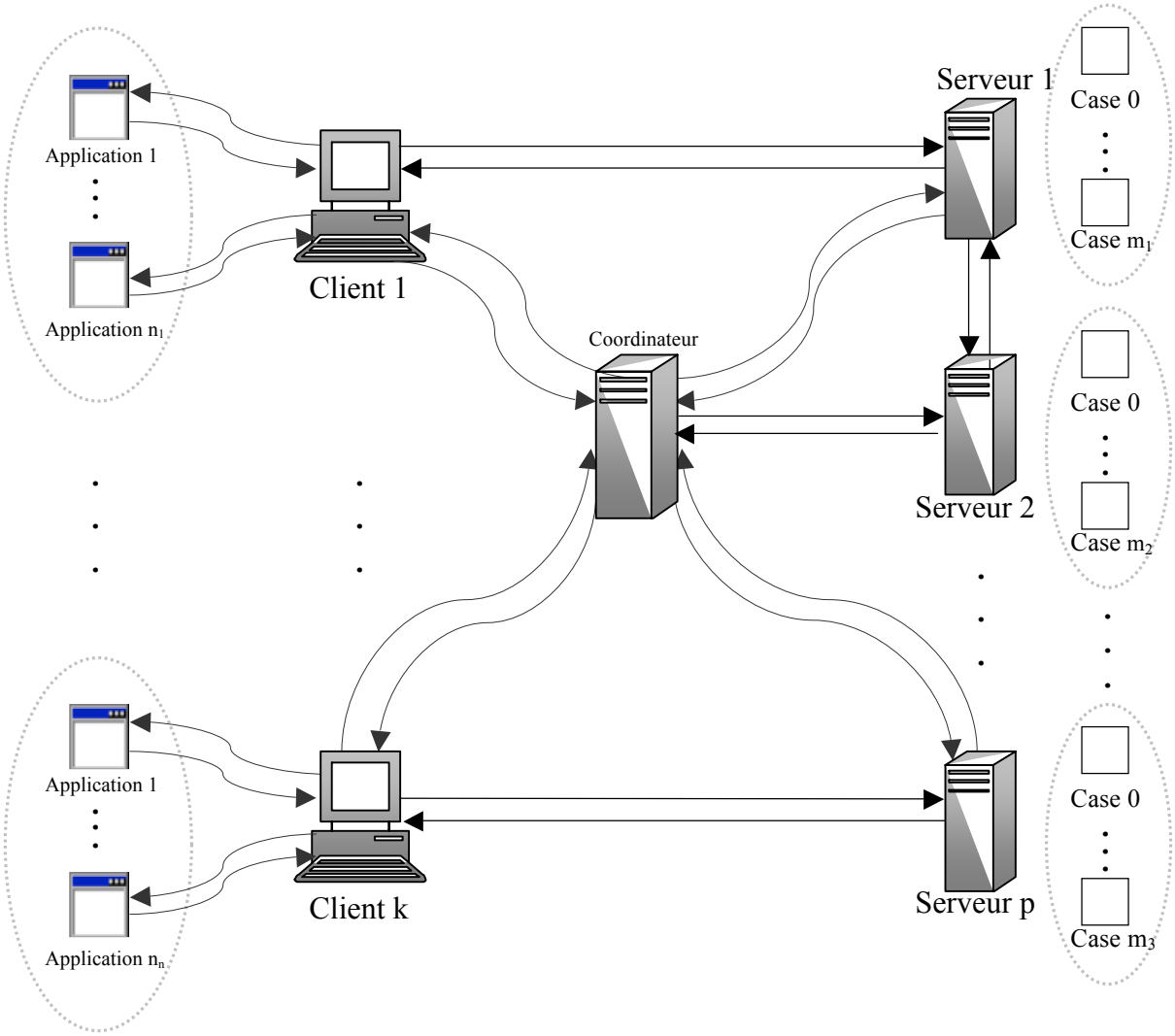


Figure II.2 Plateforme CTH* fixe

Notre objectif est d'introduire la mobilité au niveau du client et d'adapter le serveur pour qu'il puisse traiter aussi bien les clients fixes que mobiles. Nous devons également réutiliser au maximum le code existant et ne pas reprendre le développement de l'intégralité de l'application car celle qui existe est opérationnelle et efficace.

3 Synthèse

Après la description des principes et des algorithmes de la méthode CTH*, nous avons présenté la plateforme CTH* existante. Cette plateforme n'est composée que d'ordinateurs fixes communiquant à travers un réseau local haut débit.

Afin d'intégrer des clients mobiles à cette plateforme tout en garantissant des performances optimales, une connaissance des possibilités et des spécificités des environnements mobiles est indispensable.

Le chapitre qui suit décrit les propriétés et les caractéristiques des réseaux et téléphones mobiles.

Chapitre III

Environnements mobiles

Durant de longues années, les services offerts par les réseaux mobiles se réduisaient à la transmission de la parole ; les progrès technologiques nous entraînent désormais vers de nouvelles générations de réseaux mobiles qui augmentent les débits, offrent une infrastructure adaptée à la transmission de données et donnent naissance à une profusion de nouveaux services.

Parallèlement à l'essor des réseaux mobiles, nous assistons à la prolifération d'appareils de plus en plus performants qui offrent à leurs utilisateurs d'innombrables services tout en leur permettant de se déplacer sans restriction.

Les performances accrues des téléphones mobiles ainsi que l'évolution des réseaux mobiles ont facilité le développement d'applications nomades, diversifiées, centrées sur les données et à haute qualité.

Ce chapitre décrit une architecture générique des réseaux mobiles puis présente les différentes générations de ces réseaux. Il s'intéresse ensuite à la transmission de données sur les réseaux mobiles et aborde les techniques et les protocoles utilisés à cet effet. Il traite également des caractéristiques matérielles et logicielles des téléphones mobiles et se penche sur les plateformes de développement disponibles sur ces appareils.

1 Schéma d'architecture des réseaux mobiles

Les différents réseaux mobiles peuvent être définis à partir des équipements qui les composent (satellite, antenne, commutateur, carte à puce, terminal, etc.) et des interfaces mises en places entre ces équipements (interface air, sous système radio, partie fixe, boucle locale, etc.) pour assurer la communication [ALA01]. La figure III.1 représente le schéma du système réseau théorique le plus complet.

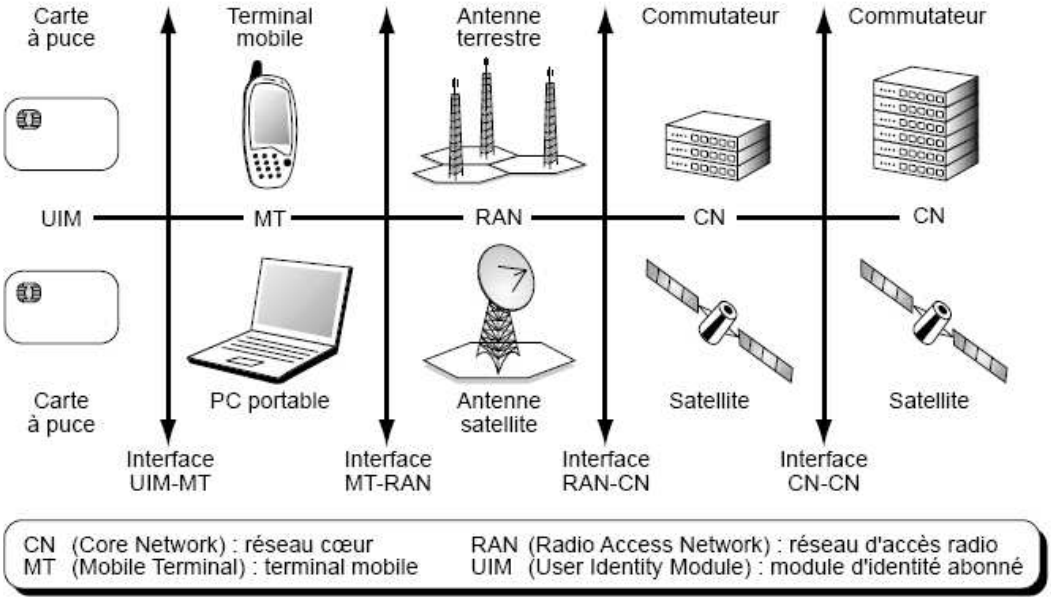


Figure III.1 Equipements et interfaces de base d'un réseau sans fil [ALA01]

La première interface relie la carte à puce ou carte UIM (User Idendity Module) et le terminal mobile. Ce terminal peut être un simple téléphone mobile ou un PC muni d'un lecteur de carte à puce. Le rôle principal de cette interface est de sécuriser la communication qui s'établit à partir du mobile.

La deuxième interface (interface MT-RAN) relie le terminal mobile à l'antenne ou éventuellement à un autre terminal. Dans le cas des systèmes satellitaires, cette interface permet la connexion directe du terminal au satellite. Elle concerne la traversée de la partie air du réseau et définit comment un terminal accède à l'antenne et réciproquement. L'interface air ou interface radio, est celle que l'on met en avant dans les réseaux de mobiles ou sans fil. Elle représente souvent le point le plus sensible du réseau car les ressources y sont rares et doivent être optimisées. De nombreuses limitations peuvent dégrader par ailleurs la qualité de service délivrée par cette interface.

La troisième interface correspond à la liaison entre l'antenne et le premier commutateur du réseau cœur. Le réseau cœur désigne le réseau filaire qui relie toutes les antennes composant le système. Théoriquement, le nombre de ces antennes peut être considérable et recouvrir un pays entier, voire un continent ou même l'ensemble de la surface terrestre. Comme il serait fastidieux de relier toutes ces antennes entre elles, on en rassemble un certain nombre pour former un sous-système d'accès ou réseau d'accès vers l'antenne. Cette

interface assure la gestion des appels en acheminant correctement chaque appel depuis ou vers l'antenne adéquate. Elle s'occupe également de la gestion de la mobilité puisque le client se déplace et peut se trouver connecté à une autre antenne, soit à l'intérieur du même sous-système, soit au sein d'un sous-système indépendant.

La quatrième interface se situe entre deux nœuds du réseau cœur. Elle permet de faire transiter les informations d'un sous-système d'accès à un autre.

2 Générations des réseaux mobiles

Il existe différents réseaux cellulaires qui se sont perfectionnés au fil du temps. L'historique suivant nous renseigne sur l'évolution de ces réseaux au niveau mondial.

2.1 Première Génération (1G)

La première génération de téléphonie mobile (notée 1G) possédait un fonctionnement analogique. Les principaux standards de cette génération sont brièvement décrits ci-dessous.

AMPS (Advanced Mobile Phone System). Apparu en 1976 aux Etats-Unis, l'AMPS constitue le premier standard de réseau cellulaire. Il utilise une méthode d'accès multiple fondée sur le multiplexage fréquentiel, le FDMA (Frequency Division Multiple Access) sur la bande des 800 MHz. Il a été déployé en Amérique du Nord, en Russie et en Asie.

NMT (Nordic Mobile Telephone). Mis en service en 1981 en Suède et en Norvège, le NMT est le seul système 1G à introduire la transmission de données sous forme de petits messages texte. Ce service est appelé DMS (Data and Messaging Service) et ne travaille qu'avec des débit allant de 600 à 1200 bit/s. Le NMT utilise une modulation de fréquence, FFSK (Fast Frequency Shift Keying) sur la bande des 450 MHz ou des 900 MHz. Il a été déployé dans les pays nordiques, en Asie, en Russie et en d'autres pays de l'Europe de l'est.

TACS (Total Access Communication System). TACS est la version européenne du modèle AMPS. Le TACS a été largement utilisé en Angleterre dès 1985. Il a aussi été utilisé au Japon sous la dénomination JTACS (Japanese Total Access Communication

System). Il utilise un accès multiple à répartition de fréquences sur la bande des 800 et des 900 MHz. Une version améliorée de ce standard a été développée au Royaume-Uni. Cette version est appelée ETACS (Extended Total Access Communication System). Elle consiste à utiliser un nombre plus important de canaux de communication dans la bande des 900 MHz.

Les réseaux analogiques de première génération n'offraient aucun support ni pour la sécurité ni pour la confidentialité. De plus, les terminaux étaient coûteux et volumineux. Par ailleurs, le prix élevé des communications et la capacité spectrale réduite rendait leur usage exclusivement professionnel.

Ces réseaux cellulaires ont été rendus obsolètes avec l'apparition d'une seconde génération entièrement numérique.

2.2 Deuxième génération (2G)

La prolifération de normes de systèmes cellulaires analogiques a conduit les pays européens à adopter une démarche commune pour le passage au numérique, ce qui a aboutit à la norme GSM. Elle marque un souhait de rationaliser les développements par une normalisation qui dépasse le seul intérêt national et qui vise à harmoniser l'usage des fréquences au niveau européen. Mais les approches sont encore très diversifiées entre Europe, Amérique et Asie. Ces systèmes acheminent encore majoritairement des communications vocales mais ils ont marqué une rupture avec la première génération en passant de l'analogique vers le numérique. Les réseaux numériques sont plus performants que les réseaux analogiques du point de vue fonctionnalité, sécurité et efficacité spectrale. De plus, comme ces réseaux utilisent de faibles puissances, les terminaux sont plus compacts et plus légers et les risques sur la santé sont quasiment inexistants. Enfin, les systèmes de deuxième génération ont introduit le principe d'itinérance qui permet à l'utilisateur nomade de continuer à utiliser son terminal mobile même en dehors de son réseau d'abonnement.

Les principaux standards de téléphonie mobile 2G sont exposés dans ce qui suit.

GSM (Global System for Mobile communications). Le système GSM utilise principalement la bande de fréquence des 900 MHz et une méthode d'accès radio de type TDMA (Time Division Multiple Access). Une extension dans la bande 1800 MHz a permis d'élargir le spectre dans certains pays européens. Dans le continent américain, une variante

du GSM a été déployée sur la bande des 1900 MHz. Ainsi, on appelle tri-bande, les téléphones mobiles pouvant fonctionner en Europe et en Amérique.

Quelle que soit la bande de fréquence utilisée, le GSM autorise également l'envoi et la réception de messages courts (SMS : Short Message Service), la transmission de données ainsi qu'un grand nombre de services supplémentaires.

IS-54 (Digital AMPS) et IS-136. Ces normes américaines décrivent les sous-systèmes radio (le réseau d'accès) et repose sur la norme IS-41 pour le réseau cœur.

L'IS-54 (Interim Standard 54) est la version numérique améliorée du système AMPS d'où la dénomination Digital AMPS. Basé sur la méthode d'accès TDMA, l'IS-54 a été implémenté par certains opérateurs cellulaires aux côtés de leurs réseaux analogiques AMPS dans la bande des 800 MHz. Ce standard a également été retenu pour être utilisé dans le cadre des réseaux PCS (Personal Communication Services) large bande sous l'appellation IS-136 [FOU01]. Une autre version appelée E-TDMA (Extended TDMA) a été proposée afin d'améliorer les performances des réseaux IS-54.

IS-95 ou N-CDMA (Narrowband CDMA). Issue de l'organisme américain TIA (Telecommunication Industry Association), la norme IS-95 a été proposée par la société Qualcomm. Elle utilise une technique d'accès multiple à répartition de code, CDMA (Code Division Multiple Access). L'IS-95 appelé aussi N-CDMA (Narrowband CDMA) a été déployé sur deux bandes de fréquences : 800 MHz et 1900 MHz. La norme IS-95 a été retenue dans quelques pays en dehors des Etats-Unis en particulier en Asie et en Amérique du Sud. Elle représente aussi l'unique système 2G fonctionnel en CDMA. Il est à noter que le CDMA est la méthode d'accès adoptée pour l'ensemble des réseaux de troisième génération.

PDC (Personal Digital Cellular system). Le PDC est la norme japonaise de radiotéléphonie cellulaire numérique. Elle s'appuie, comme le GSM et l'IS-136, sur le mode d'accès TDMA et fonctionne sur deux bandes de fréquences : 800 MHz et 1,5 GHz. Introduit au Japon en 1994 sous le nom de JDC (Japan Digital Cellular), le PDC est resté limité à ce seul pays.

Avec 65% du nombre d'abonnés à la téléphonie mobile dans le monde [ALA01], le GSM rencontre un succès commercial planétaire. Il n'en va pas de même des autres systèmes cellulaires, tels que le PDC japonais ou l'IS-136 et l'IS-95 américains, confinés à l'intérieur de leurs pays d'origine.

2.3 Evolutions de la deuxième génération (2,5G)

Victimes de leur propre succès, les réseaux de deuxième génération sont saturés. En outre, les utilisateurs deviennent de plus en plus exigeants et veulent que leur système mobile étende le service de voix à la transmission de données. C'est ainsi que plusieurs évolutions sont apparues.

GPRS (General Packet Radio Service). Le GPRS est une évolution du réseau GSM qui vise à améliorer l'efficacité du transfert de données. Fondé sur l'interface radio du GSM, mais développant un partage de ressources dynamique adapté au trafic sporadique, le GPRS introduit une architecture en mode paquet [ALA01]. Mais, si peu de changements ont été apportés au réseau d'accès, le réseau cœur a beaucoup évolué afin de permettre une interconnexion directe avec les réseaux de commutation de paquets tels qu'Internet. Ainsi, le réseau cœur est un réseau IP où de nouveaux éléments ont été introduits. Ces éléments sont : le SGSN (Serving GPRS Support Node) et le GGSN (Gateway GPRS Support Node). Le SGSN est responsable de l'acheminement des paquets entre le mobile et le réseau fixe. Il gère en conséquence les contextes de mobilité et de sécurité du mobile. Le GGSN est une passerelle qui gère la facturation et le routage des paquets provenant du SSGN vers les réseaux de commutation de paquets externes (Internet par exemple). Le réseau GPRS et le réseau GSM fonctionnent en parallèle : le premier est utilisé pour la transmission de données et le second pour les services de voix.

EDGE (Enhanced Data Rates for GSM Evolution). EDGE est une autre amélioration qui permet de s'affranchir des limites du GSM par l'introduction d'une nouvelle méthode de modulation de phase, la 8PSK (Eight-level Phase Shift Keying), de nouveaux schémas de codage et la généralisation du principe de l'adaptation de lien. La norme EDGE peut s'appliquer au GPRS. Leur association est connue sous le nom de E-GPRS (Enhanced-General Packet Radio Service). L'E-GPRS est souvent considéré comme un système de troisième génération ou de génération intermédiaire 2,75G.

Evolution des systèmes CDMA IS-95. Le standard de seconde génération américain est passé par plusieurs phases : A, B et C. Ces développements ont apporté des améliorations en matière de capacité et autorisent la transmission de données en mode paquet à 144 Kbit/s. Le mode paquet à haut débit est implémenté dans la version IS-95-HDR (High Data Rate).

Evolution des réseaux TDMA IS-136. Les réseaux TDMA IS-136 ont entamé en 1999 un processus de convergence vers le GSM et les réseaux de troisième génération en adoptant les principes de la norme EDGE mais en conservant le sous-système réseau compatible IS-41. Cette évolution est connue sous le nom d'UWC-136 (Universal Wireless Communication 136).

PDC-P (Packet Data Cellular). Appelé aussi "DoPa" (Docomo Packet), le PDC-P est un système propriétaire de NTT DoCoMo. Il autorise des transmissions de données souples sur le système cellulaire japonais PDC et offre des connexions à Internet et à des intranets en étendant la commutation de paquets à l'accès radio. Cela permet de disposer d'un système homogène à rapport coût/performance optimisé et dans lequel la commutation de paquets est une fonction totalement intégrée.

Les limitations des réseaux de deuxième génération ont vite été repoussées par les systèmes 2,5G. Ces évolutions ont permis de répondre aux exigences des utilisateurs rapidement et à un coût modéré. Néanmoins, à plus long terme, il est plus avantageux d'utiliser une technologie de troisième génération offrant de meilleures performances.

2.4 Troisième génération (3G)

C'est en 1985, dans le programme FPLMTS (Future Public Land Mobile Telecommunication System), que l'UIT (Union Internationale de Télécommunication) propose pour la première fois de définir un système de communication sans fil visant à réaliser la convergence fixe-mobile, disposant d'une bande de fréquences universelle, autorisant l'utilisation d'un terminal mobile unique et intégrant les composantes terrestres et satellitaires. Elle fixe à ce projet un certain nombre de caractéristiques techniques, telles que les débits, de 144 Kbit/s à 2 Mbit/s suivant la mobilité et la possibilité d'avoir en parallèle une commutation de paquets et une commutation de circuits.

Ce programme est devenu l'IMT2000 (International Mobile Telecommunication system 2000). Pour tenter de structurer le développement des systèmes de troisième génération, l'UIT a proposé une méthodologie en cinq étapes.

1. Dans un premier temps, les groupes régionaux tels que l'ETSI (European Telecommunication Standard Institute) en Europe, ARIB (Association of Radio Industries and Business) au Japon ou TIA (Telecommunications Industry Association) aux Etats-Unis développent des solutions.
2. Ces solutions sont ensuite proposées à l'UIT.
3. L'UIT évalue et extrait les points communs de ces propositions
4. L'UIT recherche des compromis et suggère des modifications permettant d'harmoniser les solutions.
5. A partir de ces solutions convergentes, l'UIT fait des recommandations.

C'est ainsi que l'ETSI a proposé l'UMTS (Universal Mobile Telecommunications System), ou plus exactement l'UTRA (UMTS Terrestrial Radio Access), la partie terrestre de l'UMTS, qui devait à l'origine, intégrer un segment satellitaire. On trouve par ailleurs le W-CDMA (Wideband Code Division Multiple Access) proposé par l'ARIB et le CDMA2000 proposé par l'organisme américain TIA.

W-CDMA (Wideband CDMA). C'est la norme proposée par l'ARIB. Sa dénomination est due à l'utilisation d'une méthode d'accès CDMA large bande (Wideband CDMA). Le qualificatif "large bande" provient du fait que, comparativement au seul système CDMA existant à l'époque, l'IS-95, le W-CDMA requiert des canaux beaucoup plus larges. Une des caractéristiques principales du système est sa flexibilité qui lui permet d'accepter des variations de débits importantes.

Le W-CDMA utilise la bande de fréquences 1885-2025 MHz et celle des 2110-2200 MHz.

UMTS (Universal Mobile Telecommunications System). L'UMTS est la solution proposée par l'ETSI dans le cadre du programme IMT 2000. Cette norme marie deux modes : FDD (similaire au W-CDMA) et TDD. Le mode FDD (Frequency Division Duplex) utilise un duplex en fréquences dans lequel la voie montante et la voie descendante sont séparées en fréquences. Le terminal mobile et la station de base communiquent en continu et

simultanément dans les deux directions. La communication entre la station de base et le mobile s'établit par une technique d'étalement de spectre, qui est à la base du CDMA [ALA01]. Le TDD (Time Division Duplex) est, quant à lui, fondé sur une méthode hybride entre le TDMA et le FDMA. Comme son nom l'indique, la séparation entre le sens montant et le sens descendant se fait dans le temps. Certains slots de la trame sont dédiés à la voie montante et d'autres à la voie descendante. La répartition des slots dans la trame est flexible. Le TDD peut donc supporter très facilement du trafic asymétrique alors que, par construction, le mode FDD est symétrique. Pour ces raisons, le TDD est mis en œuvre dans les bandes non appariées, contrairement au FDD, qui est utilisé dans les bandes appariées.

En décembre 1998, l'UMTS est transféré au sein du 3GPP (Third Generation Partnership Project), avec comme première conséquence une harmonisation avec la solution W-CDMA de l'ARIB. Dès lors, la solution technique se précise, et une activité plus importante du mode FDD se dessine, même si le TDD reprend de l'importance du fait de l'intérêt que lui porte la Chine.

Le réseau cœur de l'UMTS a initialement été semblable à celui du GPRS. Le 3GPP a ensuite rendu obligatoire l'adoption de IPv6 pour les autres versions (release).

CDMA2000. Les différentes améliorations apportées à l'IS-95 ont permis de le faire passer vers le mode paquet. Néanmoins, ce nouveau service est loin d'atteindre l'optimisation souhaitée du fait de son insertion dans des infrastructures initialement conçues pour la voix. Le passage vers la troisième génération impose naturellement un mode paquet plus performant. Cette transition peut se faire en souplesse car l'IS-95 (appelé aussi CDMAOne) et le CDMA2000 peuvent cohabiter sur une même bande de fréquence grâce à la modulation multiporteuse. Une phase ultérieure permettra au CDMA200 d'augmenter les débits et de renforcer la sécurité. Ceci est possible grâce à l'adoption de bandes de fréquences élargies. Aussi, le réseau cœur est similaire à celui de la deuxième génération, à savoir l'IS-41, mais devrait évoluer pour permettre des performances optimales.

La troisième génération visait l'harmonisation des différentes normes au niveau mondial. Même si l'UMTS et le W-CDMA sont totalement compatibles, la divergence de leurs principes avec ceux du CDMA2000 élimine toute compatibilité entre ces systèmes. La différence se situe au niveau de la méthode de transmission, de la taille de la trame et de la

signalisation du réseau cœur. Ainsi, les systèmes de quatrième génération tentent d'homogénéiser tous ces systèmes et de faire en sorte que la mobilité soit totalement transparente.

2.5 Quatrième génération (4G)

La quatrième génération de réseaux mobiles fait appel aux techniques dites ITS (Intelligent Transportation Systems) qui permettent d'associer différentes technologies en un même ensemble. A l'intérieur des bâtiments, ce sont principalement les systèmes sans fil de type WPAN (Wireless Personnel Area Network) ou WLAN (Wireless Local Area Network) qui sont utilisés par le terminal pour se connecter vers l'extérieur. Dans les environnements moins fortement urbanisés, les techniques GPRS et UMTS peuvent prendre le relais, tandis que, dans les lieux peu peuplés, le satellite se substitue aux autres techniques pour maintenir la connexion avec les réseaux d'opérateurs.

L'objectif de la quatrième génération est de réaliser l'intégration de ces différentes composantes pour que l'utilisateur soit en mesure de choisir à tout instant la meilleure solution pour se connecter vers l'extérieur, c'est-à-dire celle qui correspond le mieux à ses besoins et au tarif le plus bas possible.

Le HSDPA (High Speed Downlink Packet Access). Fondé sur le W-CDMA, le HSDPA se concentre sur l'optimisation du lien descendant, entre la station émettrice et le terminal mobile. L'optimisation du lien ascendant est traitée par le HSUPA (High Speed Uplink Packet Access). L'amélioration consiste à effectuer une transmission par "canaux partagés" (High Speed Downlink Shared Channels) où les ressources sont affectées à chaque utilisateur connecté selon ses besoins et la capacité de son terminal. De plus, une technique de modulation de fréquence est introduite, la QPSK (Quadrature Phase Shift Keying) et une méthode d'adaptation des paramètres de transmission est utilisée. Le HSDPA offre un débit maximal théorique de 14,4 Mbit/s. Mais, pour l'heure, les rares terminaux compatibles (qui ne sont pas encore commercialisés) ne peuvent offrir que 4 Mbit/s en débit crête.

Flash OFDM (Flash Orthogonal Frequency Division Multiplexing). OFDM est une technique de multiplexage consistant à transporter le signal sur de multiples fréquences porteuses. Celles-ci sont modulées individuellement et transmises de manière orthogonale pour limiter les interférences entre les signaux. De cette manière, la technologie Flash OFDM permet des connexions mobiles dix fois plus rapides que l'UMTS. Elle est d'abord testée sur des ordinateurs portables où elle autorise des connexions Internet à haut débit sans coupure même en mouvement. Elle utilise les mêmes fréquences radio et les mêmes antennes que les systèmes de téléphonie mobile de troisième génération.

WiMax Mobile. Le WiMax (IEEE 802.16) est une norme WMAN (Wireless Metropolitan Area Network). Il offre une large portée, de l'ordre de plusieurs kilomètres, et des débits de plusieurs dizaines de mégabits par seconde.

Depuis décembre 2002, une évolution "mobile" du WiMax a été développée. Le WiMax mobile est plus performant que la 3G pour prendre en charge les services de données, tels que les téléchargements de vidéos. De nombreuses sociétés sont très intéressées par le WiMax mobile ; citons à titre non exhaustif LG Electronics, Samsung, Motorola, Nokia et Nortel. D'autres comme Ericsson lui préfèrent la technologie HSDPA.

Enfin, la cinquième génération est évidemment plus lointaine, mais plusieurs solutions sont déjà envisagées. La plupart d'entre elles exploitent des satellites géostationnaires, des ballons stratosphériques à haute altitude ou des avions de type AWACS à basse altitude.

La figure III.2 résume les différents systèmes des trois premières générations.

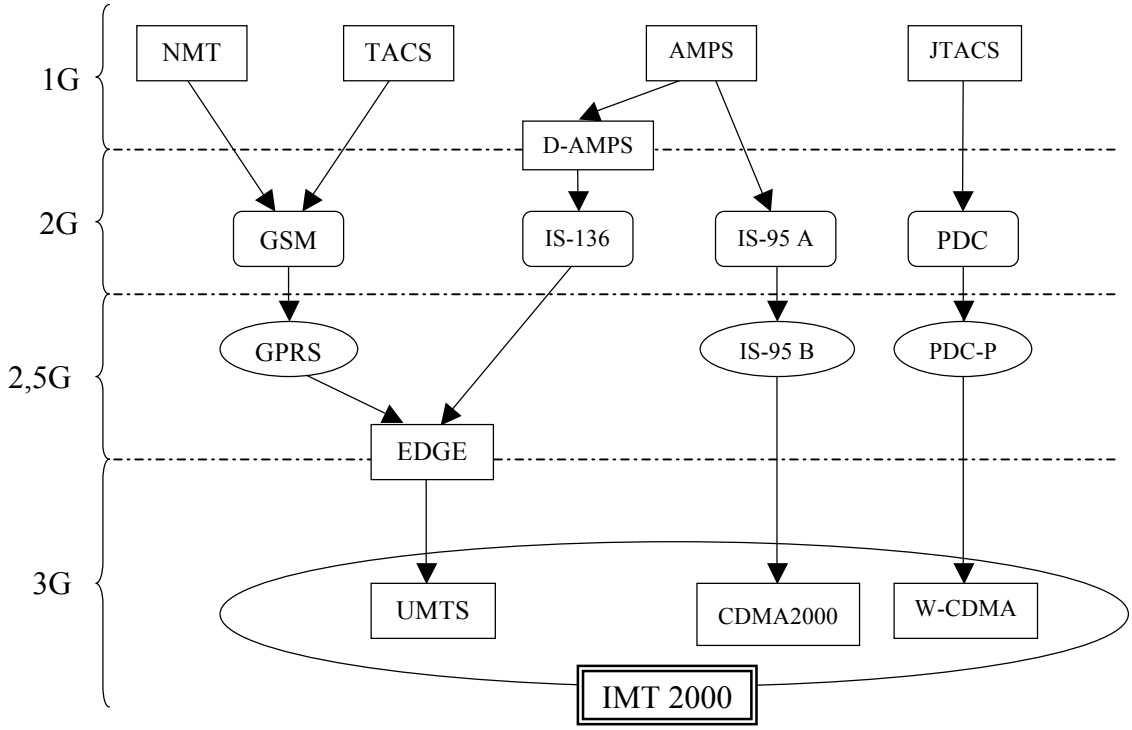


Figure III.2 Les générations de réseaux mobiles

3 Transmission de données dans les réseaux mobiles

La première génération de réseaux de mobiles n'avait qu'une seule application, la parole. Le passage à la deuxième génération et la numérisation des systèmes ont permis d'entrevoir l'idée d'une transmission de données. Cependant, les performances réduites des réseaux de cette génération ne permettaient pas de réaliser des applications de haute qualité. Il faudra attendre le passage au mode paquet pour voir une profusion des services de données.

Cette section dresse un panorama des différentes technologies de transmission de données.

3.1.1 GSM DATA OU CSD (Circuit Switched Data)

Le CSD est le premier système de transmission de données pour les réseaux basés sur la technique TDMA tel que le GSM. Pour acheminer la voix dans un réseau GSM, un circuit de communication est établi, maintenu puis relâché entre un utilisateur mobile et son correspondant. Sur la voie radio, un circuit est en fait un intervalle de temps (IT) dans une trame de données. Dans la majorité des cas ce circuit est utilisé pour transporter la voix, mais rien ne s'oppose à y faire transiter des données, c'est-à-dire tout type d'information numérique. Ainsi, la transmission de données en mode circuit permet de disposer sur le

réseau GSM d'un service équivalent à celui disponible sur le réseau téléphonique avec un modem. Le débit maximal théorique au niveau de la couche physique est de 33,86 Kbit/s. Mais en tenant compte des erreurs, les données sont codées par l'interface radio et un IT correspond donc à un débit de 14,4 Kbit/s au niveau de la couche d'interface radio. Enfin, au niveau des protocoles plus élevés (TCP/IP : Transmission Control Protocol/Internet Protocol), le débit utile pour l'utilisateur est 9,6 Kbit/s.

Cependant, même si les réseaux GSM sont capables de véhiculer des données, ils le font d'une manière qui n'est pas adaptée. Le débit est limité même avec des mécanismes applicatifs pour compresser les informations. De plus, l'usage des ressources radio n'est pas optimal car un circuit est réservé dans chaque sens alors qu'en général, un seul sens est utilisé à un moment donné. Par ailleurs, l'interconnexion avec les réseaux externes est assez complexe et le temps de connexion aux réseaux à commutation de paquets tel qu'Internet est assez long. Enfin, la facturation est basée sur le temps de connexion de l'utilisateur, ce qui fait que le coût des communications tend à s'alourdir sensiblement du fait à la fois du téléchargement des données à bas débit et du temps de lecture des pages.

3.1.2 HSCSD (High Speed Circuit Switched Data)

Le service HSCSD est un service de données en mode circuit, à débit élevé, qui consiste à allouer non plus un intervalle (IT) par utilisateur et par trame TDMA mais plusieurs, jusqu'à quatre, soit la moitié de la trame. Comme pour le service de données précédent, l'allocation des ressources se fait en mode circuit, par connexion et pour toute la durée de la connexion. Cette allocation peut être asymétrique, c'est-à-dire offrir plus de débit dans un sens que dans l'autre. A l'origine, les slots pouvaient être non consécutifs dans la trame, mais, pour des raisons de complexité d'implémentation dans les terminaux, les constructeurs ont imposé une allocation contiguë. Le débit maximal offert par le HSCSD est de 57,6 Kbit/s soit quatre slots à 14,4 Kbit/s. Même si ce débit est comparable à ceux obtenus avec des modems téléphoniques pour une connexion filaire, le HSCSD souffre d'un manque de souplesse et d'efficacité dans l'allocation de ressources radio.

Finalement, les constructeurs et les opérateurs n'ont pas investi dans les évolutions logicielles et matérielles à apporter aux terminaux et aux stations de bases pour incorporer le service HSCSD dans les réseaux GSM, car en parallèle, était standardisé le GPRS beaucoup plus prometteur.

3.1.3 GPRS (General Packet Radio Service)

Les limitations en débit du GSM ont été vite repoussées par le HSCSD, qui autorise l'allocation de plusieurs timeslots à un utilisateur, multipliant d'autant son débit. L'architecture du réseau reste toutefois toujours dévolue au monde de la téléphonie, inadapté au transfert des données. Le GPRS résout ce problème en définissant une architecture de réseau à commutation de paquets qui permet de n'allouer des ressources à un utilisateur qu'au coup par coup, lorsqu'il a réellement des données à émettre ou à recevoir, et non durant toute la durée de sa connexion, ainsi que de facturer les communications au volume de données échangées (en kilobit) et non à la durée de connexion. De plus, le GPRS introduit de la flexibilité dans le choix du type de protection des données, offrant ainsi une plus grande variété de débits [ALA01]. Le débit maximal théorique est de 170 Kbit/s. Cependant, dans la pratique, les débits disponibles varient entre 30 et 70 Kbit/s [PAR05].

3.1.4 EDGE (Enhanced Data Rates for GSM Evolution)

En utilisant plusieurs timeslots dans le HSCSD et le GPRS, un utilisateur peut augmenter son débit. Le GPRS offre de surcroît différents taux de codage, permettant d'augmenter le débit lorsque les conditions de propagation sont correctes. Néanmoins, le débit brut sur un timeslot reste celui du GSM, c'est-à-dire environ 14,4 Kbit/s. EDGE permet de s'affranchir de cette limite, moyennant l'introduction d'une nouvelle modulation, de nouveaux schémas de codage et la généralisation du principe de l'adaptation de lien (link adaptation). EDGE permet ainsi d'augmenter le débit d'un intervalle de temps à environ 43,2 Kbit/s. Lorsque EDGE est appliqué au GPRS, il permet d'obtenir des débits allant jusqu'à 384 Kbit/s. Ces débits sont très proches de ceux offerts par les systèmes de troisième génération et permettent de concevoir des applications de haute qualité notamment multimédia.

3.1.5 UMTS (Universal Mobile Telecommunications System)

La troisième génération de mobiles voit ses débits augmenter dans des proportions importantes. Théoriquement, il est possible d'atteindre des débits de 144 Kbit/s avec une couverture totale et une utilisation mobile, 384 Kbit/s avec une couverture moyenne et une utilisation piétonne et 2 Mbit/s avec une zone de couverture réduite et une utilisation fixe. En réalité, les débits se situent entre 64 à 128 Kbit/s en émission et 128 à 384 Kbit/s en

réception en réalité. Ce "haut débit" mobile n'est pas si éloigné des vitesses de transmission proposées dans le cadre des premières offres d'accès à l'Internet par l'ADSL ou le câble.

De plus, les réseaux UMTS sont capables de supporter des montées en charge bien plus importantes que leurs prédécesseurs. Les apports de cette technologie se manifestent par une moindre saturation et une meilleure qualité de transmission. A cela s'ajoute l'importante capacité de traitement des terminaux 3G, qui embarquent des processeurs puissants et des mémoires de grande capacité. Ces performances facilitent le développement d'applications multimédia qui intègrent la voix, la vidéo et les données. Ces applications sont par exemple, la visiophonie, la transmission audio et vidéo en temps réel (streaming), l'envoi de message MMS incluant des données audio et vidéo ou la télévision mobile. De plus, la 3G permet de disposer d'un accès rapide au web et à la messagerie. Par ailleurs, la sécurité renforcée de ces réseaux permet le déploiement d'applications financières telles que les services d'achats et le paiement par téléphone, et développent de plus en plus le commerce mobile (m-commerce). Enfin, d'autres applications apparues avec la 2,5G connaissent une évolution remarquable avec la 3G ; citons par exemple les jeux 2D et 3D en réseau et la navigation routière par GPS (Global Positioning System).

4 Protocoles de transmission de données

Si le standard TCP/IP demeure incontestablement le protocole qui convient le mieux aux transmissions de données filaires, il est mal adapté aux environnements mobiles. Ainsi, des systèmes ont été établis afin de simplifier l'échange d'information entre les terminaux mobiles et les réseaux fixes (Internet par exemple). L'objectif principal de ces protocoles de transmission de données dans des environnements mobiles est de permettre aux utilisateurs d'accéder à des contenus toujours plus riches avec un confort maximal. Pour cela, deux systèmes coexistent et permettent d'établir un lien entre les réseaux de données fixes et la téléphonie mobile : le protocole WAP et le service i-mode.

4.1 WAP (Wireless Application Protocol)

Le WAP est un protocole standard universel qui rend possible non seulement l'accès au contenu Internet mais aussi l'apport de nouvelles applications multimédia et services ajoutés aux téléphones mobiles et autres appareils sans fil. Le WAP est optimisé pour la taille de l'écran des terminaux, adapté aux débits et aux temps de latence des réseaux mobiles et indépendants du type de réseau de communication utilisé. En Europe, il est donc développé

sur les systèmes GSM, GPRS ou UMTS. Aux Etats-Unis, il fonctionne aussi bien sur les systèmes TDMA (l'IS-136) que sur les systèmes CDMA (l'IS-95 ou le CDMA2000). Au Japon, il utilise les systèmes PDC ou W-CDMA comme support de transmission.

C'est en 1997 que les principaux fabricants d'appareils de télécommunications (Nokia, Motorola, Ericsson,…) constituent le WAP Forum pour mettre en œuvre un protocole commun. Ce protocole devait permettre aux téléphones portables d'accéder à des informations ou des services circulant par le biais des réseaux mobiles en utilisant le vecteur Internet. Le WAP est aujourd'hui devenu un standard mondial pour les opérateurs, les fabricants, les développeurs d'applications et les fournisseurs de contenus. Hébergés par des serveurs Internet, les services WAP sont adaptés aux fonctionnalités des terminaux mobiles : instantanéité, interactivité, navigation sous forme de listes de menus et sous-menus, images simples, etc. Plusieurs versions ont été élaborées. Aujourd'hui, la dernière est la version 2.0.

4.1.1 Architecture du WAP

Quelle que soit sa nature, un service WAP s'appuie toujours sur une architecture client/proxy/serveur (figure III.3). Ainsi, le WAP est constitué de trois composants : le terminal mobile muni du micronavigateur WAP, la passerelle WAP et le serveur d'applications.

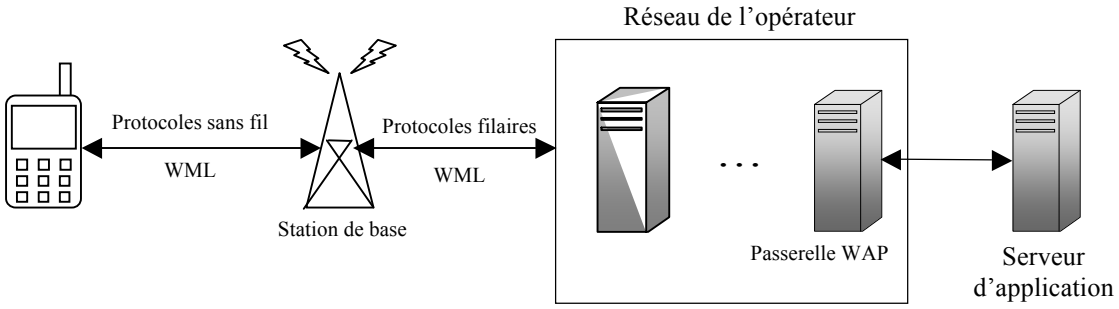


Figure III.3 Architecture d'un réseau WAP

Micronavigateur WAP

A l'instar de Netscape Navigator ou de Microsoft Internet Explorer pour les pages Web, le micronavigateur WAP intégré au téléphone gère l'affichage des données transmises par la passerelle WAP. Sauf que contrairement aux navigateurs précédents, qui supportent le HTML (HyperText Markup Language), le WAP utilise un langage de balisage particulier. Les versions 1.x utilisent WML (Wireless Markup Language) qui est dérivé du XML

(eXtensible Markup Language). La version 2.0 introduit le langage xHTML (extensible HTML) dont le principal avantage est d'avoir la même qualité d'affichage aussi bien sur un PC que sur un téléphone mobile.

La passerelle WAP

Elle assure l'interconnexion entre le réseau informatique et le réseau de communication cellulaire, ainsi que le codage et le décodage des requêtes échangées entre le terminal mobile et le serveur Web.

Le serveur d'application

Il assure l'accès aux informations contenues dans des bases de données ou des serveurs d'entreprise à travers un ensemble de connecteurs spécialisés. Seulement, au lieu de transmettre ces informations en HTML, il les envoie en WML ou en xHTML (selon la version du protocole WAP)

Lorsqu'un utilisateur demande l'exécution d'un service WAP, il émet une requête HTTP au serveur d'application. Cette requête est transmise par la passerelle WAP jusqu'à ce dernier. Le serveur réalise l'action demandée et renvoie le résultat au format WML ou xHTML. Ce résultat est codé en binaire par la passerelle puis envoyé au téléphone. Le terminal peut dès lors l'afficher dans le format demandé : tableau, texte, menu, etc.

4.1.2 Pile protocolaire du WAP

Dans ces versions 1.x, le WAP comprenait plusieurs couches :

- une couche transport – WDP (Wireless Datagram Protocol),
- une couche sécurité – WTLS (Wireless Transport Layer Security),
- une couche transaction – WTP (Wireless Transaction Protocol) souvent désigné comme le HTTP des réseaux mobiles,
- une couche session – WSP (Wireless Session Protocol),
- une couche applicative – WAE (Wireless Application Environment).

Cependant, ce découpage s'est révélé inefficace. Ainsi, la version 2.0 est basé sur le protocole TCP : des protocoles HTTP/TCP/IP optimisés sont utilisés pour la transmission radio et les protocoles HTTP/TCP/IP standards sont utilisés pour la transmission sur le

réseau fixe. La passerelle qui responsable des conversions entre ces deux types de protocoles.

4.2 i-mode

Le standard japonais i-mode utilise le langage C-HTML (Compact-HTML), et non le WML du WAP. Le C-HTML est un sous-ensemble du HTML, destiné à fonctionner sur des navigateurs mobiles allégés. L'ergonomie globale des téléphones i-mode reste cependant assez proche des téléphones WAP.

Comme le langage utilisé par l'i-mode est à base de HTML, il ne nécessite pas d'étape intermédiaire de conversion en binaire, à l'inverse du WAP qui réclame l'emploi d'une passerelle. L'accès au serveur distant se fait donc directement sans intermédiaire.

Grâce à i-mode, à son business model novateur et à ses services à réelle valeur ajoutée pour les abonnés, NTT Docomo est devenu le plus gros fournisseur d'accès Internet du Japon et le trafic généré par l'Internet mobile est à présent supérieur au trafic généré par la parole téléphonique.

Bien qu'il n'aie pas eu le succès attendu, le WAP possède certains avantages par rapport à l'i-mode. En effet, le WAP est un protocole au format ouvert, c'est-à-dire qu'un terminal mobile compatible WAP permet de visualiser des sites WAP mais aussi des sites i-mode alors que l'i-mode ne permet de visualiser que des sites compatibles i-mode. L'i-mode quant à lui est un service propriétaire de NTT Docomo.

5 Téléphone mobile

Après avoir décrit les différentes générations de réseaux mobiles et exposé les techniques et les protocoles de transmission de données sur ces réseaux, nous nous intéressons dans cette section aux moyens d'accéder à ces technologies.

Parmi les différents terminaux mobiles existants (ordinateur portable, PDA[1], smartphone[2] …), le téléphone reste le moyen le plus pratique pour profiter pleinement des

[1] PDA : Personnel Digital Assistant, assistant personnel

[2] Smart phone peut se traduire par téléphone intelligent

possibilités du réseau. Les variétés, formes, ergonomies et technologies des téléphones mobiles évoluent sans cesse et à une vitesse vertigineuse. Ainsi, nous voyons se développer une nouvelle génération de téléphones intelligents qui offrent non seulement des services de voix mais aussi un accès simplifié au web, à la messagerie et aux services de transmission de données. Avec un écran couleur, des logiciels évolués et un navigateur, les utilisateurs peuvent accéder à leurs informations personnelles et professionnelles en toute sécurité et fiabilité sur des appareils qui tiennent dans la paume de la main et ne pèsent que quelques dizaines de grammes.

De plus, les téléphones mobiles peuvent être utilisés comme modem pour des PC (Personnel Computer), des ordinateurs portables ou des PDA non connectés afin de pouvoir accéder à Internet n'importe où et n'importe quand.

Les téléphones mobiles se transforment petit à petit en véritables ordinateurs ; grâce au progrès de la microélectronique, il est désormais possible d'y intégrer la plupart des composants d'un PC (microprocesseur, mémoire …). De plus, ils sont gérés par des systèmes d'exploitation de plus en plus évolués.

Cette section détaille l'architecture matérielle et la plateforme logicielle de ces appareils.

5.1 Composants matériels

La miniaturisation des circuits intégrés a conduit à une intégration de la plupart des composants d'un ordinateur au sein d'une seule puce. Cette puce est au cœur du téléphone mobile et regroupe plusieurs éléments : microprocesseur, DSP (Digital Signal Processor), mémoire. Ces éléments permettent de gérer différents périphériques tels que l'écran, le clavier et les équipements multimédia. Ils permettent également d'optimiser la consommation d'énergie et de traiter les signaux émis ou reçus par le module radio. Comme il est possible d'utiliser le téléphone en tant que modem pour un PC ou un PDA, il existe différents mode de connectivité locale : câble USB, infrarouge ou Bluetooth.

Le téléphone contient également un lecteur de carte SIM[3] (Subscriber Identity Module). La carte SIM est une carte à puce qui contient les caractéristiques de l'utilisateur, de son abonnement et quelques services offerts par l'opérateur téléphonique.

Cette section décrit succinctement chaque composant.

[3] Dans le cas des réseaux UMTS, elle est appelée U-SIM (UMTS SIM)

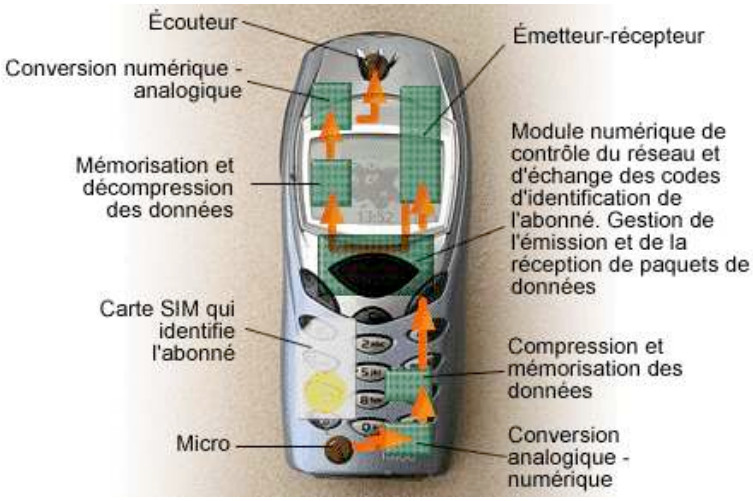


Figure III.4 Vue générale d'un téléphone mobile [WIK06]

Processeur. Le processeur est l'élément le plus important du téléphone mobile. Ils contrôle et coordonne le travail des différents modules et gère l'interface homme-machine (HMI : Human Machine Interface).

Deux types de processeurs sont utilisés : les processeurs RISC (Reduced Instruction Set Computer) ou plus exactement ARM (Advanced RISC Machine) et les processeurs CISC (Complex Instruction Set Computer). La fréquence d'horloge est généralement de l'ordre d'une centaine de mégahertz mais Intel a pu montrer, avec son XScale, que des fréquences de 1GHz pouvaient être atteintes. Ni Motorola Dragon Ball ni Texas Instrument OMAP n'ont pu réaliser de telles performances.

Ces processeurs sont généralement intégrés dans une puce, qui contient également un DSP et de la mémoire pour embarquer un système complet sur une puce (SoC : System on Chip).

DSP (Digital Signal Processor). Le DSP est un processeur dédié au traitement du signal. Il s'occupe des conversions entre numérique et analogique. Il gère également la détection et la correction des erreurs de transmission et aiguille les informations reçues vers les modules adéquats.

L'utilisation du DSP permet de décharger le processeur central et d'augmenter la puissance et la vitesse de traitement.

Mémoires. Généralement embarquée sur la même puce que le microprocesseur, la SRAM (Static Random Access Memory) est utilisée en tant que mémoire cache et permet d'accélérer considérablement la vitesse de traitement.

La ROM (Read Only Memory) est, quant à elle, utilisée pour stocker le système d'exploitation. Elle tend à être remplacée par des ROM Flash qui peuvent être reprogrammées afin de mettre à jour le système d'exploitation.

De plus, des mémoires Flash externes amovibles peuvent être utilisées pour stocker les données et les applications de l'utilisateur. Il existe plusieurs types de mémoires flash : SmartMedia, CF (Compact Flash), MMC (Multimedia Memory Card) ou SD (Secure Digital). Ces technologies se basent sur des standards industriels et sont adoptées par plusieurs sociétés. Ces mémoires peuvent stocker de grandes quantités de données ; à titre d'exemple, la carte CF peut contenir jusqu'à 4Go de mémoire.

Enfin, sur certains téléphones haut de gamme, il est possible de trouver un disque dur de quelques gigaoctets.

Module radio. Le module radio (RF unit : Radio Frequency unit) permet au téléphone d'être relié au réseau et gère la qualité d'émission/réception. Dans le sens de l'émission, il amplifie, module et transmet les données vers la station de base la plus proche. Dans le sens de la réception, il filtre les signaux utiles, les démodule et les transmet vers le composant approprié.

Microphone. Le microphone est un dispositif de conversion des ondes sonores en impulsions électriques. C'est donc un capteur analogique.

Haut-parleur. Le haut-parleur est un système électromécanique destiné à produire des sons à partir d'un signal électrique.

Ecran. L'écran affiche les informations sur les numéros d'appel, la couverture du réseau, l'état de la batterie, etc. Les écrans sont de plus en plus sophistiqués afin de faciliter l'accès à Internet et aux services multimédia. Ils sont souvent en couleur, plus larges, avec une meilleure définition des couleurs (jusqu'à 64 000 couleurs) et une haute résolution (QVGA : Quarter Video Graphics Array soit 320 × 240).

La technologie LCD (Liquid Crystal Display) est utilisée pour la réalisation des écrans car elle consomme peu d'énergie.

Clavier. La plupart des téléphones adoptent un clavier adapté à la composition de numéro et composé de douze touches alphanumériques et des touches de navigation et de fonction. Cependant, certain téléphones, souvent haut de gamme, incluent un clavier QWERTY ou AZERTY qui est une version minimisée des claviers d'ordinateurs traditionnels ; d'autres smartphones utilise même un stylet tactile.

Batterie. La batterie est la seule source d'énergie pour les composants du téléphone mobile. Il en existe deux sortes : les batteries NiMH (Nickel Metal Hydride) et les batteries au Lithium. Les batteries NiMH comprennent un boîtier en carbure, elles sont donc plus lourdes que celles au Lithium. Il existe deux types de batteries Lithium: les batteries au Lithium-Ion qui bénéficient d'une grande autonomie et les batteries Lithium-polymère qui sont très fines et très légères et sont utilisées dans les téléphones haut de gamme.

Connectivité locale. Outre la transmission de la voix et des données sur un réseau mobile, le téléphone peut communiquer avec d'autres équipements fixes ou mobiles ou bien être utilisé comme modem pour un ordinateur ou un PDA non connecté.

Un câble série, souvent de type USB (Universal Serial Bus), peut être utilisé pour relier le téléphone à un ordinateur. Cette technique est très efficace dans la mesure où elle permet d'établir une connexion stable, mais il est à noter que chaque type de téléphone possède un câble spécifique.

La technologie infrarouge permet, quand à elle, de connecter le téléphone à tout équipement compatible IrDA (Infrared Data Association). L'inconvénient de cette méthode est que les deux équipements doivent être en face l'un de l'autre comme une télécommande.

Grâce à la norme Bluetooth, il est possible d'établir un réseau domotique sans fil (WPAN : Wireless Personnal Area Network) et le téléphone peut communiquer avec tout appareil appartenant à ce réseau.

Carte SIM (Subscriber Identity Module). Le téléphone ne peut accéder aux services du réseau mobile sans la carte SIM. En effet, dès son insertion dans l'appareil, elle est lue par le lecteur de carte SIM du téléphone et les informations d'abonnement contenues dessus sont transmises à la station de base via le module radio.

La majorité des cartes SIM exploitent un processeur de 8 ou 16 bits avec 64 Ko de mémoire embarquée. L'utilisateur y stocke ses contacts ou ses SMS. L'opérateur mobile y place ses paramètres réseaux ou des applications. De plus, la carte SIM est au centre de profondes évolutions; des cartes munies d'un processeur 32 bits et d'une mémoire de 256 Ko sont déjà commercialisées et des cartes de capacité bien plus importante (jusqu'à 4Go) sont en développement. Avec de telles capacités, la carte peut être utilisée pour héberger du contenu et des applications multimédias (photos, vidéos, MMS, serveur de blogs, etc.).

Le système d'exploitation est généralement un système ouvert : Java Card et dans de rares cas un système propriétaire. Mais de nouveaux systèmes font leur apparition : Multos de Master Card et Windows Card de Microsoft.

La figure III.5 illustre l'architecture interne d'un terminal mobile 3G.

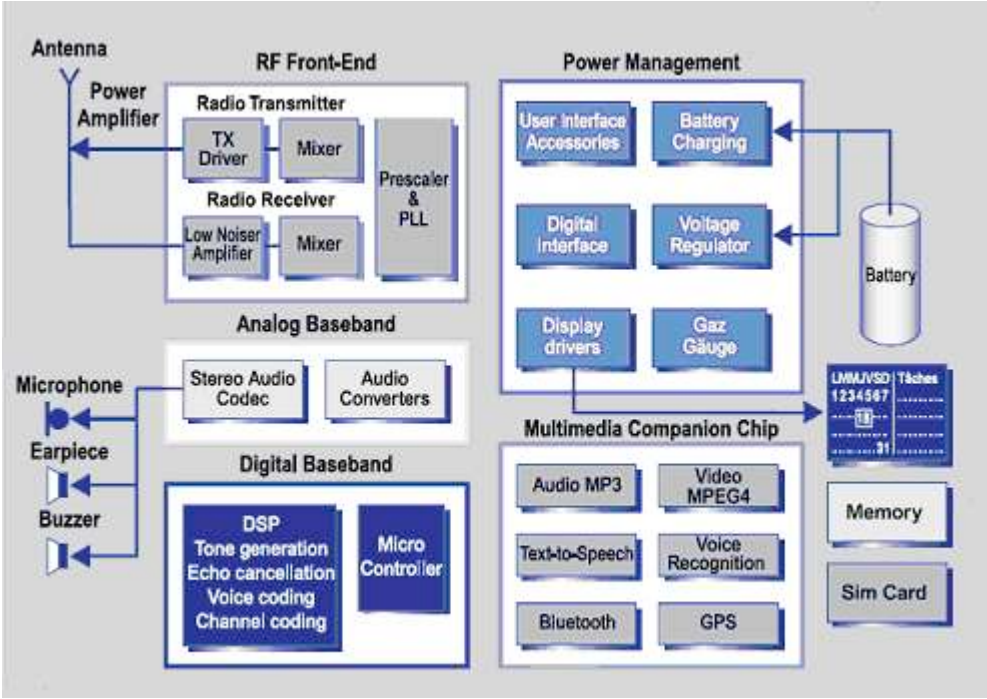


Figure III.5 Architecture interne d'un téléphone mobile 3G [THE01]

5.2 Plateforme logicielle

Après avoir présenté les composants essentiels d'un téléphone mobile, nous abordons dans cette section le logiciel nécessaire à la gestion et à la coordination de ces composants, le système d'exploitation (OS : Operating System). Nous présentons ensuite les principales plateformes de développement pour les appareils mobiles.

5.2.1 Système d'exploitation

Tout comme dans le cas des ordinateurs personnels, le choix du système d'exploitation est très important car il détermine quelles autres technologies ou logiciels peuvent coexister sur l'appareil. Certains systèmes d'exploitation sont des systèmes exclusifs, souvent détenus par des constructeurs d'appareils téléphoniques, alors que d'autres font l'objet d'une licence d'utilisation par plusieurs constructeurs. La majorité des appareils vendus aujourd'hui sont équipés de systèmes d'exploitation propriétaires développés par le constructeur de l'appareil

téléphonique; cependant, le pourcentage d'appareils équipés de systèmes d'exploitation sous licence est en évolution constante.

Ces systèmes fournissent des interfaces de programmation d'applications (API: Application Programming Interface) et des outils de développement (SDK : Software Development Kit) adaptés. Afin de faciliter les tests et le déboguage les applications sur les appareils réels, ses plateformes sont accompagnées d'un émulateur qui permet de déployer et de tester le code des applications sans avoir physiquement recours au téléphone mobile.

Dans ce qui suit, nous décrivons les différents systèmes d'exploitation tout en précisant leurs caractéristiques et les outils de développement disponibles sur ces plateformes.

Symbian OS. Symbian OS a été développé par Symbian, un consortium regroupant les plus grands constructeurs d'appareils mobiles : Nokia, Ericsson, Sony Ericsson et Samsung. Basé sur le système EPOC, Symbian OS définit plusieurs modèles d'interface (UI : User Interface) pour différents types d'appareil ; par exemple, Crystal pour les téléphones classiques et les série 60 pour les téléphones intelligents (smartphones). Symbian OS est un système d'exploitation en temps réel supportant le multithreading. Il permet la gestion de la mémoire, l'ordonnancement des processus et des threads, la communication interprocessus, la gestion des ressources propres au processus ou au threads et le traitement des erreurs. Il inclut également des services de messagerie tel que le SMS (Short Message Service), des services multimédia et des protocoles de transmission de données. La dernière version de Symbian OS et la 9.

Etant implémenté en C++, Symbian OS supporte naturellement le développement d'applications C++. Il gère également les applications Java mobiles (J2ME : Java 2 Micro Edition). Les principaux constructeurs d'appareils utilisant Symbian OS (Nokia, Motorola) fournissent aussi des SDK spécifiques à leurs téléphones

Windows Mobile. Microsoft Windows Mobile inclut un système d'exploitation basé sur Windows CE, des outils de développement et un environnement d'exécution et de programmation : .Net Compact Framework.

Le développement d'applications utilise les outils Visual Studio .Net et les programmeurs peuvent choisir entre C# ou Visual Basic. Aussi, il existe des machines virtuelles Java permettant d'exécuter une application J2ME sous Windows CE. Ainsi, MicroJBlend, une extension de la KVM (Kilobyte Virtual Machine) de Sun, propose nativement ce genre de solution.

Embedded Linux. De plus en plus de constructeurs (Motorola, NEC, PalmSource) adoptent une version de Linux adaptée aux environnements embarqués : Embedded Linux. Ce système supporte le multithreading et le temps réel.

Le développement d'application se fait en C/C++.

Le système Symbian OS tient la place de leader car il équipe 60% des téléphones intelligents contre moins de 20% pour Microsoft Mobile et moins de 10% pour Linux [MAC05]

5.2.2 Plateformes de développement

Java. La plateforme Java est constituée d'un langage de programmation et d'un environnement d'exécution. Elle comprend trois éditions : l'édition entreprise (J2EE : Java 2 Entreprise Edition) dédiée aux serveurs, l'édition standard (J2SE : Java 2 Standard Edition), destinée aux ordinateurs de bureau et l'édition micro (J2ME : Java 2 Micro Edition) adaptée aux appareils à faible ressource tels que les téléphones ou les PDA. Elle inclut en plus une plateforme destinée à la programmation de cartes à puce, Java Card. Cette classification vise à mieux adapter l'environnement de développement à l'appareil cible et à améliorer les performances des applications; en effet, un téléphone portable ne peut pas traiter l'information de la même façon qu'un serveur.

La quasi-totalité des téléphones portables, mêmes les modèles les plus anciens utilisant un système d'exploitation propriétaire, embarquent une machine virtuelle Java. Comme les appareils mobiles possibles sont très hétérogènes, J2ME définit plusieurs spécifications et s'appuie sur une architecture modulaire composée de configurations et de profiles.

Les configurations définissent les caractéristiques de base d'un environnement d'exécution et se composent d'une machine virtuelle et d'un ensemble d'API de base. Deux configurations sont actuellement définies : la CDC (Connected Device Configuration) adaptée aux terminaux relativement puissants comme les PDA ou les smartphones et utilisant une machine virtuelle optimisée appelée CVM (Compact Virtual Machine) ; et la CLDC (Connected Limited Device Configuration) dédiée aux appareils à très faibles capacités comme les téléphones mobiles et utilisant une machine virtuelle allégée KVM.

Les profiles se composent d'un ensemble d'API spécifiques et doivent être associés à une configuration. Sun Microsystems propose deux profiles de référence : le Profile Foundation

destiné à la configuration CDC et le profile MIDP (Mobile Information Device Profile) destiné à la configuration CLDC.

La figure III.1 résume les différentes éditions et les diverses couches de J2ME.

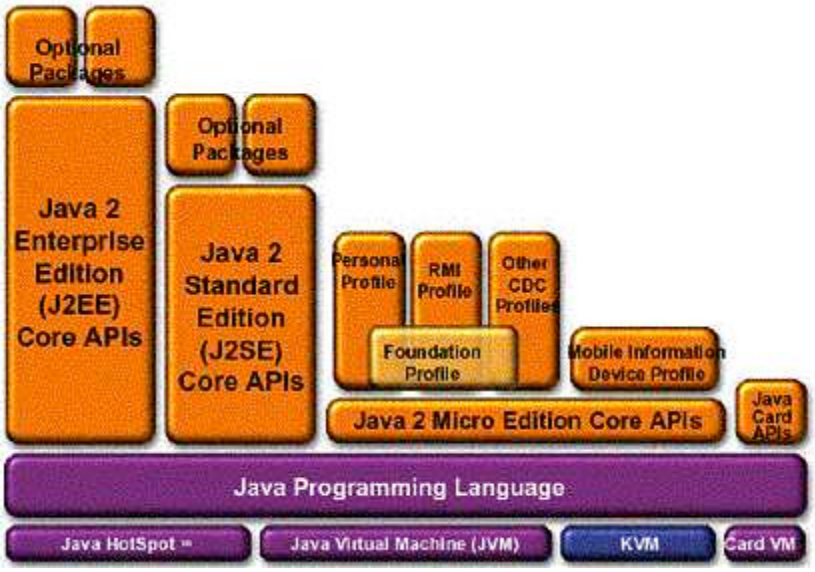


Figure III.6 Plateforme Java

J2ME est un sous-ensemble de la spécification Java 2 Standard Edition, et comprend des ajouts spécifiques aux appareils mobiles. J2ME permet ainsi de gérer l'interface utilisateur, les mécanismes de stockage permanent, l'accès au réseau, le multithreading et les applications audio et vidéo. De plus, des paquetages additionnels peuvent être ajoutés mais ne garantissent pas la portabilité de l'application. Néanmoins, J2ME ne supporte pas les chargeurs de classes, les appels réflexifs et la finalisation.

Autres plateformes. Si la plateforme Java comprend un environnement d'exécution autonome et indépendant du système d'exploitation, la plupart des autres plateformes sont associées à un système d'exploitation propriétaire, comprenant un environnement d'exécution natif et fournissant des fonctionnalités additionnelles, telles que des API pour les appareils photos ou la messagerie.

Outre les kits de développement logiciel (SDK) propriétaires spécifique à chaque constructeur, d'autres technologies essayent d'adopter des approches indépendantes du matériel ; par exemple, le Microsoft .NET Compact Framework est un environnement permettant d'exécuter des programmes sur divers périphériques informatiques à ressources

limitées : assistants numériques personnels (PDA, Personal Data Assistant) tels que le Pocket PC, téléphones mobiles, décodeurs, périphériques informatiques automobiles et périphériques personnalisés. Cependant, ces périphériques doivent intégrer un système d'exploitation Windows CE .NET.

6 Limitations des environnements mobiles

Après avoir exposé les concepts et les caractéristiques des réseaux mobiles et étudié les propriétés des téléphones mobiles, nous pouvons déduire leurs limitations.

6.1 Contraintes liées au réseau mobile

Bande passante. La bande passante dans les réseaux mobiles est souvent inférieure à celle des réseaux fixes haut débit. Elle est mesurée en dizaines de kilobits par seconde pour les réseaux 2G tel que GPRS. Afin d'optimiser son utilisation, l'application doit minimiser la taille des messages et leur fréquence d'envoi.

Temps de latence. Le temps de latence est le temps entre l'envoi de la requête et la réception de la réponse. Le temps de latence et ses variations (temps de réponse, temps d'attente) sont généralement plus grands dans les réseaux mobiles (mesurés en seconde) que dans les réseaux fixes (mesurés en milliseconde, voire en microseconde).

Le tableau suivant résume quelques propriétés des réseaux GSM et GPRS :

	Bande passante (kilobit/seconde)	Temps initial pour se connecter (seconde)	Temps de latence (seconde)
GSM DATA	9,6 – 14,4	5 – 10	2 – 3
HSCSD	10 – 50	5 – 10	2 – 3
GPRS	30 – 70	/	3 – 6

Comprendre le flux des messages de l'application et ses attributs permet de concevoir des applications plus performantes et s'adaptant mieux à l'environnement de développement.

Hétérogénéité des réseaux. Il existe une multitude de réseaux mobiles et sans fil et l'application doit s'adapter aux spécificités de chaque réseau ; d'où la nécessité d'utiliser des protocoles et des plateformes à haut niveau d'abstraction.

Déconnexion. L'utilisateur peut se déconnecter de sa station à la suite d'une panne du réseau ou parce qu'il a quitté la zone de couverture. Les déconnexions peuvent totalement paralyser une application ayant besoin d'un accès distant. Des techniques de préchargement peuvent anticiper les déconnexions et permettre à l'utilisateur de travailler en mode déconnecté ; une application de base de données pourrait, par exemple, précharger la partie de la base de données sur laquelle travaille l'utilisateur.

6.2 Contraintes liées au téléphone mobile

Plusieurs restrictions sont imposées aux applications distribuées mobiles par le téléphone lui-même ; certaines sont liées au matériel, d'autres à la plateforme logicielle.

Ressources limitées. La puissance de traitement et la capacité de stockage sont en constante évolution mais sont toujours très inférieures à celles disponibles sur des appareils fixes. Nous devons donc optimiser les calculs, la consommation de la mémoire et la taille de l'application.

Interface restreinte. Même si les téléphones sont de plus en plus ergonomiques, ils restent néanmoins moins confortables qu'un appareil fixe avec des résolutions moindres, un nombre de couleurs minime, des écrans aux dimensions réduites et des claviers destinées à la composition de numéros. Ces paramètres doivent être pris en compte lors de la conception et nous devons limiter les données à introduire ou à afficher.

Autonomie limitée. La durée de vie de la batterie est un paramètre limitant incontournable. Actuellement, les mécanismes d'économie de la batterie sont principalement matériels et n'interviennent que lorsque la batterie est très faible ou lorsque le terminal n'est pas utilisé.

Hétérogénéité des plateformes. La diversité des plateformes mobiles est sans comparaison avec ce que nous connaissons dans le monde des ordinateurs de bureau ; la plupart des constructeurs adoptent un système d'exploitation propriétaire et incompatible avec ceux des autres constructeurs. Il est donc extrêmement difficile de fournir une application adaptée à tous les modèles d'appareils, d'où l'intérêt d'une solution la plus indépendante possible de la plateforme.

Langage de programmation limité. Un ordinateur de bureau supporte différents langages de programmation, chacun meilleur que d'autres pour la résolution d'un certain

type de tâches. Sur un équipement mobile, quelques langages seulement sont pris en compte, tels que Java, C++ natif et dans de rares cas Visual Basic ou Visual C#.

Interruption. Lors de la conception, nous devons prévoir des mécanismes qui mettent en attente l'application si une interruption se produit ; par exemple, l'application ne doit pas se bloquer si l'utilisateur reçoit un appel ou un message.

7 Synthèse

L'étude des caractéristiques et des propriétés des réseaux et des téléphones mobiles nous a permis de déduire les limitations de ces environnements. Ainsi, le développement d'applications mobiles de qualité pose de nouveaux problèmes auxquels il faut répondre pour atteindre les performances escomptées.

Si la plateforme CTH* utilise des réseaux locaux dont la largeur de bande est de l'ordre d'une centaines de mégabits, il faudra se contenter de quelques dizaines de kilobits dans un réseau mobile.

De plus, les composants CTH* sont des ordinateurs fixes aux performances beaucoup plus grandes que celles des téléphones mobiles.

Enfin, l'hétérogénéité des plateformes est bien plus importante en environnement mobile qu'en environnement fixe ; d'où l'intérêt d'adopter une solution indépendante de la plateforme et à haut niveau d'abstraction.

Toute la difficulté de concevoir des applications mobiles réside dans cette complexité inhérente aux réseaux mobiles ; nous pouvons dire que cela aura un impact important sur la conception interne des composants et le choix de l'architecture de notre application CTH* mobile.

Le prochain chapitre détaille les choix conceptuels concernant l'architecture et le protocole de communication CTH*. Il décrit également les techniques de développement utilisées pour implémenter le prototype CTH*.

Chapitre IV

Conception et réalisation

La diversité des plateformes, l'hétérogénéité des interfaces de communication ainsi que les limitations des ressources et des performances des environnements mobiles soulèvent des défis importants en matière de conception, réalisation et déploiement d'applications mobiles de qualité.

Ce chapitre présente l'architecture générale de la plateforme CTH* gérant des entités fixes et mobiles puis détaille l'architecture de chaque composant. Il décrit ensuite le protocole de communication entre ces composants, les techniques de mise en œuvre du prototype CTH* et l'interface de l'application mobile réalisée.

1 Architectures

Cette section commence par présenter l'architecture générale de l'application en étayant les choix conceptuels de ce modèle, puis décrit plus en détails l'architecture de chaque composant du système.

1.1 Architecture générale

Lors de la conception d'une application dans un environnement local, le choix de la méthode de transport est assez trivial. Nous pouvons choisir la meilleure solution remplissant les besoins de l'application, puisqu'il n'existe généralement pas de pare-feu gênant la connexion à l'intérieur du réseau local. Lorsque l'application doit être accessible de l'extérieur, ou à partir d'un appareil mobile, le nombre de méthodes de transport convenables se trouve considérablement réduit.

Si l'application CTH* existante utilise les sockets UDP (User Datagram Protocol) pour l'échange de messages entre les différentes entités, cette méthode est inapplicable pour un environnement mobile ; la plupart des opérateurs utilisent des pare-feu qui bloquent les connexions par sockets. De plus, rares sont les téléphones mobiles supportant les connexions

par sockets. Nous devons alors choisir un protocole plus adapté aux environnements mobiles.

Parmi les protocoles réseau existant, HTTP (Hypertext Transfer Protocol) se révèle le choix le plus attractif. Grâce aux succès du Web, HTTP est devenu un protocole largement utilisé ; aujourd'hui, pratiquement tout appareil offrant un accès au réseau supporte ce protocole. De plus, HTTP peut facilement naviguer à travers les firewalls. Nous avons donc utilisé le protocole HTTP pour les communications entre le téléphone mobile et les machines du réseau local.

Cependant, il est à préciser qu'en dépit des améliorations apportées au protocole HTTP, il reste un protocole lent. C'est principalement dû aux propriétés du protocole de transmission de données sous-jacent, TCP (Transmission Control Protocol). TCP est un protocole fiable travaillant en mode connecté, ce qui le rend moins rapide que le protocole UDP qui fonctionnent en mode non connecté et qui ne garantit ni la délivrance ni l'ordre des paquets reçus.

Outre la lenteur, les données HTTP ont une faible priorité dans les routeurs, cela signifie que lorsque le réseau est congestionné, le trafic HTTP est considérablement ralenti. Ces limitations du protocole HTTP ont fait que nous avons conservé le protocole UDP pour les échanges de messages dans le réseau local.

Le serveur CTH* doit donc gérer deux mode de communication : le mode UDP pour les messages émanant des clients fixes, des autres serveurs ou du coordinateur et le protocole HTTP pour les requêtes provenant de clients mobiles. Comme l'application existante a été développée en Visual C++ et que ce langage n'offre aucun moyen pour gérer le protocole HTTP côté serveur, nous utilisons un programme intermédiaire (middleware) que nous appelons préserveur. Ce préserveur est en fait une servlet[1] HTTP qui intercepte les requêtes HTTP des clients mobiles, les transmet au format adéquat en utilisant les sockets UDP ; lorsqu'il reçoit les réponses des serveurs, il les transmet au client approprié. Le client perçoit donc le préserveur comme un serveur car il lui envoie des requêtes et en reçoit les réponses et le serveur le perçoit comme un client car il en reçoit les requêtes et lui envoie les réponses. Cette stratégie nous permet de profiter de la souplesse du HTTP en environnement mobile, de la vitesse de UDP en environnement local et de garder la même architecture pour le

[1] Servlet : application Java développée pour les serveurs.

serveur car le préserveur masque les échanges HTTP et permet au serveur de traiter les requêtes émanant des clients fixes et mobiles de la même manière.

La figure IV.1 décrit l'architecture générale de la plateforme CTH* intégrant des clients fixes et mobiles.

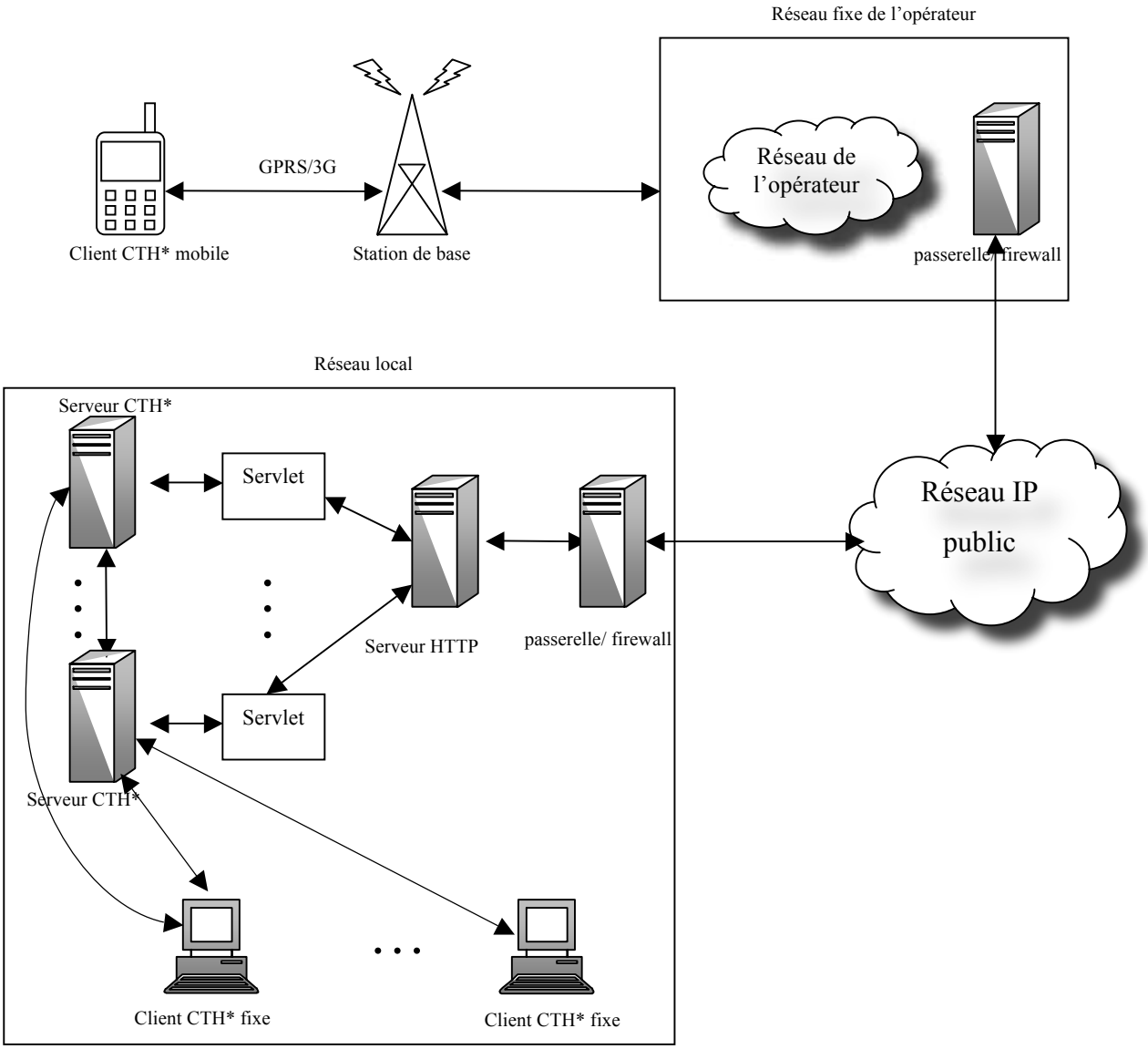


Figure IV.1 Architecture générale de la plateforme CTH*

Nous décrivons d'abord les éléments existants, à savoir le client fixe, le serveur et le coordinateur ; nous nous intéressons ensuite aux éléments permettant d'introduire la mobilité, c'est-à-dire le client mobile et le préserveur.

1.2 Architecture du client fixe

Le client est constitué de plusieurs threads synchronisés par des mécanismes de sémaphores et d'exclusion mutuelle et communiquant à l'aide de files de type FIFO (First In First Out).

Le thread AppReceiverThread est toujours à l'écoute des messages UDP provenant des applications locales. A la réception d'une requête, il la décode et la met dans la file Application-Client.

Le thread AppSenderThread récupère un message de la file Client-Serveur, le code en une chaîne de caractères et le transmet à l'application appropriée.

Le thread ServReceiverThread est toujours à l'écoute des réponses des serveurs ou du coordinateur. A la réception d'un message, il le décode et le met dans la file Serveur-Client.

Le thread ServSenderThread défile un message de la file Client-Application, le code et l'envoie au nœud approprié.

Le client contient également un ensemble de threads travailleurs WorkerThread responsables du traitement des requêtes. Un thread récupère une requête à partir de la file Application-Client, s'il peut la traiter localement, il exécute l'opération adéquate et met la réponse dans la file Client-Application ; sinon, il affecte un numéro unique à la requête (*IdReq*), ajoute le couple (IdReq, port de l'application) à la table de correspondance requête/Application et envoie un ou plusieurs messages aux serveurs ou au coordinateur. Lorsque les réponses arrivent, un thread les récupère à partir de la file Serveur-Client, effectue les traitements nécessaires et met le résultat dans la file Client-Application.

La figure V.2 illustre l'architecture du client CTH* fixe.

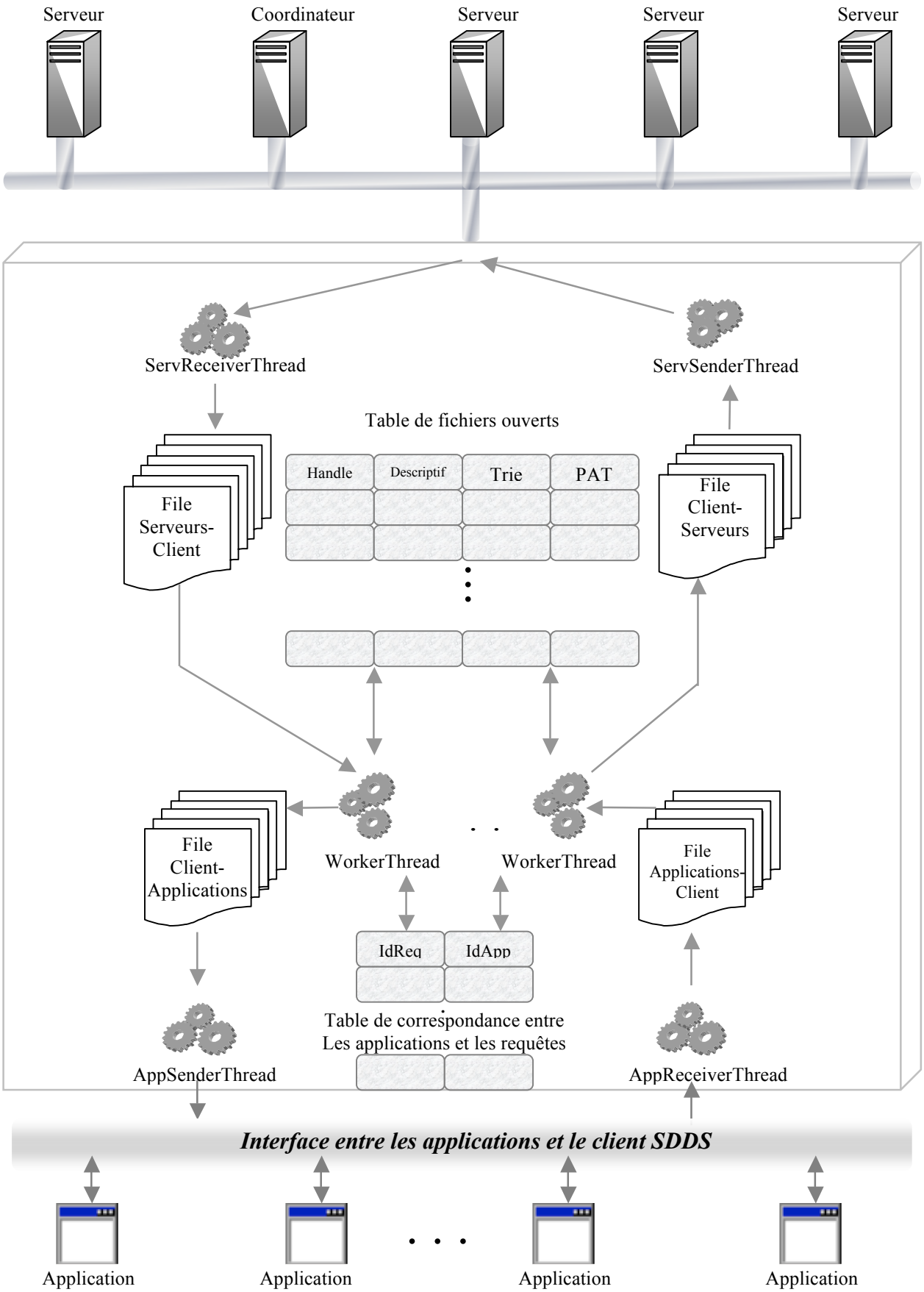


Figure IV.2 Architecture du client CTH* fixe

1.3 Architecture du serveur

Le serveur s'appuie sur une architecture modulaire, il définit un module d'envoi, un module de réception et un module de traitement.

Le module d'envoi est constitué d'un thread SenderThread qui récupère les messages à partir de la file File-Output, les sérialise en les transformant en chaîne de caractères et les transmet en utilisant le protocole UDP.

Le module de réception est lui aussi constitué d'un thread ReceiverThread qui est toujours à l'écoute d'un port UDP spécifique à chaque serveur ; dès qu'un message est reçu, il le désérialise, c'est-à-dire le transforme d'une chaîne de caractère en un format de message précis, et l'enfile dans File-Input.

Le module de traitement est quant à lui constitué d'un ensemble de threads WorkerThread coopérants et synchronisés par des mécanismes d'exclusion mutuelle et de sémaphore. Un WorkerThread défile les requêtes à partir de File-Input, les analyse, les traite puis enfile les réponses dans File-Output.

L'accès aux cases des fichiers CTH* est contrôlé par un gestionnaire d'accès qui utilise des mécanismes d'exclusion mutuelle afin que l'accès aux cases se fasse selon le modèle lecteurs/rédacteur. Ce modèle assure la cohérence des données en autorisant plusieurs consultations en lecture simultanément alors que la mise à jour des données doit être exclusive et bloque toute lecture ou écriture sur la case.

A chaque case est associé un arbre Trie représentant l'image qu'a le serveur du fichier CTH* et une table PAT (Physical Allocation Table) précisant les adresses physiques des serveurs logiques associés à ce fichier CTH* selon la vue du serveur.

La figure IV.5 décrit l'architecture du serveur CTH*.

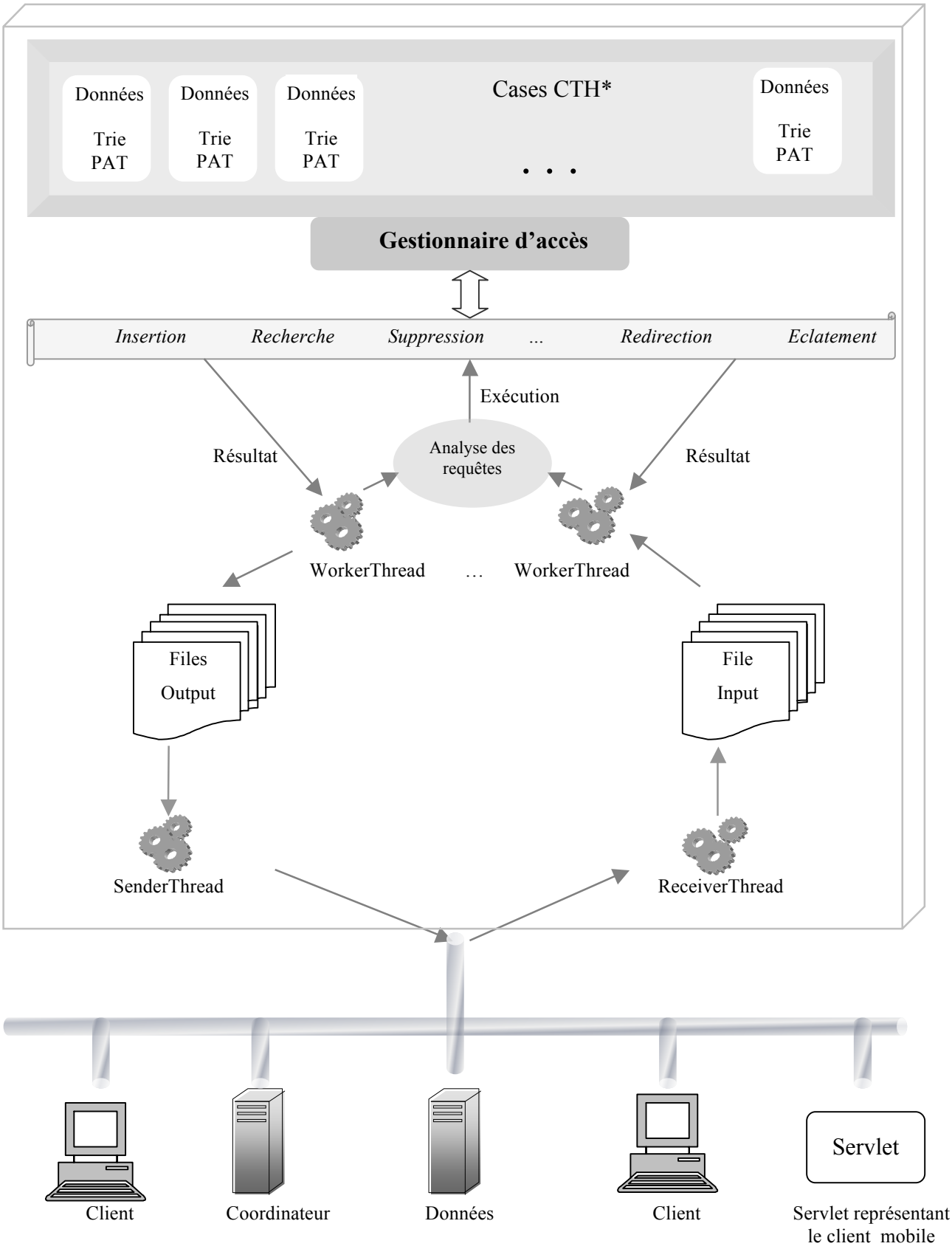


Figure IV.3 Architecture du serveur CTH*

1.4 Architecture de coordinateur

Le coordinateur est basé sur une architecture modulaire et définit un module d'envoi, un module de réception et un module de traitement.

Le module d'envoi comprend un thread SenderThread dont le rôle est de récupérer les messages à partir de la file FileOutput, de les transformer en une chaîne de caractères et de les envoyer en utilisant le protocole UDP.

Le module de réception est formé d'un thread ReceiverThread qui écoute constamment sur un port UDP spécifique au coordinateur les messages envoyés par les serveurs ou les clients. Dès qu'une requête arrive, il la décode et la met dans la file FileInput.

Le module de traitement est constitué d'un ensemble de threads qui récupère les requêtes à partir de la file FileInput, les traite et enfile les réponses dans FileOutput.

Une table d'allocation physique (PAT) recense les différents serveurs ainsi que leurs caractéristiques. Lorsqu'un serveur rejoint le système, il envoie au coordinateur un message en précisant ses caractéristiques, le coordinateur ajoute le serveur à sa table PAT et acquitte son message. La PAT permet au coordinateur de gérer les débordements, car comme il est le seul à connaître le numéro logique du dernier serveur alloué et à connaître la liste des serveurs disponibles, il peut indiquer au serveur débordé un serveur susceptible de contenir une nouvelle case du fichier et lui précise son adresse logique.

Le coordinateur utilise aussi une table de fichier qui contient les caractéristiques des différents fichiers CTH*. Elle permet au coordinateur de gérer les requêtes de création, ouverture, fermeture ou suppression du fichier.

La figure IV.6 présente l'architecture du coordinateur.

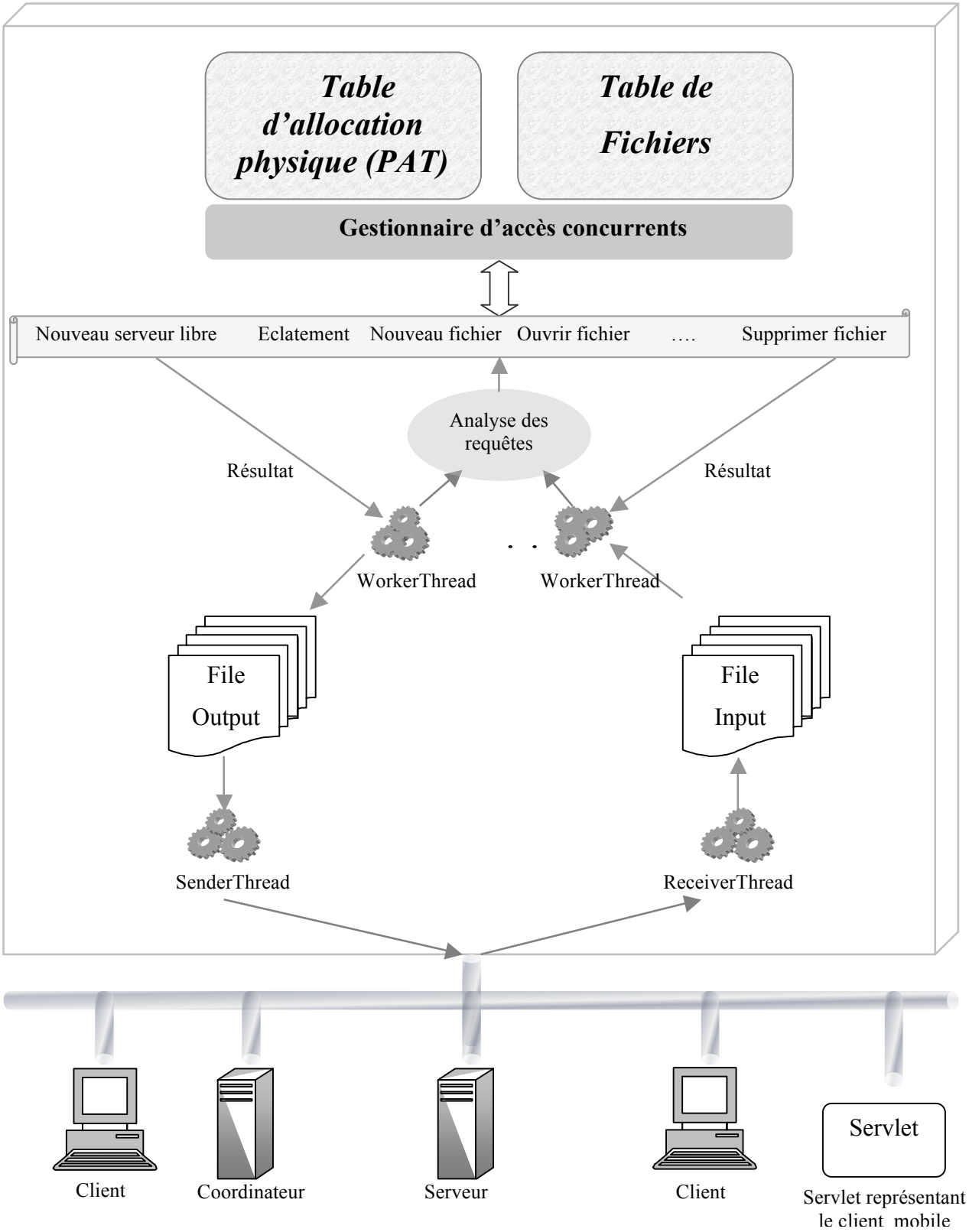


Figure IV.4 Architecture du coordinateur

1.5 Architecture du client mobile

Pour plus d'efficacité et de clarté nous avons séparé la logique de l'application de la gestion de l'interface et du contrôle de la transmission de données. Nous avons donc un thread qui gère l'interface et qui fait appel aux différentes fonctions CTH* disponibles. A l'appel d'une fonction CTH*, un nouveau thread est créé afin de traiter la requête. Ce thread utilise la table des fichiers (FAT) pour traiter la requête ; lorsqu'un accès réseau est requis il prépare le message à envoyer et le met dans la file des requête, il enfile aussi l'adresse du site vers lequel le message doit être transmis dans la file File-URL et un pointeur vers lui-même dans la file File-écouteurs; le thread peut ensuite traiter d'autres appels de fonction jusqu'à ce qu'il reçoive la réponse de sa requête, il finit alors le traitement de l'appel de fonction et retourne le résultat au thread d'interface qui va se charger de l'afficher. Un appareil mobile peut gérer jusqu'à dix connexions HTTP simultanément, il est alors plus judicieux d'utiliser un ensemble de threads pour traiter les connexions HTTP, ces threads récupèrent les messages à transmettre de la file des requêtes, il défile ensuite une adresse de File-URL et l'utilise pour envoyer le message; lorsqu'il reçoivent la réponse HTTP, il défile le pointeur du thread ayant fait la requête à partir de la File-écouteurs et le préviennent de l'arrivée de la réponse.

L'architecture du client mobile est illustrée dans la figure IV.3.

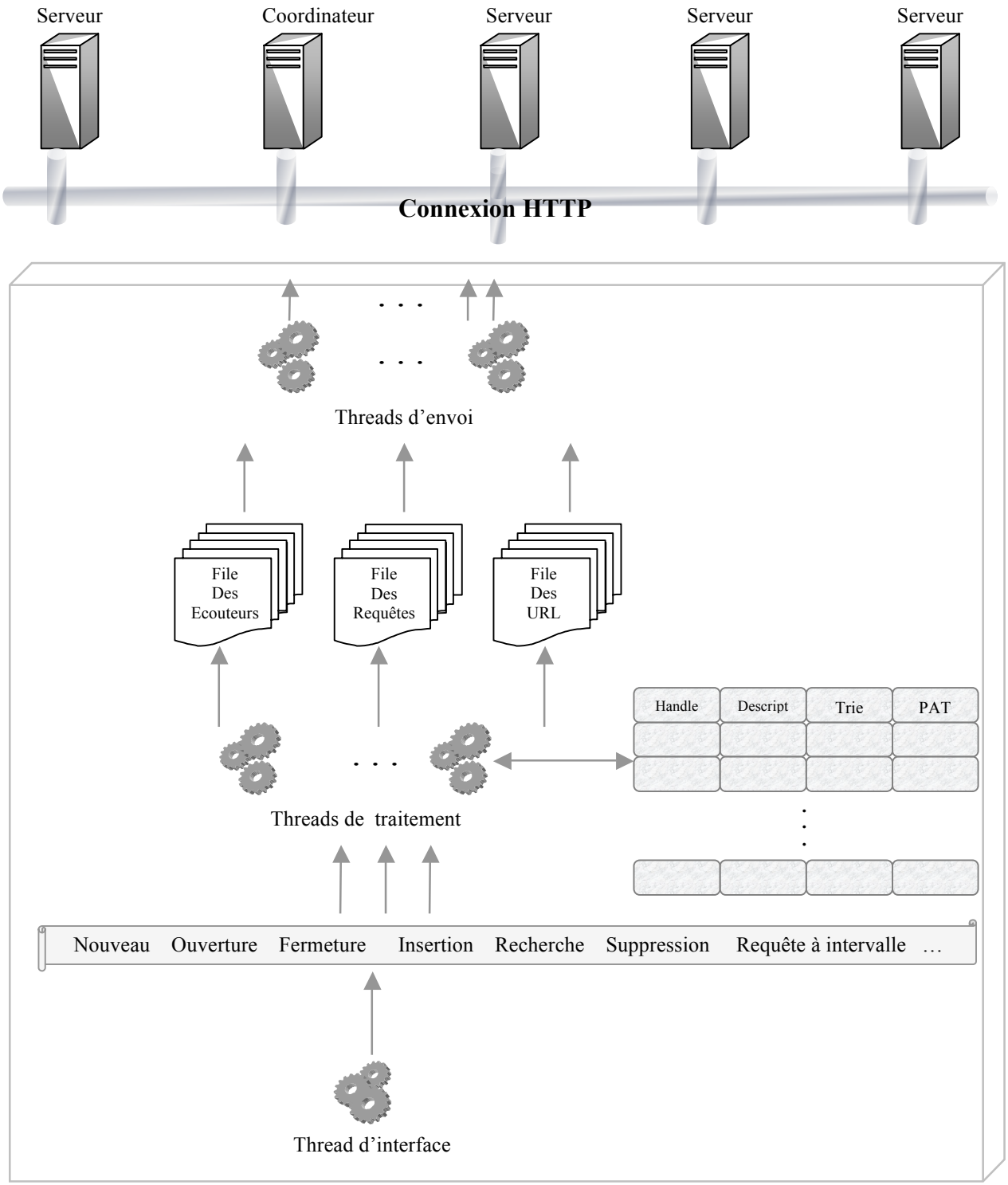


Figure IV.5 Architecture du client mobile

1.6 Architecture du préserveur

Le préserveur joue le rôle d'intermédiaire entre le client mobile et les serveurs CTH*. Lorsque une requête HTTP est reçue, il la transmet au serveur CTH* du même poste. Ce serveur traite la requête et envoie la réponse au préserveur qui la transmet à son tour au client et met fin à la connexion ; s'il y a eu une erreur d'adressage, le serveur transmet alors un IAM au préserveur qui va l'envoyer aussitôt au client mais sans mettre fin à la connexion. Le préserveur ne ferme la connexion que lorsque la requête est entièrement traitée, il continue toutefois à transmettre tous les messages reçus au fur et à mesure de leur arrivée.

Afin d'accélérer le traitement, le préserveur adopte une approche multithread ; il utilise un pool de threads dont la tâche consiste à récupérer les requêtes HTTP de clients mobiles, les transmettre par socket UDP à un serveur du même poste, attendre la réponse de la requête tout en transmettant les IAM reçus au client mobile via HTTP.

La figure IV.4 décrit l'architecture du préserveur.

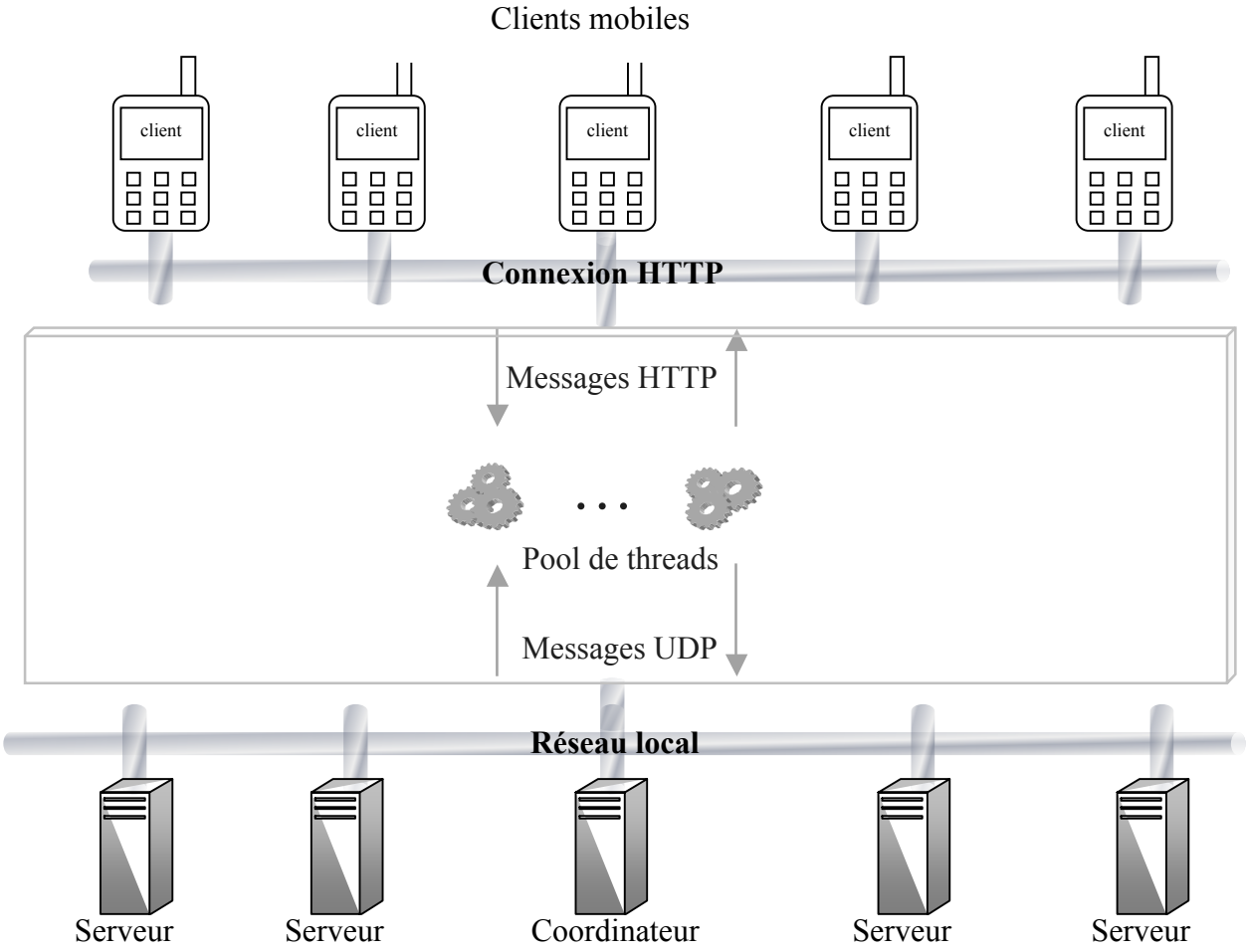


Figure IV.6 Architecture du préserveur

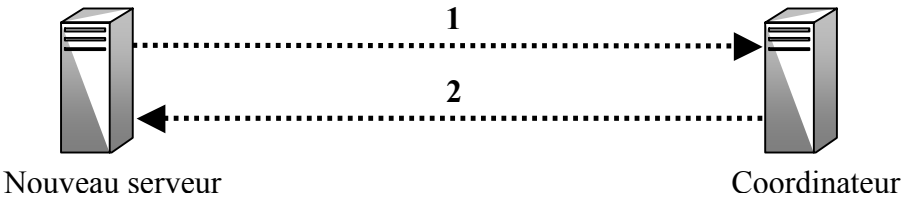
2 Protocole de communication CTH*

CTH* se base sur des architectures distribuées sans partage, les multiordinateurs ; la méthode de communication la plus adaptée à ces modèles est l'échange de messages. Cette section décrit la structure des différents messages échangés entre les composants d'un système CTH* dans un environnement mobile.

Il est à noter que les messages envoyés ou reçus par la servlet ont la même structure que ceux échangés avec le client fixe. Nous n'aborderons donc que les messages échangés entre le client mobile, la servlet, le serveur et le coordinateur. La communication entre les entités fixes est similaire à ce qui a été présenté par SAAD et BOULARIAS [SAA04].

*2.1 Description du protocole CTH**

2.1.1 Entrée en système d'un nouveau serveur



1 : New Free Server

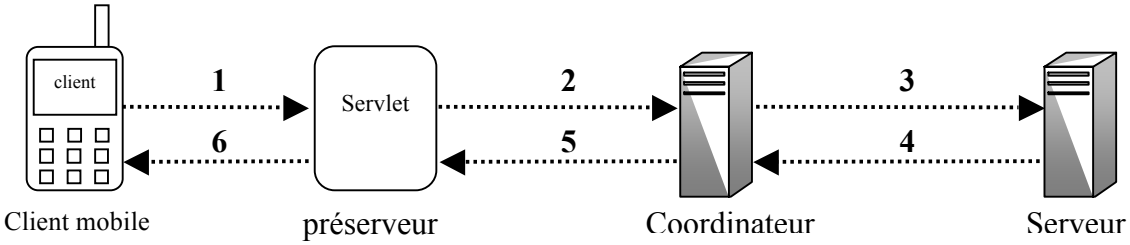
Command	*ServerIpAdr*	*ServerPort*	*FreeSize*

2 : New Free Server Answer

Command	*ErrorCode*

Lorsqu'un nouveau serveur est lancé, il s'inscrit auprès du coordinateur en lui envoyant un message NewFreeServer portant les informations sur son adresse IP (*ServerIpAdr*), son port (*ServerPort*) et sa capacité (*FreeSize*). Le coordinateur répond en envoyant une commande NewFreeServerAnswer.

2.1.2 Création d'un fichier



1 : Mobile New File

Command	*FileName*	*CaseSize*	*NbFields*	*FieldSize*	*KeyIndex*

2 : New File

Command	*IdReq*	*ClientIpAdr*	*FileName*	*CaseSize*	*NbFields*	*FieldSize*	*KeyIndex*

3 : Set First Server

Command	*handle*	*CaseSize*

4 : Set First Server Answer

Command	*handle*	*ErrorCode*

5 : New File Answer

Command	*IdReq*	*ErrorCode*	*handle*	*FirstServerIpAdr*	*FirstServerPort*

ou

Command	*IdReq*	*ErrorCode*

6 : Mobile New File Answer

Command	*ErrorCode*	*handle*	*FirstServerIpAdr*	*FirstServerPort*

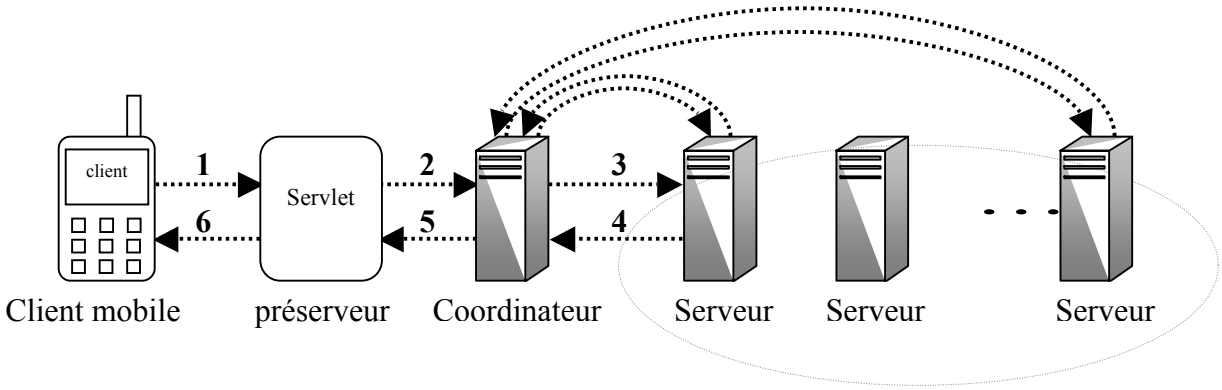
ou

Command	*ErrorCode*

Le client envoie par HTTP un message MobileNewFile en précisant les paramètres du nouveau fichier, à savoir le nom (*FileName*), la taille de la case (*CaseSize*), le nombre de champs (*NbFields*), leurs types (*FieldList*), leurs tailles (*FieldSize*) et l'index de la clé primaire (*KeyIndex*). Le préserveur ajoute *IdReq* qui est un champ non significatif mais

permet d'adopter le même formalisme qu'un client fixe ; il ajoute également *ClientIpAdr* qui représente l'adresse de la machine sur laquelle s'exécute la servlet. Le préserveur code ensuite le message sous forme d'une chaîne de caractères et l'envoie au coordinateur. Lorsque le message NewFile arrive au coordinateur, il consulte sa table de fichier pour vérifier si le fichier existe ; si c'est le cas il envoie immédiatement un message NewFileAnswer en indiquant comme code d'erreur FileExists. Au cas où le fichier n'existe pas, le coordinateur consulte sa table des serveurs pour vérifier si une case peut être allouée au fichier. Si ce n'est le cas, le serveur répond immédiatement par un message ayant un *ErrorCode* égal à NoAvailableFreeSize. S'il existe un serveur pouvant contenir une nouvelle case du fichier, le coordinateur lui envoie un message SetFirstServer et lorsqu'il reçoit l'acquittement correspondant, il répond au préserveur en envoyant un NewFileAnswer en indiquant le numéro associé au fichier (*Handle*), l'adresse (*FirstServerIpAdr*) et le port du premier serveur (*FirstServerPort*). Dès que le préserveur reçoit un message NewFileAnswer, il le décode et envoie la réponse par HTTP au client. Comme la valeur *IdReq* n'est pas significative pour le client, le préserveur ne la lui envoie pas.

2.1.3 Ouverture d'un fichier



1 : Mobile Open File

Command	*FileName*

2 : Open File

Command	*IdReq*	*ClientIpAdr*	*FileName*

3 : Server Open File

Command	*handle*

4 : Server Open File Answer

Command	*ErrorCode*	*handle*	*ServerIpAdr*	*ServerPort*

5 : Open File Answer

Command	*IdReq*	*ErrorCode*	*handle*	*FileName*	*CaseSize*	*NbFields*	*FieldsList*
FieldsSize	*KeyIndex*	*FirstServerIpAdr*	*FirstServerPort*	*FirstServerLogicalAdr*			

ou

Command	*IdReq*	*ErrorCode*

6 : Mobile Open File Answer

Command	*ErrorCode*	*handle*	*FileName*	*CaseSize*	*NbFields*	*FieldsList*	*FieldsSize*
KeyIndex	*FirstServerIpAdr*	*FirstServerPort*	*FirstServerLogicalAdr*				

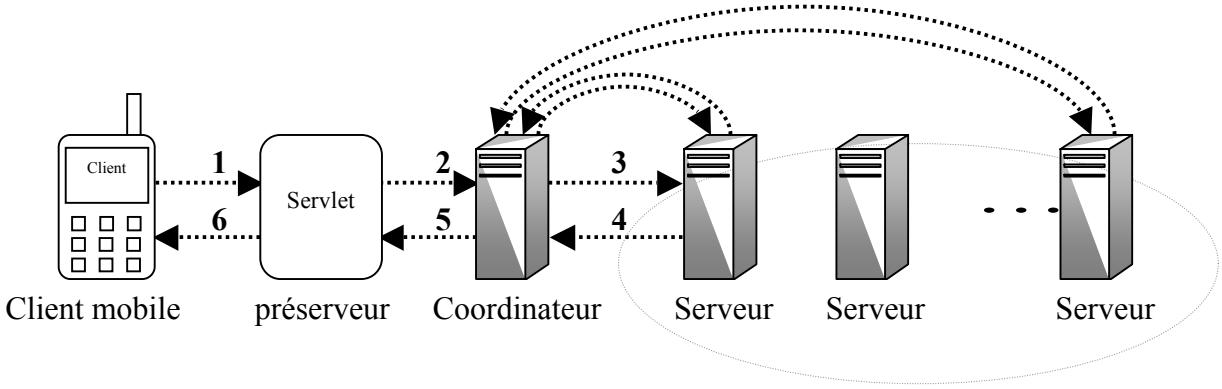
ou

Command	*ErrorCode*

Le client mobile envoie par HTTP un message MobileOpenFile au préserveur en indiquant le nom du fichier (*FileName*). Le préserveur code le message, l'envoie au coordinateur et attend sa réponse. Lorsque le coordinateur reçoit la requête, il vérifie l'existence du fichier en utilisant sa FAT. Si le fichier n'existe pas le coordinateur répond immédiatement en mettant le champ *ErrorCode* à FileNotExists. Si le fichier existe et n'est pas ouvert le coordinateur consulte la structure caractéristique du fichier et envoie un message ServerOpenFile à tous les serveurs contenant une case du fichier. Une fois tous les acquittements reçus ou si le fichier existe et est déjà ouvert, le coordinateur répond en envoyant au préserveur les paramètres du fichier : le numéro correspondant (*Handle*), le nom (*FileName*), la capacité de la case (*CaseSize*), le nombre des champs (*NbFields*), la liste de leurs types (*FieldsList*), la liste de leurs taille (*FieldsSize*), l'index de la clé primaire (*KeyIndex*), l'adresse IP (*FirstServerIpAdr*), le port l'adresse logique (*FirstServerPort*) et l'adresse logique (*FirstServerIpAdr*) du premier serveur. Lorsque le préserveur reçoit un

message OpenFileAnswer, il le décode, l'envoie aussitôt au client et ferme la connexion HTTP.

2.1.4 Fermeture d'un fichier



1 : Mobile Close File

Command	*handle*

2 : Close File

Command	*IdReq*	*ClientIpAdr*	*handle*

3 : Server Close File

Command	*handle*

4 : Server Close File Answer

Command	*ErrorCode*	*handle*	*ServerIpAdr*	*ServerPort*

5 : Close File Answer

Command	*IdReq*	*ErrorCode*	*handle*

ou

Command	*IdReq*	*ErrorCode*

6 : Mobile Close File Answer

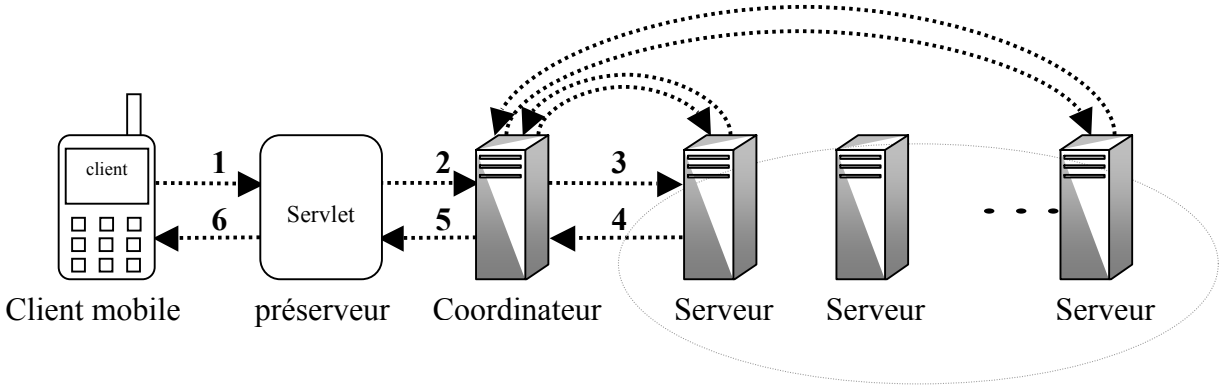
Command	*ErrorCode*	*handle*

ou

Command	*ErrorCode*

Le traitement de la fermeture de fichier est similaire à l'ouverture. Le client mobile envoie par HTTP un message MobileCloseFile au préserveur en indiquant le numéro associé (*Handle*), le préserveur code alors le message, l'envoie au coordinateur et reste à l'écoute jusqu'à ce qu'il reçoive la réponse. Une fois que la requête arrive au coordinateur, il vérifie l'existence du fichier en utilisant sa FAT. Si le fichier n'existe pas, il répond en envoyant un message avec *ErrorCode* mis à FileNotExists. Si le fichier existe et n'est ouvert par aucun autre client, le coordinateur envoie un message ServerCloseFile à tous les serveurs contenant une case du fichier. Une fois tous les acquittements reçus, le coordinateur répond en envoyant au préserveur un message CloseFileAnswer avec un *ErrorCode nul*. Le préserveur décode la réponse, l'envoie au client et ferme la connexion HTTP.

2.1.5 Suppression d'un fichier



1 : Mobile Delete File

Command	*handle*

1 : Delete File

Command	*IdReq*	*ClientIpAdr*	*handle*

3 : Server Delete File

Command	*handle*

4 : Server Delete File Answer

Command	*ErrorCode*	*handle*	*ServerIpAdr*	*ServerPort*

5 : Delete File Answer

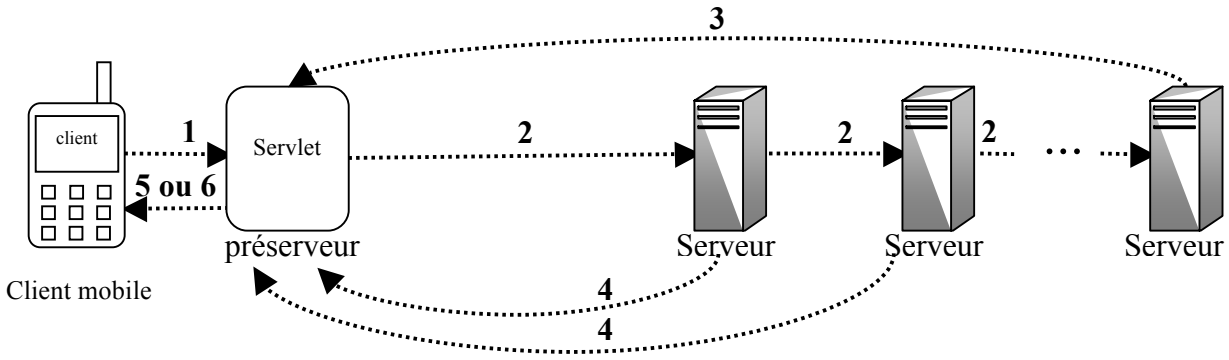
Command	*IdReq*	*ErrorCode*	*handle*

6 : Mobile Delete File Answer

Command	*ErrorCode*	*handle*

Le client mobile envoie par HTTP un message MobileDeleteFile au préserveur en indiquant le *Handle* correspondant, le préserveur code le message, l'envoie au coordinateur et attend la réponse. Lorsque le coordinateur reçoit le message, il consulte sa FAT pour vérifier l'existence du fichier. Si celui-ci n'existe pas, le coordinateur répond aussitôt en envoyant un DeleteFileAnswer avec un *ErrorCode* de valeur FileNotExists. Sinon, il vérifie si le fichier est ouvert par d'autres clients. Si c'est le cas, il répond par un *ErrorCode* à OpenByOthers. Sinon, le coordinateur envoie un message ServerDeleteFile à tous les serveurs contenant une case du fichier. Une fois tous les acquittements reçus, le coordinateur répond en envoyant au préserveur un message DeleteFileAnswer. Le préserveur décode la réponse, l'envoie au client et ferme la connexion HTTP.

2.1.6 Insertion d'un enregistrement



1 : Mobile Insert Data

Command	*handle*	*Key*	*Record*	*Cle_max*	*Port*

2 : Insert Data

Command	*IdReq*	*ClientIpAdr*	*handle*	*Key*	*Record*	*Cle_max*

3: Insert Data Answer

Command	*IdReq*	*ErrorCode*

4 : Image Ajustement

Command	*handle*	*CommunDigitNumber*	*LastServer*	*ImageAjustementSize*
ImageAjustementTrie	*NbServer*	*ListServerLogical*	*ListServerPort*	*ListServerIp*
ServerIpRedirect	*ServerPortRedirect*			

5 : Mobile Insert Data Answer

Command	*ErrorCode*

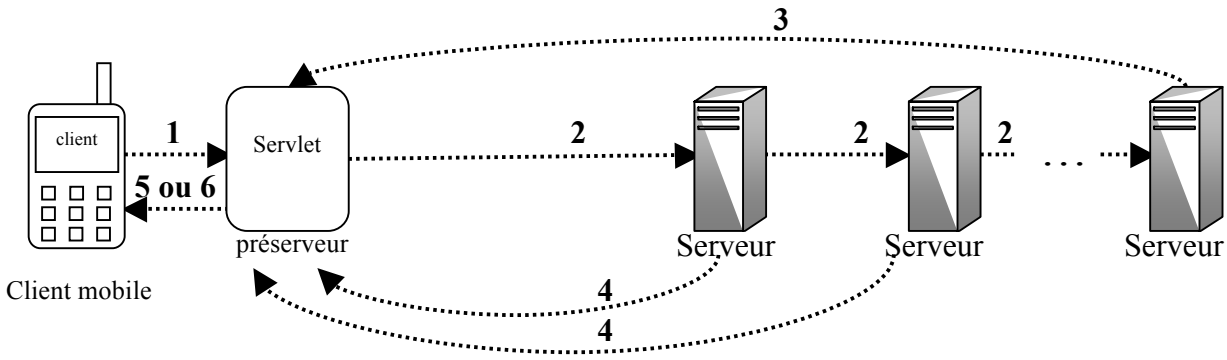
6 : Mobile Image Ajustement

Command	*handle*	*CommunDigitNumber*	*LastServer*	*ImageAjustementSize*
ImageAjustementTrie	*NbServer*	*ListServerLogical*	*ListServerPort*	*ListServerIp*

Le client envoie la requête MobileInsertData en précisant l'enregistrement (*Record*), sa clé (Key) et la clé maximale selon son image du fichier (*Cle_max*) et en indiquant au préserveur le port (*Port*) vers lequel le message doit être envoyé car ce dernier ne détient aucune information sur le fichier CTH* et ne connaît donc pas le port d'écoute du serveur. Le préserveur extrait le numéro de port et code le reste du message en lui ajoutant *IdReq* et *ClientIpAdr* et en le transformant en une chaîne de caractères, il utilise le message codé ainsi que le numéro de port pour créer un datagram qu'il envoie au serveur. Le serveur traite la requête : s'il y a une erreur d'adressage, il envoie un IAM vers le préserveur qui l'envoie aussitôt au client. S'il n'y a pas d'erreur d'adressage, le serveur envoie InsertDataAnswer avec le code d'erreur adéquat (0 si pas d'erreur ou KeyExists si la clé existe déjà). Dès que le préserveur reçoit la réponse, il essaye de restituer l'itinéraire de la requête pour vérifier si tous les IAM ont été reçus ; dans ce cas, la réponse est envoyée et la connexion fermée.

Le message d'ajustement contient le *Handle* du fichier, et les éléments qui permettront au client de mettre à jour son arbre : le nombre de digits communs (*CommonDigitNumber*), le serveur N' (*LastServer*), la partie de l'arbre à remplacer (*ImageAjustementTrie*) ainsi que la liste des numéros logiques *(ListServerLogical*), des ports (*ListServerPort*) et des adresses IP (*ListServerIp*) des serveurs que le client doit ajouter à sa table. Le message d'ajustement contient également l'adresse IP (*ServerIpRedirect*) et le numéro de port (*ServerPortRedirect*) du serveur vers lequel la requête a été redirigée, ce qui permet au préserveur de reconstituer l'itinéraire des messages et de savoir quand mettre fin à la connexion mais n'est d'aucune utilité pour le client mobile.

2.1.7 Modification d'un enregistrement



1 : Mobile Update Data

Command	*handle*	*Key*	*Record*	*Cle_max*	*Port*

2 : Update Data

Command	*IdReq*	*ClientIpAdr*	*handle*	*Key*	*Record*	*Cle_max*

3: Update Data Answer

Command	*IdReq*	*ErrorCode*

4 : Image Ajustement

Command	*handle*	*CommunDigitNumber*	*LastServer*	*ImageAjustementSize*
ImageAjustementTrie	*NbServer*	*ListServerLogical*	*ListServerPort*	*ListServerIp*
ServerIpRedirect	*ServerPortRedirect*			

5: Mobile Update Data Answer

Command	*ErrorCode*

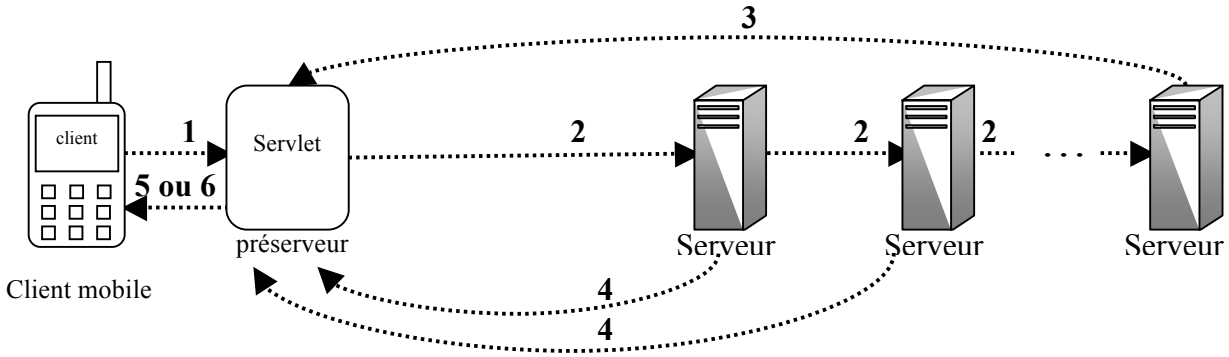
6 : Mobile Image Ajustement

Command	*handle*	*CommunDigitNumber*	*LastServer*	*ImageAjustementSize*
ImageAjustementTrie	*NbServer*	*ListServerLogical*	*ListServerPort*	*ListServerIp*

La modification de l'enregistrement est similaire à l'insertion. Le client envoie la requête MobileUpdateData en indiquant au préserveur le port vers lequel le message doit être envoyé. Le préserveur extrait le numéro de port et code le reste du message, il utilise le résultat ainsi que le numéro de port pour créer un datagram qu'il envoie au serveur. Le serveur traite la requête : s'il y a une erreur d'adressage, il envoie un IAM vers le préserveur qui l'envoie aussitôt au client. Le préserveur vérifie si c'est le dernier message à envoyer. Si c'est le cas, il ferme la connexion. Sinon, il laisse la connexion ouverte et reste à l'écoute.

S'il n'y a pas d'erreur d'adressage, le serveur envoie UpdateDataAnswer avec le code d'erreur adéquat (0 si pas d'erreur ou KeyNotFound si la clé n'existe pas). Dès que le préserveur reçoit la réponse, il essaye de restituer l'itinéraire de la requête pour vérifier si tous les IAM ont été reçus. Si c'est le cas, il transmet la réponse et ferme la connexion.

2.1.8 Recherche d'un enregistrement



1 : Mobile Search Data

Command	*handle*	*Key*	*Cle_max*	*Port*

2 : Search Data

Command	*IdReq*	*ClientIpAdr*	*handle*	*Key*	*Cle_max*

3 : Search Data Answer

Command	*IdReq*	*ErrorCode*	*Record*

ou

Command	*IdReq*	*ErrorCode*

4 : Image Ajustement

Command	*handle*	*CommunDigitNumber*	*LastServer*	*ImageAjustementSize*
ImageAjustementTrie	*NbServer*	*ListServerLogical*	*ListServerPort*	*ListServerIp*
ServerIpRedirect	*ServerPortRedirect*			

5 : Mobile Search Data Answer

Command	*ErrorCode*	*Record*

ou

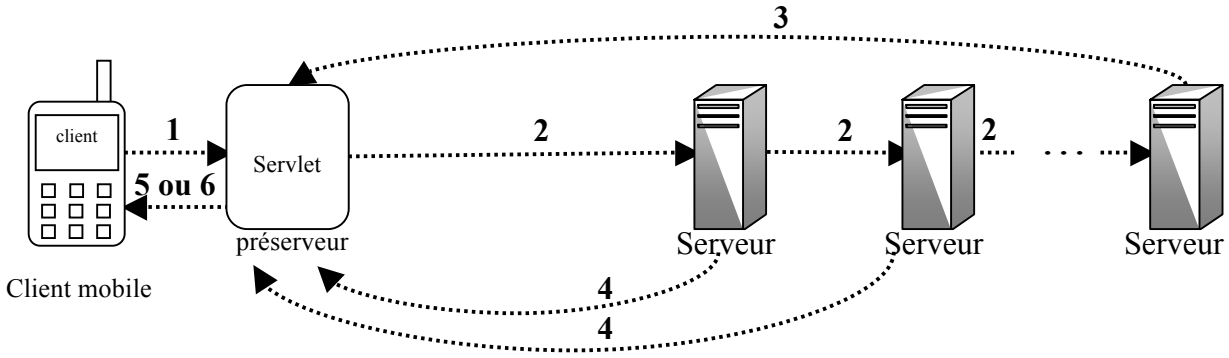
Command	*ErrorCode*

6 : Mobile Image Ajustement

Command	*handle*	*CommunDigitNumber*	*LastServer*	*ImageAjustementSize*
ImageAjustementTrie	*NbServer*	*ListServerLogical*	*ListServerPort*	*ListServerIp*

Pour supprimer une clé, le client adresse une requête MobileSearchData au préserveur en lui indiquant le port vers lequel le message doit être envoyé. Le préserveur code la requête et l'envoie au serveur approprié. Le serveur utilise *Cle_max* pour savoir si le client a commis une erreur d'adressage, si c'est la cas, il envoie un IAM vers le préserveur qui l'envoie aussitôt au client. S'il n'y a pas d'erreur d'adressage, le serveur envoie SearchDataAnswer avec le code d'erreur adéquat (0 si pas d'erreur ou KeyNotFound si la clé n'existe pas). Dès que le préserveur reçoit la réponse, il essaye de restituer l'itinéraire de la requête pour vérifier si tous les messages ont été reçus, il transmet alors la réponse et met fin à la connexion.

2.1.9 Suppression d'un enregistrement



1 : Mobile Delete Data

Command	*handle*	*Key*	*Cle_max*	*Port*

2 : Delete Data

Command	*IdReq*	*ClientIpAdr*	*handle*	*Key*	*Cle_max*

3 : Delete Data Answer

Command	*IdReq*	*ErrorCode*

5 : Mobile Delete Data Answer

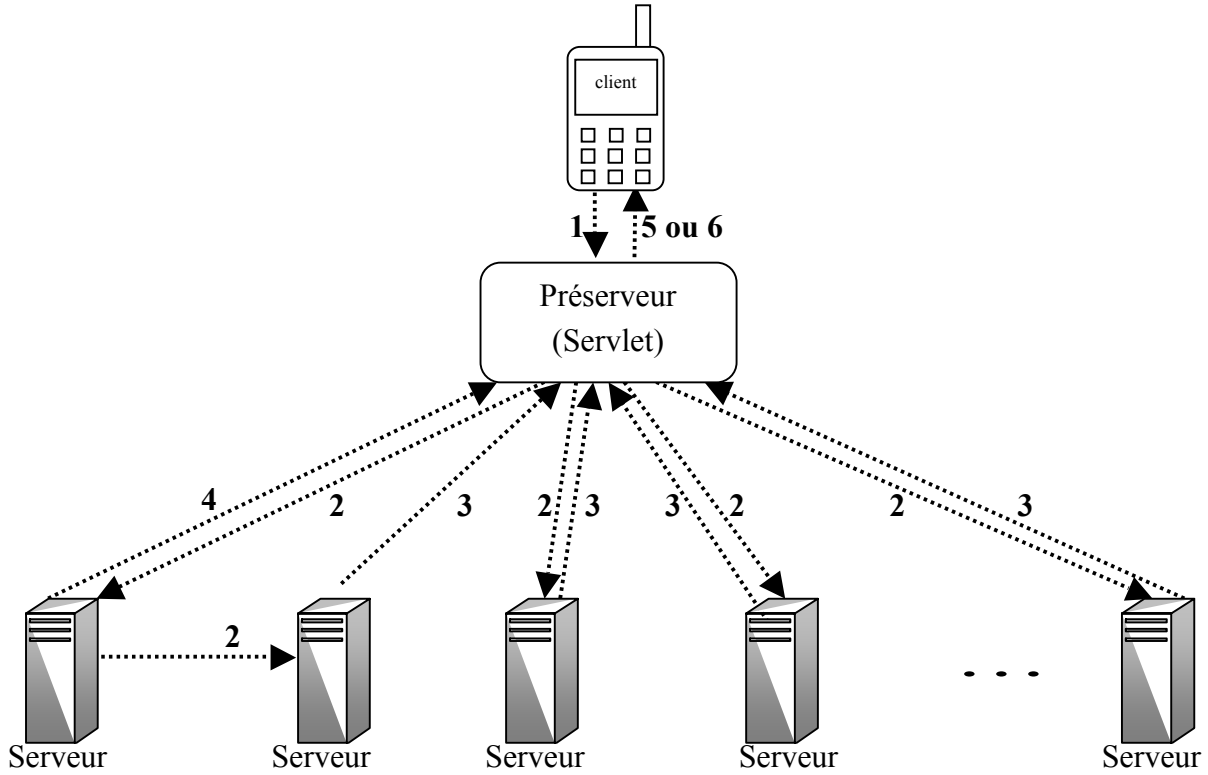
Command	*ErrorCode*

6 : Mobile Image Ajustement

Command	*handle*	*CommunDigitNumber*	*LastServer*	*ImageAjustementSize*
ImageAjustementTrie	*NbServer*	*ListServerLogical*	*ListServerPort*	*ListServerIp*

Le client envoie la requête DeleteData en désignant le numéro du fichier (*Handle*), la clé (*Key*) de l'enregistrement à supprimer, la clé maximale du serveur vers lequel la requête doit être envoyée (*Cle_max*) et son port (*Port*). Le préserveur extrait le numéro de port, code le message et le transmet au serveur adéquat. Le serveur traite la requête : s'il y a une erreur d'adressage, il envoie un IAM vers le préserveur qui l'envoie aussitôt au client. S'il n'y a pas d'erreur d'adressage, le serveur envoie DeleteDataAnswer avec le code d'erreur adéquat (0 si pas d'erreur ou KeyNotExists si la clé n'a pas été trouvée). Une fois que tous les messages ont été reçus, le préserveur ferme la connexion HTTP.

2.1.10 Requête à intervalle



1 : Mobile Range Data Search

Command	*handle*	*MinKey*	*MaxKey*	*Cle_max*	*Port*

2 : Range Data Search

Command	*IdReq*	*ClientIpAdr*	*handle*	*MinKey*	*MaxKey*	*Cle_max*

3 : Range Data Search Answer

Command	*IdRe*	*ErrorCode*	*RecordsList*	*MinKey*	*MaxKey*

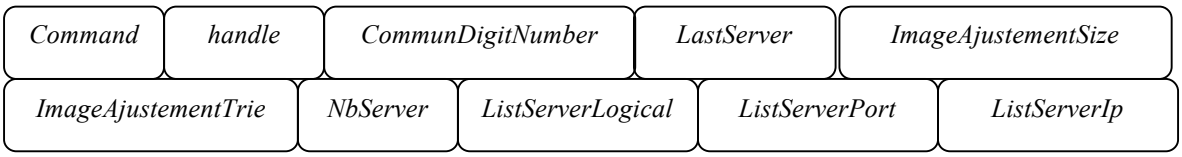
4 : Image Ajustement

Command	*handle*	*CommunDigitNumber*	*LastServer*	*ImageAjustementSize*
ImageAjustementTrie	*NbServer*	*ListServerLogical*	*ListServerPort*	*ListServerIp*
ServerIpRedirectList	*ServerPortRedirectList*			

5 : Range Data Search Answer

Command	*RecordsList*	*MinKey*	*MaxKey*

6 : Mobile Image Ajustement



Le client envoie par HTTP un message RangeDataSearch en précisant le numéro du fichier (*Handle*), la clé minimale de l'intervalle de recherche (*MinKey*), et sa clé maximale (MaxKey) ainsi que la clé maximale du serveur adressé (*Cle_max*) et son port. En recevant ce message, le préserveur le code, utilise le résultat pour construire un datagram et envoie la requête vers le port approprié. Le serveur traite la requête, il peut alors envoyer un IAM, une réponse ou les deux. Le préserveur attend de recevoir toutes les réponses et tous les messages d'ajustement avant de fermer la connexion. Pour ce faire deux conditions doivent être réalisées :

- les intervalles des réponses recouvrent l'intervalle initial,
- tous les serveurs vers lesquels la requête a été redirigée doivent envoyer une réponse ou un IAM.

En plus des éléments permettant de corriger l'arbre : le *Handle* du fichier, le nombre de digits communs (*CommonDigitNumber*), le serveur N' (*LastServer*), la partie de l'arbre à remplacer (*ImageAjustementTrie*) ainsi que la liste des numéros logiques *(ListServerLogical*), des ports (*ListServerPort*) et des adresses IP (*ListServerIp*) des serveurs que le client doit ajouter à sa table, le message d'ajustement contient la liste des adresses IP (*ServerIpRedirect*) et des numéros de port (*ServerPortRedirect*) des serveurs vers lesquelles la requête a été redirigée, ce qui permet au préserveur de connaître les serveurs responsables du traitement de la requête et de pouvoir mettre fin à la connexion une fois que le traitement est terminé. Toutefois, ces champs ne sont d'aucune utilité au client mobile, ils ne lui sont donc pas transmis.

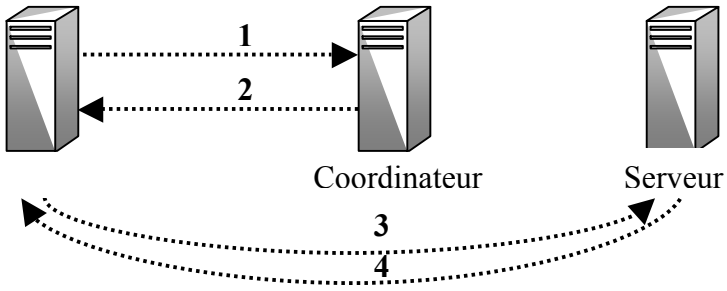
2.1.11 Ajustement de l'image de client

Le protocole HTTP est un protocole de niveau application qui s'appuie sur un modèle client/serveur. Le serveur ne peut donc pas initier des appels, il ne peut que répondre aux requêtes du client. De ce fait et comme les messages d'ajustement ne sont envoyés que

lorsqu'une requête sur des données est émise, le préserveur les transmet au client dès leur réception et ferme la connexion lorsque tous les réponses et tous les IAM ont été reçus.

C'est ainsi que nous ajoutons des champs indiquant les serveurs vers lesquelles la requête a été redirigée, ce qui permet au préserveur de savoir si tous les messages ont été reçus et de fermer la connexion.

2.1.12 Eclatement d'une case d'un serveur



1: Get Free Server

Command	*handle*	*ServerIpAdr*	*ServerPort*	*FreeSize*

2: Get Free Server Answer

Command	*ErrorCode*	*handle*	*ServerLogicalAdr*	*ServerIpAdr*	*ServerPort*	*FreeSize*

3: Init Server

Command	*handle*	*CaseSize*	*MaxKey*	*ServerLogicalAdr*	*ServerIpAdr*	*ServerPort*
RecordsList	*KeysList*					

4: Init Server Answer

Command	*handle*	*ErrorCode*

Le serveur débordé contacte le coordinateur pour lui demander d'allouer une nouvelle case au fichier. Le coordinateur consulte sa table de fichiers FAT et sa table de serveurs PAT pour chercher un serveur pouvant contenir cette case. S'il n'en trouve pas, il envoie aussitôt la réponse au serveur avec un code d'erreur (*ErrorCode*) égal à NoAvailableFreeSize. S'il existe un serveur pouvant contenir une nouvelle case du fichier, le coordinateur lui envoie un message GetFreeServerAnswer avec un code d'erreur nul. En recevant un message favorable

(*ErrorCode* nul), le serveur éclaté contacte alors le nouveau serveur pour lui envoyer les clés à insérer.

3 Implémentation et mise en œuvre

Après avoir décrit les choix conceptuels de l'application CTH*, nous décrivons dans cette section les outils et les techniques utilisés pour mettre en œuvre ce prototype.

3.1 Environnement de développement

3.1.1 Système d'exploitation

L'application CTH* fixe a été développée sous Windows 2000. Ce système offre un support pour le multitâche, une interface de programmation réseau très riche, une infrastructure de sécurité puissante et des mécanismes de gestion de la mémoire virtuelle évolués. Il fournit ainsi un ensemble de fonctionnalités sûres, robustes, évolutives et inter-opérables.

En ce qui concerne le développement du client mobile, l'environnement importe peu ; nous pouvons utiliser notre système d'exploitation habituel (en l'occurrence Windows XP) ou tout autre système (Linux, MAC OS) puisque l'exécution se fera sur un appareil mobile dont les caractéristiques sont très différentes de l'environnement de développement.

3.1.2 Langage de développement

Le coordinateur, le serveur et le client fixe sont développés en Visual C++, le client mobile et le préserveur, quant à eux, utilisent la plateforme Java. Cette section justifie le choix de ces deux langages de programmation.

Visual C++

C++ est le langage de niveau système le plus répandu au monde, et Visual C++ offre un outil de génération de logiciels d'envergure pour les systèmes Windows.

Visual C++ est un langage de programmation orienté objets, très efficace et très rapide, qui fournit plusieurs librairies et classes facilitant considérablement la programmation réseau.

Java

Java est, lui aussi, un langage de programmation orienté objet. Le principal avantage de Java est son indépendance vis-à-vis de la plateforme ; cela est possible grâce à la définition d'une machine virtuelle propre à chaque environnement. Ainsi, la plateforme Java permet de développer des applications portables, sûres, fiables, peu gourmandes en ressources et faciles à installer et à mettre à jour.

Ce caractère de portabilité confère à la plateforme Java une grande souplesse et lui permet de s'adapter à tous les environnements. De plus, afin d'améliorer les performances des applications, Java définit plusieurs plateformes de développement en fonction des systèmes cibles. Nous utilisons donc une édition micro J2ME (Java 2 Micro Edition) pour la programmation de l'application Client mobile sur le téléphone et une édition entreprise J2EE (Java 2 Entreprise Edition) pour le développement du préserveur.

J2ME est une plateforme standardisée, évolutive, largement répandue et adaptée aux appareils mobiles à faibles ressources. Elle possède des mécanismes d'abstraction de haut niveau, des interfaces de programmation (API) évoluées et une gestion sophistiquée de l'interface graphique. De plus, la standardisation des caractéristiques du langage et de ses librairies permet aux applications d'être transférées de manière très flexible entre différents appareils, dans la mesure où ils respectent les contraintes des configurations et profiles supportés par J2ME.

La plateforme J2EE offre avec les servlets HTTP, une solution puissante et facile à déployer pour gérer la réception de requêtes HTTP et l'envoi des réponses HTTP correspondantes.

En utilisant la plateforme Java du côté du client mobile et du côté du préserveur, nous simplifions la gestion du protocole HTTP.

3.2 Outils techniques de mise en oeuvre

3.2.1 Programmation concurrente

Si le multitâche instaure le parallélisme en permettant l'exécution, apparemment simultanée, de plusieurs processus, le multithreading le développe en offrant la possibilité à un processus de réaliser plusieurs traitements en même temps. Cette technique est d'autant plus efficace qu'elle permet d'optimiser l'utilisation des ressources. En effet, chaque

processus possède une copie unique de ses variables alors que les threads appartenant au même processus partagent les mêmes données.

S'il est courant que les systèmes d'exploitation pour ordinateur supportent le multitâche et le multithreading, seuls quelques rares téléphones offrent la possibilité de lancer plusieurs processus simultanément. Ainsi, la spécification MIDP n'impose qu'un support pour le multithreading. La plateforme Java, elle-même, se sert d'un thread pour gérer le ramasse-miettes (Garbage Collector) en tâche de fond et crée un thread séparé pour récupérer les événements de l'interface utilisateur.

Si le multithreading améliore les performances et optimise la gestion des ressources, il nécessite une gestion des accès concurrents afin de garantir l'intégrité des données. Différents mécanismes sont utilisés pour synchroniser l'accès aux données partagées : sections critiques, exclusion mutuelle, sémaphores ou événements. La technique la plus simple est la section critique, mais elle ne peut être utilisée que pour les threads du même processus alors que les trois autres permettent la synchronisation de threads issus de processus différents. L'exclusion mutuelle autorise un accès exclusif à une ressource, les sémaphores contrôlent le nombre d'utilisations simultanées d'une ressource et les événements permettent d'avertir un ou plusieurs threads afin d'effectuer une action donnée.

Visual C++ offre plusieurs primitives pour synchroniser les accès aux ressources ; dans la réalisation des applications clients mobiles, serveurs ou coordinateur nous utilisons la synchronisation par sémaphore ou par *mutex* (mutual exclusion).

Le langage Java offre une technique simple de synchronisation basée sur le mécanisme des sections critiques ; il suffit de placer le terme *synchronized* devant un bloc d'instructions ou devant une méthode pour la rendre utilisable par un seul thread à la fois (thread safe).

3.2.2 Les sockets

Les sockets sont la technique la plus simple et la plus rapide pour gérer les communications interprocessus. Pour les utiliser, il suffit de créer, côté serveur, une application qui écoute sur un port spécifique les connexions entrantes, ensuite il faut configurer le client pour qu'il puisse se connecter à l'hôte au port approprié. Le client peut alors envoyer les données dans le format qu'il désire.

Selon les besoins de l'application, les communications par sockets peuvent utiliser le protocole UDP (User Datagram Protocol) ou le protocole TCP (Transmission Control Protocol).

Nous utilisons les sockets UDP pour gérer les communications à l'intérieur du réseau local c'est-à-dire pour les échanges de messages entre les servlets, les clients fixes, les serveurs et le coordinateur.

Les applications MIDP 1.0 ne peuvent pas établir de connexion par sockets, cette caractéristique n'est disponible qu'avec MIDP 2.0.

3.2.3 Protocole HTTP (Hypertext Transfer Protocol)

HTTP est un protocole de communication de niveau application s'appuyant sur le modèle client/serveur. Ce protocole est idéal pour les applications Java mobiles car par spécification, tout appareil compatible MIDP doit supporter le protocole HTTP alors que d'autres protocoles, tels que TCP ou UDP sont optionnels.

Puisque tous les appareils MIDP ne supportent pas une communication basée sur les sockets ou les datagrams, le déploiement de HTTP sur des appareils mobiles permet une portabilité maximale des applications entre les appareils issus de différents constructeurs.

De plus HTTP est devenu le protocole réseau caractéristique de la plateforme J2EE.

Il utilise trois méthodes : la méthode GET pour demander la transmission d'une ressource statique, par exemple un fichier de données ou une page web ; la méthode HEAD pour vérifier les méta-informations concernant une ressource donnée, et la méthode POST pour demander un service dynamique au serveur. Dans notre application, nous utilisons la méthode POST, car le client envoie une requête qui dépend de l'état du système et qui n'est donc pas une ressource statique.

Il existe plusieurs technologies pour implémenter la partie serveur http. Afin de compléter la programmation Java et pour garantir une portabilité maximale nous utilisons les servlets HTTP. Aussi, pour installer et exécuter des servlets, nous devons utiliser un conteneur de servlets ; parmi les solutions commerciales ou libres disponibles sur le marché, Apache

Tomcat[2] se démarque par sa puissance et sa simplicité d'utilisation. Il peut, en outre, être utilisé en tant que serveur Web ou serveur d'applications.

3.2.4 Sécurité

Lors d'un transfert de données à travers un réseau, la sécurité et l'intégrité des données sont un problème important ; lorsque le réseau est mobile, le problème l'est encore plus. Ainsi, la plupart des opérateurs offrent un mécanisme de sécurité de base en ne permettant la connexion qu'à travers une passerelle gérant les mécanismes de sécurité.

En général, HTTP n'est pas sécurisé, avec un support de confidentialité, d'authentification et d'intégrité faible ou inexistant. HTTPS améliore considérablement cette situation en exécutant le protocole HTTP au dessus d'une couche sécurisée, comme par exemple TLS (Transport Layer Security) ou SSL (Secure Sockets Layer). De même, TLS ou SSL sont exécutés au dessus de TCP en y ajoutant un support pour le cryptage, l'intégrité et l'authentification. Cependant, l'utilisation du protocole HTTPS engendre un surplus de trafic et une plus grande consommation de ressources. De plus, seuls quelques appareils mobiles supportent ce protocole.

Nous avons donc préféré instaurer un mécanisme d'authentification au niveau du coordinateur CTH* et ne permettre l'accès aux données qu'aux clients authentifiés. Pour ce faire, nous utilisons une servlet qui gère l'ajout de nouveaux utilisateurs par l'administrateur de l'application CTH*. Lors de son lancement, l'application client CTH* mobile demande à l'utilisateur de fournir un nom d'utilisateur et un mot de passe ; elle envoie ensuite ces deux paramètres à la servlet d'authentification présente sur le poste coordinateur en utilisant le protocole HTTP; la servlet vérifie ces données et renvoie une réponse au client mobile ; si cette réponse est positive, l'utilisateur peut accéder aux différentes fonctionnalités de la plateforme CTH*, sinon il a le choix entre réessayer de s'authentifier ou quitter l'application.

4 Interface de l'application

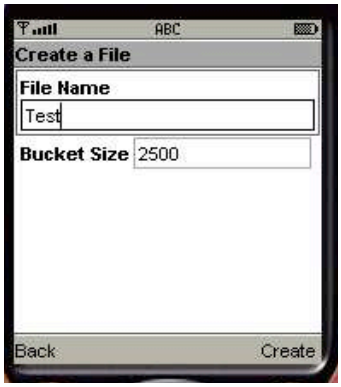
Après s'être authentifié, l'utilisateur accède à une liste de fonctionnalités permettant la manipulation des fichiers CTH* et de leurs données.

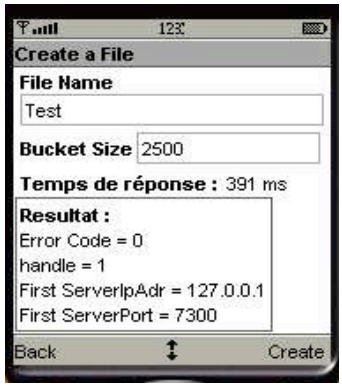
---

[2] Apache Tomcat est disponible en téléchargement à l'adresse http://jakarta.apache.org

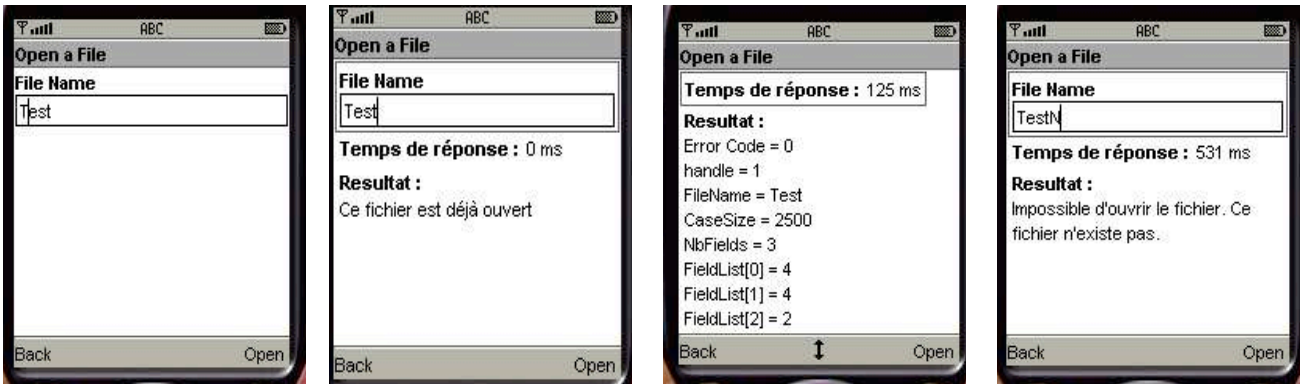
Nous pouvons ouvrir un fichier en indiquant son nom. Si le fichier n'existe pas, le coordinateur le crée puis transmet ses paramètres, sinon, il précise que ce fichier existe déjà

.





Nous pouvons également ouvrir un fichier en indiquant son nom. Le client vérifie dans sa table que le fichier n'est pas déjà ouvert et envoie la requête au coordinateur qui s'assure de l'existence du fichier et transmet ses paramètres.



Nous pouvons également envoyer une requête d'insertion en indiquant le fichier utilisé, la clé et le contenu de l'enregistrement. Si la clé ne peut pas être insérée parce qu'elle existe déjà ou par manque d'espace, nous le précisons ; sinon nous indiquons que l'article a été correctement inséré.

Une recherche est initiée en précisant le numéro du fichier et la clé de l'enregistrement recherché. Le client commence par vérifier que le fichier existe dans sa FAT puis utilise son arbre pour envoyer la requête au serveur approprié. Si la clé existe, nous spécifions le contenu de l'enregistrement associé; sinon, nous mentionnons qu'elle n'existe pas.

Nous pouvons également exécuter des requêtes à intervalle en précisant le numéro du fichier et l'intervalle de recherche. Le client CTH* vérifie que le fichier existe puis traite la

requête ; le nombre de clés trouvées est ensuite précisé et le contenu des enregistrements associés affiché.

Nous pouvons aussi avoir accès à la table de fichiers du client CTH* mobile.

Enfin, nous gérons également des fichiers pour permettre une utilisation aisée et un confort accru. Nous pouvons alors créer des fichiers à l'aide du simulateur en bénéficiant de la souplesse et de l'ergonomie de l'ordinateur, puis les transférer sur le téléphone pour les utiliser. Nous pouvons également conservé les résultats obtenus dans un fichier pour les consulter ultérieurement.

5 Synthèse

Nous venons d'examiner en détail l'architecture des composants de la plateforme CTH* ainsi que les échanges de messages entre ces entités.

Nous avons également exposé les techniques de mise en œuvre du prototype CTH* intégrant des clients fixes et mobiles et présenté l'interface de l'application CTH* mobile.

La prochaine étape est donc d'effectuer les tests permettant de valider notre schéma de conception et de mesurer les performances de notre prototype.

Chapitre V

Validation et Tests

Après avoir présenté les différentes étapes de conception et de réalisation du prototype client CTH* mobile, nous décrivons, dans ce chapitre, les expérimentations que nous avons conduites afin de valider nos choix conceptuels et techniques et de mesurer les performances de notre prototype.

La première section présente l'environnement expérimental, la seconde les tests menés pour vérifier la faisabilité de la solution et valider les choix architecturaux et la dernière les tests de performances et l'analyse des résultats obtenus.

1 Environnement expérimental

Habituellement les applications sont développées, testées et évaluées dans une plateforme similaire à celle où elles vont être déployées et exécutées. Le développement d'applications mobiles est assez différent car l'application est développée dans une plateforme (telle que MS Windows ou Linux) et qu'elle est déployée dans une autre plateforme totalement différente de la première (telle qu'un téléphone cellulaire ou un PDA). En conséquence, les émulateurs permettent de tester l'application dans la plateforme de développement.

Afin d'évaluer l'impact de chaque composante du système sur le temps de réponse, nous testons le prototype dans différents environnements ; d'abord en environnement local puis sur un simulateur distant pour mesurer l'influence de l'utilisation du protocole HTTP et évaluer l'effet du passage par Internet. Enfin, nous déployons l'application client CTH* mobile sur le téléphone mobile et testons ses performances dans un réseau mobile réel.

Les serveurs CTH* sont déployés sur trois machines ; la première dispose d'un processeur Pentium IV avec une fréquence d'horloge de 2,66 GHz et 248 Mo de RAM, la seconde possède un processeur Pentium III tournant à 1GHz et muni de 256 Mo de RAM et le troisième est un Pentium III à 688 MHz avec 256 Mo de RAM. Ces machines sont interconnectées par un réseau Ethernet 100 Mbits/s. Pour tester l'application en environnement local, nous exécutons le client CTH* fixe sur un ordinateur portable à processeur Pentium de technologie Centrino avec une fréquence d'horloge de 1,73 MHz et une RAM de 512 Mo, il est relié au

réseau Ethernet local. Cette même machine est utilisée pour tester le client mobile distant à l'aide du simulateur intégré de Sun J2ME Wireless Toolkit. Pour vérifier le parallélisme, nous utilisons une autre machine connectée à Internet, qui dispose d'un processeur Pentium IV tournant à 3GHz et à 256 de RAM. Toutes les machines utilisées sont munies d'un système d'exploitation Windows XP SP2. Enfin, pour les tests réels nous utilisons un téléphone mobile Nokia 3200 à 800 Ko de mémoire, compatible Java MIDP 1.0 et relié à Internet via le réseau GPRS d'Algérie Télécom.

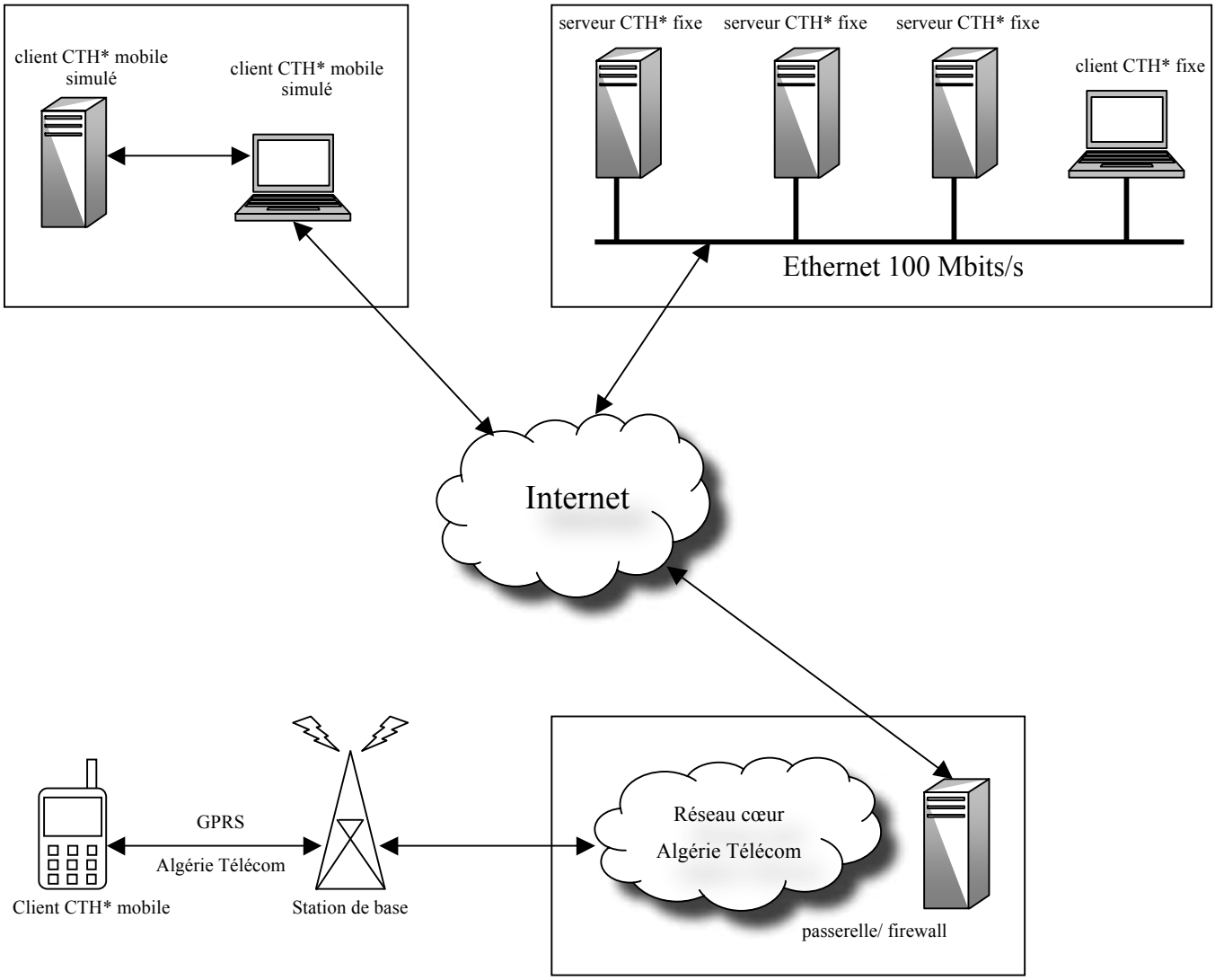


Figure V.1 Architecture du réseau expérimental

2 Tests de validité

Afin de valider notre solution, nous avons testé les différentes opérations proposées dans diverses configurations et avons mesuré les temps de réponse.

Les cases des fichiers sont de capacité limitée, cela permet de mieux analyser le comportement de la méthode lorsqu'un grand nombre de serveurs est utilisé. De plus, la structure du fichier reste la même si la capacité de la case et le nombre d'enregistrements sont multipliés par le même facteur. Autrement dit, l'insertion de 500 clés avec une capacité de case de 250 octets donne à peu près le même nombre de serveurs et la même structure de fichier que l'insertion de 5 000 000 de clés avec une capacité de case de 2 500 000 octets.

Nous avons d'abord vérifié la création, ouverture, fermeture et suppression de fichiers par un client puis par plusieurs simultanément.

Nous avons ensuite testé la scalabilité en mesurant le temps total puis le temps moyen d'insertion en fonction du nombre d'articles insérés et de la capacité de la case. Nous avons aussi évalué le temps d'exécution des requêtes à intervalle en fonction du nombre de serveurs auxquels la requête fait appel et calculé le facteur de chargement.

Nous avons également vérifié le parallélisme : dans une première étape, plusieurs clients insèrent en parallèle un ensemble d'enregistrements. Ensuite, chaque client lance une requête à intervalle de la plus petite clé (a_) à la plus grande (z|) et récupère tous les articles insérés. Puis les requêtes à intervalle sont lancées par tous les clients en même temps et doivent donner les mêmes résultats.

Enfin, nous avons, dans chaque cas, calculé le facteur de chargement des cases des fichiers et comparé les résultats en fonction de la taille de la case et du nombre d'enregistrements insérés.

3 Tests de performances

Insertion

Pour évaluer les temps de réponse, nous insérons des enregistrements dont la taille des clés est entre 5 et 30 octets en variant le nombre d'articles insérés et la capacité de la case du fichier.

Nous commençons les tests par l'évaluation des temps d'insertion en fonction du nombre d'enregistrements insérés par un client fixe pour différentes capacités de case (figure V.2).

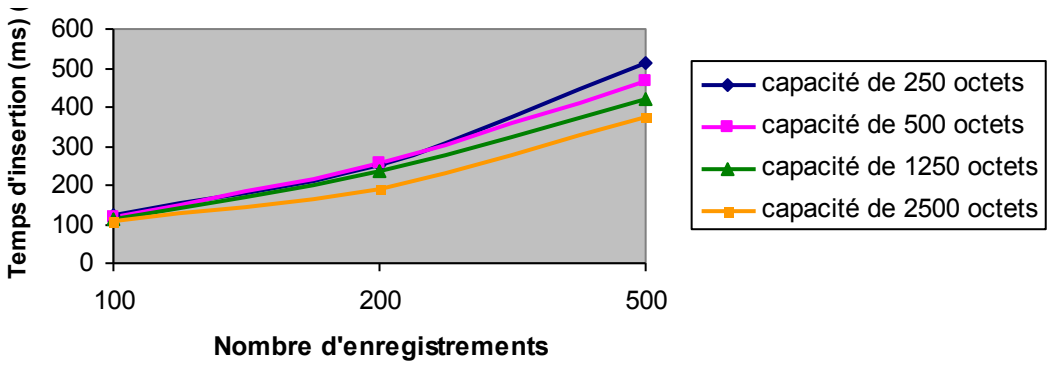


Figure V.2 Evolution du temps total d'insertion pour un client fixe d'image initialement vide

Nous remarquons que les temps d'insertion augmentent linéairement avec le nombre d'enregistrements quelle que soit la capacité de la case, ce qui signifie que l'accroissement du volume de données ne dégrade pas les performances.

Pour des résultats plus significatifs, nous calculons le temps moyen d'insertion d'un seul enregistrement (figure V.3).

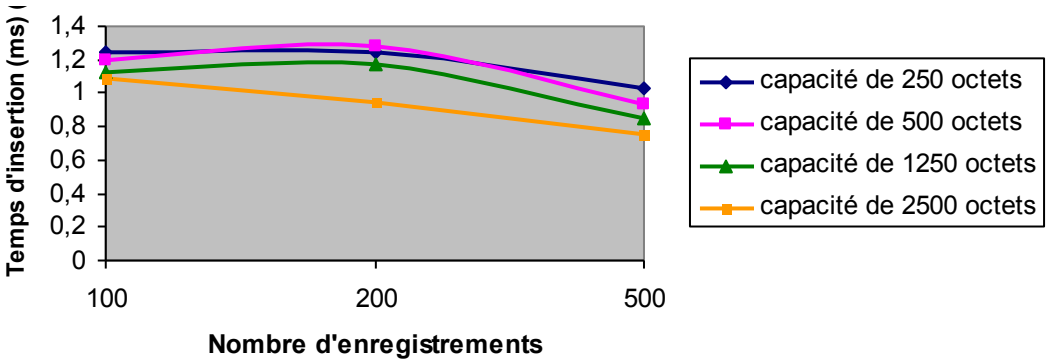


Figure V.3 Temps moyen d'insertion pour un client fixe

Nous observons une diminution du temps moyen d'une insertion lorsque la capacité de la case croît. En effet, plus la capacité de la case est importante moins il y a d'éclatements de serveurs surchargés, ce qui provoque une réduction des temps de réponse. De plus, lorsque le nombre de serveurs diminue, les clients font moins d'erreurs d'adressage, ce que limite les redirections et réduit la charge des serveurs.

Nous constatons également une baisse des temps d'insertion lorsque le nombre d'enregistrements grandit, ceci est du à l'augmentation du niveau de parallélisme. En effet, plus le nombre d'enregistrements insérés est élevé plus le nombre de serveurs contenant une

case du fichier est important, par conséquent, les requêtes faisant appel à des cases différentes peuvent être traitées en parallèle.

Ces résultats sont comparés à ceux obtenus avec un simulateur distant.

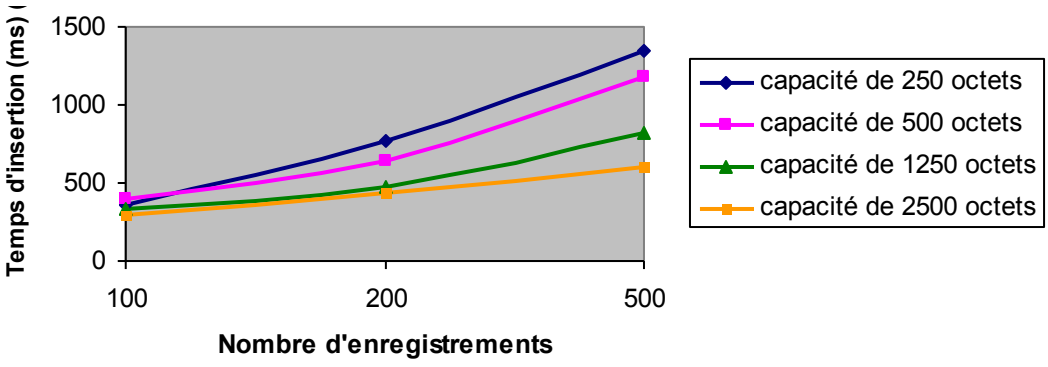


Figure V.4 Evolution du temps total d'insertion pour un client distant

Tout comme dans le cas du client fixe, le temps d'insertion d'un enregistrement par un client mobile simulé croît linéairement avec le nombre d'enregistrements.

La figure V.5 illustre les temps moyen d'insertion d'un enregistrement.

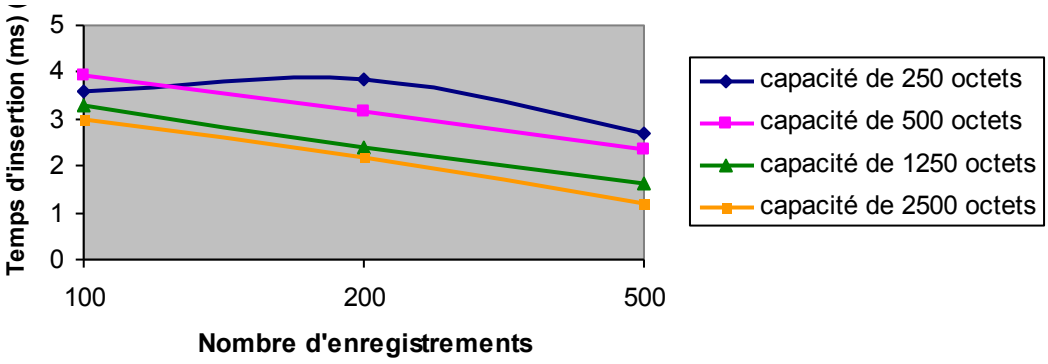


Figure V.5 Temps moyen d'insertion pour un client distant

Les temps moyens d'insertion restent stables et peuvent diminués en cas d'augmentation du nombre d'enregistrements pour les raisons évoquées plus haut. Toutefois, l'utilisation du protocole HTTP et le passage par Internet engendrent des temps 2 à 3 fois plus grands que ceux d'un environnement local.

Ces résultats sont à leur tour comparés à ceux obtenus avec un téléphone mobile connecté à Internet via un réseau GPRS.

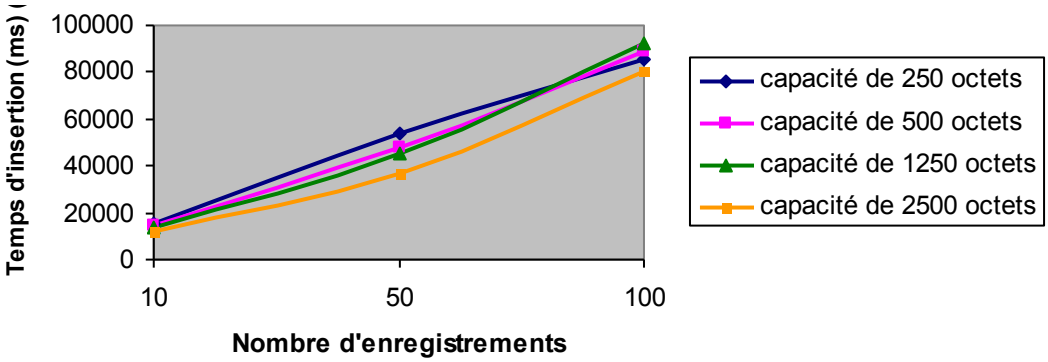


Figure V.6 Evolution du temps total d'insertion pour un client mobile

Nous observons, dans ce cas aussi, une augmentation linéaire des temps d'accès en fonction du nombre d'enregistrements insérés.

Pour une comparaison indépendante du nombre de clés insérées, nous utilisons les temps moyens d'insertion.

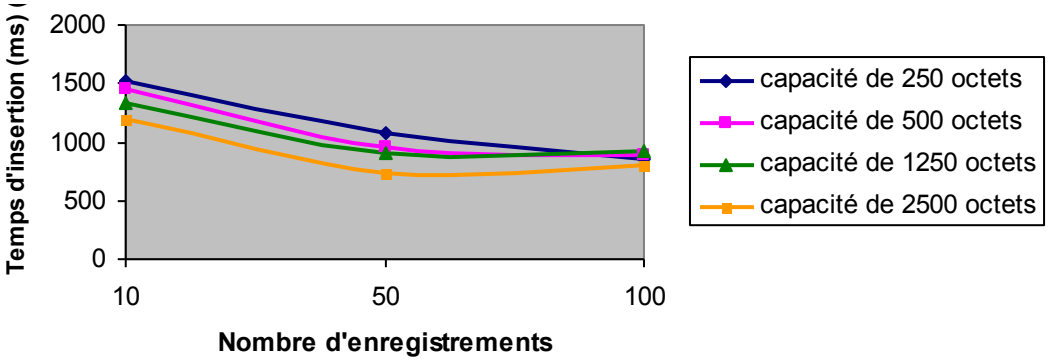


Figure V.7 Temps moyen d'insertion pour un client mobile

Les temps d'insertion sont, tout comme dans le cas d'un environnement local ou sur un simulateur distant, constants et n'augmentent pas avec le volume de données stockées. Toutefois, ces temps sont 700 à 1000 fois plus grands que ceux obtenus avec le client fixe, cela est principalement du aux faibles débits offerts par les réseaux mobiles.

Requête à intervalle

Nous avons mesuré le temps d'exécution d'une requête à intervalle portant sur tous les enregistrements du fichier par un client dont l'arbre est vide. Les résultats sont décrits dans la figure V.8.

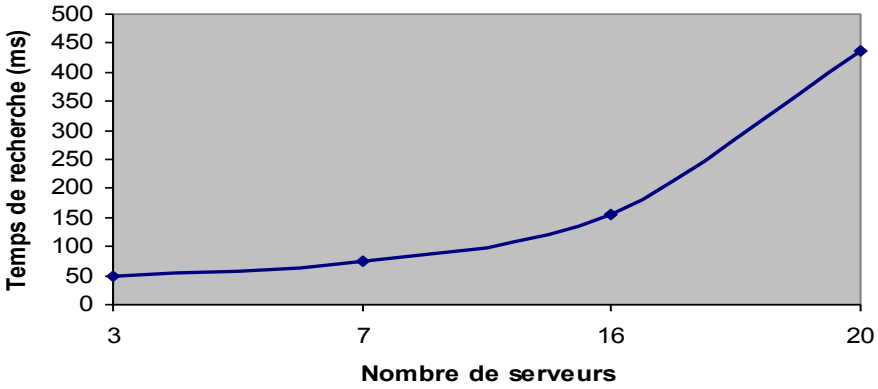


Figure V.8 Evolution du temps de recherche à intervalle pour un client fixe (plusieurs serveurs par machine)

Nous observons une augmentation des temps de recherche à mesure que le nombre de serveurs grandit. Cette augmentation est du à l'utilisation de plusieurs serveurs logiques sur le même serveur physique ; le traitement de la requête est donc pseudo-parallèle.

En affectant un serveur unique à chaque machine, nous obtenons des temps d'exécution qui diminuent à mesure que le nombre de serveurs augmente et même si le nombre d'enregistrements recherchés croît.

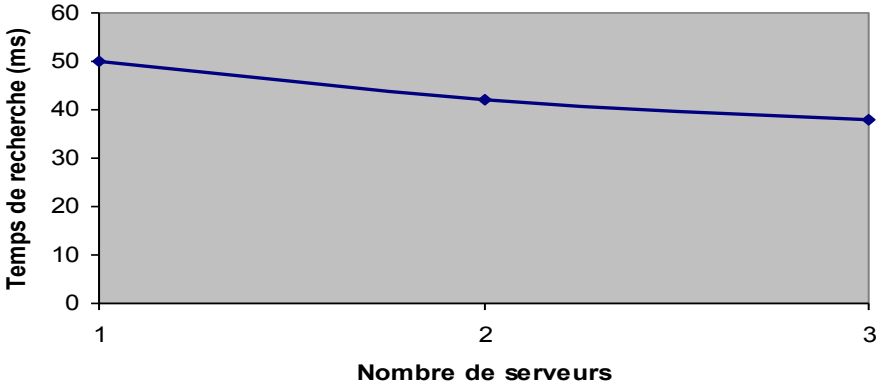


Figure V.9 Evolution du temps de recherche à intervalle pour un client fixe (un seul serveur par machine)

La même remarque est valable pour le cas distant et mobile.

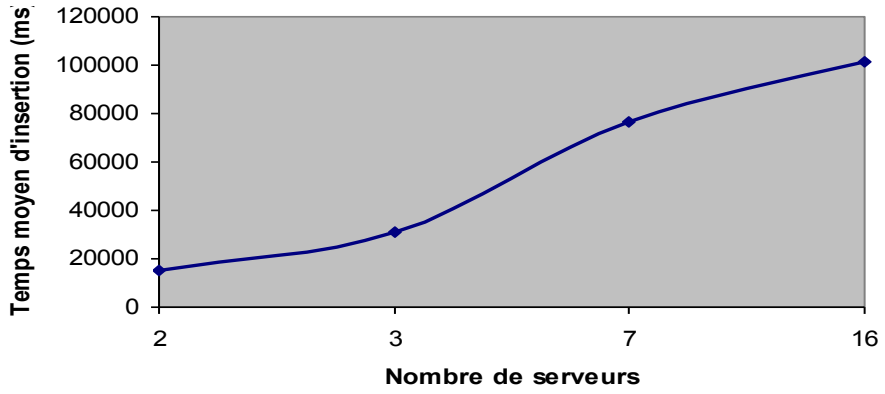


Figure V.10 Evolution du temps de recherche à intervalle pour un client mobile (plusieurs serveurs par machine)

En affectant un serveur CTH* unique à chaque machine, nous obtenons les résultats de la figure V.11.

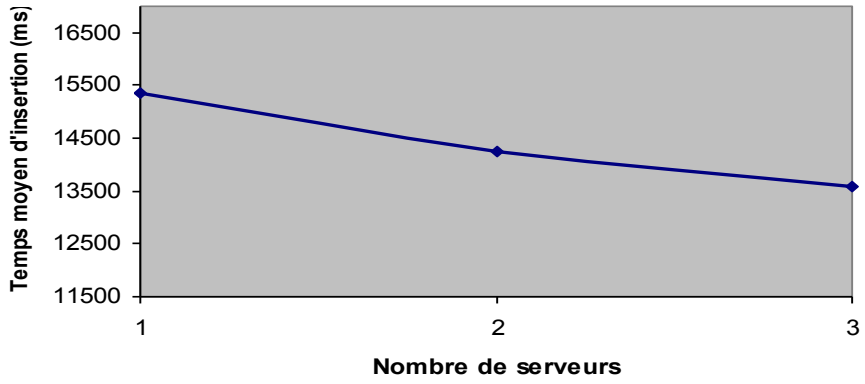


Figure V.11 Evolution du temps de recherche à intervalle pour un client mobile (un serveur par machine)

Facteur de chargement

Lors des différentes insertions, nous avons calculé le taux de remplissage des différents serveurs en fonction de la capacité de la case du fichier. Les résultats obtenus sont portés dans la figure V.12.

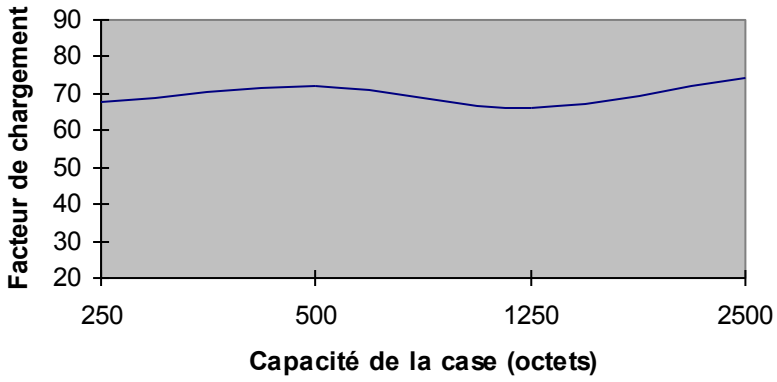


Figure V.12 Evolution du facteur de chargement

Nous remarquons que le facteur de chargement est indépendant de la taille de la case et varie autours de 70%, ce qui est comparable au facteur de chargement des méthodes LH* et RP*.

4 Synthèse

Si les tests de validité ont permis de vérifier la faisabilité de notre prototype client CTH* mobile, les tests de performances ont servi à vérifier la scalabilité et à mesurer les temps de réponses et le facteur de chargement; ces temps sont stables et indépendants du volume de données traitées.

A partir de ces résultats, nous pouvons prévoir les performances du prototype CTH* dans des conditions réelles et anticiper les temps de réponses en cas de fichiers volumineux. En effet, les tests ont utilisé de petites capacités de case pour créer un grand nombre de serveurs et mesurer l'impact sur la structure du fichier et sur les temps d'accès.

Dans le cas réel, il est préférable d'affecter un serveur unique à un chaque machine et d'utiliser de grandes capacités de case. Cette stratégie permet d'augmenter le niveau de parallélisme global et par conséquent améliore les temps de réponse.

Enfin, en comparant les temps de réponse dans divers environnements, nous avons constaté que dans un environnement mobile réel les temps de réponse sont de 700 à 1000 fois plus grands que ceux des réseaux locaux. Ces performances restent tout de même avantageuses pour un environnement mobile car une insertion est exécutée en une seconde environ.

Cette situation est appelée toutefois à évoluée, notamment par l'avènement des réseaux 3G qui offrent des débits de l'ordre d'une centaines de kilobits et par l'utilisation de terminaux mobiles évolués qui disposent de ressources en calcul et en stockage accrues.

Conclusion

Le thème de ce mémoire était donc la conception, l'implémentation et l'évaluation d'un client mobile intégré à la plateforme CTH* fixe. Le prototype réalisé et que nous avons présenté se caractérise par son efficience, sa simplicité, sa scalabilité et sa flexibilité.

En effet, en offrant une bibliothèque de fonctions pour gérer les opérations sur les fichiers CTH*, les applications mobiles peuvent utiliser la plateforme CTH* de manière **simple** et **rapide** sans se préoccuper des mécanismes sous-jacents et des échanges de messages entre les différentes entités CTH*.

Nous avons aussi pu tester la **scalabilité** de notre système. Grâce à la distribution des données sur plusieurs sites et l'utilisation des technologies appropriées, nous pouvons garantir des performances stables et optimales quelle que soit la masse d'informations stockées.

De plus, en élargissant l'accès à travers des réseaux mobiles, nous assurons l'**ubiquité** et l'omniprésence du service.

Enfin, l'architecture modulaire de la plateforme lui confère un caractère de **flexibilité** et d'**extensibilité**. Ainsi, le prototype réalisé est une base que nous pouvons enrichir avec diverses fonctionnalités. Nous pouvons contrôler le flux ou gérer la haute disponibilité pour accroître la résistance aux pannes. Nous pouvons également améliorer l'organisation de la case du fichier en adoptant une structure en B-arbre ou en CTH (Compact Trie Hashing).

Ces résultats ont pu être obtenus grâce à l'utilisation d'un préserveur, qui joue le rôle d'un intermédiaire entre les clients mobiles et les serveurs et coordinateur fixes, ce qui nous a permis de dépasser l'hétérogénéité des systèmes et des réseaux, de masquer la diversité des matériels et logiciels et de réutiliser la plateforme existante.

Le prototype réalisé peut par la suite être intégré à un système de gestion de bases de données performant prenant en charge de grandes masses de données de natures variées (voix, texte, image, vidéo, multimédia,…) et offre un accès en temps réel, une gestion de la haute

disponibilité et une ubiquité du service. Ce prototype peut également être utilisé pour la mise en œuvre de systèmes décisionnels et d'applications à hautes performances en général. Nous pouvons aussi envisager son utilisation dans une architecture pair-à-pair (P2P) qui permet de garantir un partage de ressources simple, flexible et adapté aux environnements mobiles.

Le projet SD²M ouvre donc des perspectives d'avenir pour le développement et le déploiement d'applications potentiellement illimitées, intégrant des dispositifs fixes et mobiles et bénéficiant d'un accès simple, rapide et efficace à une grande masse d'informations, sans limite de temps ni d'espace.

Il illustre également la convergence des domaines de l'informatique et des télécommunications. Cette convergence est matérialisée à la fois par l'évolution des services de voix offerts par les réseaux mobiles vers la transmission de données et par le prolongement des applications fixes vers les environnements mobiles.

Bibliographie

[AIT05] K. AIT ALI, *Un schéma de hachage digital compact distribué avec multicast,* thèse de magistère, université Mouloud Mammeri de Tizi-Ouzou, 2005.

[ALA01] K. AL AGHA, G. PUJOLLE, G. VIVIER, *Réseaux de mobiles & réseaux sans fil*, Eyrolles, septembre 2001

[BEN00] F. BENNOUR, *Contribution à la Gestion de Structures de Données Distribuées et Scalables*, thèse de doctorat, Université Paris Dauphine, Juin 2000

[BIR02] MS. BIRECH, Y. LAALAOUI, *Protocole de communication pour la nouvelle classe de SDDS : CTH**, mémoire d'ingénieur, INI, 2002.

[BOU05] A. BOURIDAH, A. KHELDOUN, *Implémentation et étude de performance de la SDDS CTH* avec T-trees comme organisation interne du serveur*, mémoire d'ingénieur, INI, 2005.

[BOZ02] P. BOZANIS, Y. MANOLOPOULOS, *LDT: A Logarithmic Distributed Search Tree*, In Proceedings of WDAS 2002, Carleton scientific, 2002.

[BUR83] WA. BURKHARD, *Interpolation-based Index Maintenance*, In Proceedings of the 2nd Symposium on Principles of databases Systems. PODS, 1983.

[DEV93] R. DEVINE, *Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm*, 4th International Foundation of Data Organization and Algorithms–FODO, 1993.

[DIE01] AW. DIENE, *Contribution à la Gestion de Structures de Données Distribuées et Scalables*, thèse de doctorat, Université Paris Dauphine, novembre 2001.

[FAG79] R. FAGIN, J. NIEVERJELT, N. PIPPENGER, *Extendible Hashing – A Fast Access Method for Dynamic Files*, ACM Transactions on Database Sytems 1979.

[FOU01] A. FOUQUET, JP. WORGAGUE, *La révolution des usages portée par l'évolution technologique des réseaux*, Orange Press, 2001.

[HID03] WK. HIDOUCI, DE. ZEGOUR, *Actor oriented Databases*, Institut National d'Informatique, 2003

[HIL97] V. HILFORD, F. B. BASTANI, B. CUKIC, *EH* : Extendible Hashing in a Distributed Environment*, COMPSAC '97 - 21st International Computer Software and Applications Conference, 1997

[KAR96] J. KARLSON, W. LITWIN, T. RISCH, *LH^*_{LH}: A Scalable high performance data structure for switched multicomputers*, EDBT 96, Springer Verlag.

[KHA04] A. KHALED, A. BEKKOUCHE, *Implémentation d'une Structure de Données Distribuée et Scalable sous Linux : Une nouvelle version de CTH**, mémoire d'ingénieur, INI, 2004.

[KRO94] B. KROLL, P. VIDMAYER*, Distributing a Search Tree among a Growing Number of Processors*, ACM International Conference on Management of Data –SIGMOD, 1994.

[LAR78] P. LARSON, *Dynamic Hashing*, BIT Vol. 18-(2), 1978.

[LIT80] W. LITWIN, *Linear Hashing: a new tool for file and tables addressing*, Reprint from VLDB-80.

[LIT81] W. LITWIN, *Trie hashing*, SIGMOD 81, ACM, mai 1981

[LIT93] W. LITWIN. MA. NEIMAT, D. SCHNEIDER, *LH* : Linear Hashing for Distributed Files,* ACM-SIGMOD International Conference on Management of Data, 1993.

[LIT94] W. LITWIN, M.A. NEIMAT & D. SCHNEIDER, *RP*: A Family of Order-preserving Scalable Distributed Data Structures*, In Proceedings of the 20th VLDB Conference, Santiago, Chilli, 1994.

[LIT95] W. LITWIN, MA. NEIMAT, *$k\text{-}RP^*_S$: A Scalable Distributed Data Structure for High-Performance Multi-Attribute Access,* Res. Rep. GERM Paris 9 & Distributed Inf. Techn. Dep. HPL Palo Alto, Avril 1995.

[LIT96a] W. LITWIN, MA. NEIMAT, *High-Availability LH* Schemes with Mirroring*, Intl. Conf. on Coope, Inf. Syst. COOPIS-96, Bruxelles, 1996.

[LIT96b] W. LITWIN, MA. NEIMAT, D. SCHNEIDER, *LH*: A Scalable Distributed Data Structure,* ACM-TODS, December 1996

[LIT97a] W. LITWIN, MA. NEIMAT, G. LEVY, S. NDIAYE, *LH*$_S$: a High-availability and High-security Scalable Distributed Data Structure*, IEEE Workshop on Res. Issues in Data Engineering, 1997.

[LIT97b] W. LITWIN, T. RISCH, *LH*g: A High-Availability Scalable data Structure by Record Grouping*, Res. Rep. U. Paris9 & U. Linkoping, 1997.

[LIT98] W. LITWIN, J. MENON, T. RISCH, *LH* with Scalable Availability*, IBM Almaden research report RJ 10121 (91937), Mai 1998.

[MAC05] G. MACKE, *Microsoft, la vie au-delà du PC*, Le Monde, Samedi 3 décembre 2005, page17.

[MER03] B. MERANI, H. YAKOUBEN, *Implémentation de CTH* avec arbre central sous Linux*, mémoire d'ingénieur, INI, 2003.

[MOU04] R. MOUSSA, *Contribution à la Conception et l'Implantation de la Structure de Données Distribuée & Scalable à Haute Disponibilité LH*$_{RS}$*, thèse de doctorat, Université Paris Dauphine, 4 octobre 2004.

[NDI00] Y. NDIAYE, F. BENNOUR, AW. DIENE, *Scalable and distribued linear hashing LH*$_{LH}$ under Windows NT*, SCI-2000 Orlando, Florida, USA, juillet 23-26 2000.

[NDI01] Y. NDIAYE, *Interopérabilité d'un Serveur de Structures de Données Distribuées et Scalables et d'un SGBD relationnel-objet*, thèse de doctorat, Université Paris Dauphine, 13 novembre 2001,.

[PAR05] P. PARREND, *Les Evolutions des Réseaux de Seconde Génération*, 2005

[PAS99] A. DI PASQUALE, E. NARDELLI, *A Very Efficient Order Preserving Scalable Distributed Data Structure*, In Proceedings of WDAS 1999, Carleton scientific, 1999.

[SAA04] B. SAAD, A. BOULARIAS, *implémentation et étude des performances de la SDDS Compact Trie Hashing (CTH*) sous Windows 2000*, mémoire d'ingénieur, INI, 2004.

[SAH04] S. SAHRI, W. LITWIN, Implementing SD-SQL Server: a Scalable Distributed Database System, Res. Rep. CERIA, Mars 2004

[SAI04] S. SAICHI, B. CHALGOU, *Adaptation de la SDDS CTH* à un environnement Peer To Peer*, mémoire d'ingénieur, INI, 2004.

[SEV90] C. SEVERANCE, S. PRAMANIK, AND P. WOLBERG*, Distributed Linear Hashing and Parallel Projection in main memory database*, In Proceedings of VLDB, 1990.

[TAN94] TANENBAUM, *Distributed Operating Systems*, Prentice Hall, 1994.

[THE01] F. THENOZ, *Les nouveaux objets communicants*, Orange Press, 2001.

[WIK06] *Téléphone mobile*, Wikipédia, http://fr.wikipedia.org/wiki/Téléphone_mobile, visité le 21 mai 2006.

[ZEG04] DE. ZEGOUR, *Scalable Distributed Compact Trie Hashing*, Information and Software Technologie, 25 mai 2004.

[ZEG94] D.E ZEGOUR, W. LITWIN, *Trie hashing with the sequential representations of the trie*, Revue internationale des technologies avancées. CDTA, Alger. 1994.

Liste des abréviations

3GPP	Third Generation Partnership Project
8PSK	Eight-level Phase Shift Keying

A

AFS	Andrew File System
AMPS	Advanced Mobile Phone System
API	Application Programming Interface
ARIB	Association of Radio Industries and Business
ARM	Advanced RISC Machine

B

BDST	Balanced and Distributed Search Trees

C

CDC	Connected Device Configuration
CDMA	Code Division Multiple Access
CF	Compact Flash
C-HTML	Compact-HTML
CISC	Complex Instruction Set Computer
CLDC	Connected Limited Device Configuration
CSD	Circuit Switched Data
CTH	Compact Trie Hashing
CTH*	Hachage Digital Compact Distribué
CVM	Compact Virtual Machine

D

DDH	Dynamic Distributed Hashing
DFS	Distributed File System
DLH	Distributed Linear Hashing
DMS	Data and Messaging Service
DoPa	Docomo Packet
DRT	Distributed Random Tree

DSP	Digital Signal Processor

E

EDGE	Enhanced Data Rates for GSM Evolution
EH	Extendible Hashing
ETACS	Extended TACS
E-TDMA	Extended TDMA
ETSI	European Telecommunication Standard Institute

F

FAT	File Allocation Table
FDD	Frequency Division Duplex
FDMA	Frequency Division Multiple Access
FFSK	Fast Frequency Shift Keying
FIFO	First In First Out
FPLMTS	Future Public Land Mobile Telecommunication System

G

GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications

H

HDR	High Data Rate
HMI	Human Machine Interface
HSCSD	High Speed Circuit Switched Data
HSDPA	High Speed Downlink Packet Access
HSUPA	High Speed Uplink Packet Access
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol

I

IAM	Image Adjustment Message
IMT 2000	International Mobile Telecommunication system 2000
IP	Internet Protocol
IrDA	Infrared Data Association

IS-54	Interim Standard 54
ITS	Intelligent Transportation Systems

J

J2EE	Java 2 Entreprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JDC	Japan Digital Cellular
JTACS	Japanese Total Access Communication System

K

KVM	Kilobyte Virtual Machine

L

LCD	Liquid Crystal Display
LDT	Logarithmic Distributed Search Tree
LH	Linear Hashing

M

MIDP	Mobile Information Device Profile
MMC	Multimedia Memory Card

N

N-CDMA	Narrowband CDMA
NFS	Network File System
NiMH	Nickel Metal Hydride
NMT	Nordic Mobile Telephone

O

OFDM	Flash Orthogonal Frequency Division Multiplexing
OS	Operating System

P

P2P	Peer-to-Peer
PAT	Physical Allocation Table
PC	Personnel Computer
PCS	Personal Communication Services
PDA	Personnel Digital Assistant

PDC	Personal Digital Cellular system
PDC-P	Packet Data Cellular

Q

QPSK	Quadrature Phase Shift Keying
QVGA	Quarter Video Graphics Array

R

RF	Radio Frequency
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RP	Range Partitioning

S

SD	Secure Digital
SD-DBS	Scalable and Distributed Data Base System
SDDS	Structure de Données Distribuée et Scalable
SDK	Software Development Kit
SGSN	Serving GPRS Support Node
SIM	Subscriber Identity Module
SMS	Short Message Service
SoC	System on Chip
SRAM	Static Random Access Memory
SSL	Secure Sockets Layer

T

TACS	Total Access Communication System
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TDD	Time Division Duplex
TDMA	Time Division Multiple Access
TIA	Telecommunication Industry Association
TLS	Transport Layer Security

U

UDP	User Datagram Protocol
UI	User Interface

UI	User Interface
UIM	User Idendity Module
UIT	Union Internationale de Télécommunication
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
UTRA	UMTS Terrestrial Radio Access
UWC-136	Universal Wireless Communication 136

V

W

WAE	Wireless Application Environment
WAP	Wireless Application Protocol
W-CDMA	Wideband Code Division Multiple Access
WDP	Wireless Datagram Protocol
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
WML	Wireless Markup Language
WPAN	Wireless Personnal Area Network
WSP	Wireless Session Protocol
WTLS	Wireless Transport Layer Security
WTP	Wireless Transaction Protocol

X

xHTML	extensible HTML

Y

Z