UNIVERSITÉ DU QUÉBEC À MONTRÉAL

LES BASES DE DONNÉES MOBILES ET RÉPARTIES

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUES

PAR

PIERRE SHARUSVENS

DÉCEMBRE 2016

REMERCIEMENTS

J'aimerais tout d'abord exprimer mes remerciements à Monsieur Abdellatif OBAID, professeur au département d'informatique de l'Université du Québec à Montréal ayant accepté de diriger ce travail, pour ses pertinents conseils, sa disponibilité et surtout son humilité.

J'exprime ma plus sincère reconnaissance à mes collègues du Laboratoire de recherche LATECE qui m'ont prodigué des conseils, particulièrement à Anne Marie, Fabio, Orival et Claude pour leurs conseils et leurs motivations.

Je tiens aussi à remercier les personnes de ma famille pour leur soutien, leur confiance et leur amour indéfectibles. Merci à Papa, Maman, James, Hernsly, mes sœurs et ma femme Pierre A. Sandy. Merci pour votre amour et d'être toujours là quand j'en ai besoin.

DÉDICACE

À Ormenthide Pierre et Rivière Damilia,
qui ont toujours souhaité ce grand jour.

À mon père,
pour toutes ces valeurs qu'il m'a transmises.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

3G : 3^{e} génération de normes de téléphonie mobile

4G : 4^{e} génération de normes de téléphonie mobile

BC : Base de données cachée

BS : Serveur de base de données

BTS : Base transceiver station.

DU : Demande Utilisateur

eNode B : evolved Node B

GC : Gestionnaire de cache

LTE : Long Term Evolution

LTE-Advanced : Long Term Evolution - Advanced

RU : Réponse Utilisateur

TCP/IP : Transmission Control Protocol/Internet Protocol

T_R : Temps de réception

T_S : Temps d'envoi

T_{Trans} : Temps de transfert

UM : Utilisateur mobile

RÉSUMÉ

Depuis quelques années, les évolutions réalisées dans les domaines des terminaux portables et des réseaux sans-fil suscitent un intérêt croissant pour l'informatique mobile. La croissance des architectures des données distribuées, l'augmentation des ressources et la répartition des données et des services deviennent de plus en plus une obligation. Les systèmes de gestion de base de données ont beaucoup évolué et sont présents dans tous les environnements informatiques.

Cependant, les bases de données dans un environnement mobile font face à un certain nombre de contraintes de disponibilités des ressources, de stockage et de connectivité. Ainsi, le traitement des requêtes dans un environnement mobile nécessite des approches différentes par rapport à l'approche classique. Surtout dans l'environnement distribué qui est souvent caractérisé par des nœuds hétérogènes.

Pour pallier à ce problème, nous proposons une approche basée sur l'approche classique. Elle consiste à élaborer un plan de gestion de cache axé sur la mise en cache des résultats des requêtes de l'utilisateur mobile. Elle permet de mieux répondre à d'autres requêtes dans le futur. Elle utilise un plan de traitement des requêtes qui priorise une exécution locale afin de minimiser le coût de traitement et de faciliter une gestion efficace des données. Notre approche traite localement une requête déjà traitée sur un même BTS en utilisant les données cachées sur ce BTS. Elle permet diminuer le coût de traitement et des frais d'utilisations des données.

Mots-clés : cache, utilisateur mobile, gestionnaire de cache, LTE, base de données distribuée, eNodeB, BTS.

CHAPITRE I

INTRODUCTION

Une base de données est un outil permettant de stocker et de retrouver l'intégralité des données stockées. Une base de données est constituée de plusieurs fichiers de données situées sur un site ou sur différents sites sur un réseau dans le cas d'une base donnée répartie (Swaroop et Shanker, 2010*)*

De nos jours, les systèmes de gestion de base de données peuvent être centralisés, distribués, mobiles ou embarqués. Elles ont toutes pour objectif de répondre à des requêtes exprimées par des usagers. Au fil des années, les avancées des télécommunications sans fil et la prolifération de la technologie mobile permettent de stocker des données de tailles relativement importantes. Elles permettraient également à un utilisateur d'exécuter des requêtes n'importe où, n'importe quand et à partir de n'importe quel terminal. Mais les architectures des bases de données mobiles sont de plus en plus complexes et sont limitées par un manque d'évolutivité. Ces contraintes portent sur plusieurs aspects, d'une part la résolution des requêtes, d'autre part la possibilité de faire une gestion améliorée du cache.

Partant du fait que la croissance afflux des terminaux mobiles, la complexité et la multiplication quotidienne des requêtes des utilisateurs mobiles. Il est essentiel de prévoir une évolution afin de répondre aux changements constants et à l'adaptabilité.

1.1 Contexte de la recherche

Depuis quelques années, les évolutions réalisées dans les domaines des terminaux portables et des réseaux sans-fil suscitent un intérêt croissant pour l'informatique mobile. L'utilisation de ces nouveaux environnements introduit de nouvelles problématiques (Forman et Zahorjan, 1994) et crée de nouveaux besoins. De plus, la disponibilité de ces ressources n'est pas stable et peut varier en fonction d'ajout, de suppression de périphériques ou des limitations imposées par des politiques d'économie de la batterie. (Le Mouël, 2003). De ce fait, les performances observées sur le lien sans-fil sont soumises à d'importantes variations occasionnées par l'environnement proche comme les interférences et les déconnexions dues à des éléments physiques, des changements de lieux géographiques ou de réseau. (Le Mouël, 2003).

Les différences qui existent entre les environnements mobiles par rapport aux environnements fixes nous amènent à reconsidérer l'exploitation des périphériques dans un tel environnement. En effet, nous ne pouvons pas considérer les changements d'état des terminaux utilisés (connecté, non connecté, etc.) comme des erreurs fatales. D'ou la nécessitée d'adapter le comportement des bases de données mobiles aux différents états que peuvent prendre un équipement dans l'environnement mobile.

1.2 Problématique

Le traitement efficace des requêtes est d'un grand intérêt dans les bases de données mobiles. Ce point regroupe l'optimisation des requêtes, les mécanismes de transaction, la gestion de la mémoire cache, etc. Du fait que les bases de données mobiles font face à des contraintes telles que : l'insuffisance des ressources de traitement et de stockage, les coupures de connexion réseau, la bande passante, etc. Ces contraintes ne sont pas prises en considération dans la technologie traditionnelle de base de données.

Ainsi, le traitement des requêtes dans un environnement mobile nécessite des approches différentes surtout dans les environnements distribués qui sont caractérisés par l'hétérogénéité des nœuds et des liens. Dans un tel environnement, lors de l'exécution d'une requête, la localisation de chaque fragment de données est nécessaire. Après cette localisation, l'exécution de chaque sous-requête est faite par l'exécuteur. Le résultat final est envoyé à l'usager qui a initié la requête. Un mécanisme de traitement des requêtes impliquant la mise en cache des données est nécessaire pour améliorer le système actuel.

Une mémoire cache est une mémoire dans laquelle sont stockées de façon temporaire les données les plus fréquemment utilisées afin d'améliorer la performance et réduire les états d'attente d'un système. Le mécanisme de mise en cache implique de répondre à un certain nombre de questions :

1. Quelle stratégie de gestion de la mise en cache utiliser?
2. Quelle politique de remplacement de cache utiliser pour créer de l'espace en mémoire et garder la base de données dans un état cohérent?
3. Quelles stratégies d'optimisation et de transformation des requêtes utiliser?
4. Quel algorithme de synchronisation utiliser afin de gérer efficacement la mémoire cache?

1.3 Objectif de recherche

Pour répondre aux questions posées précédemment, une approche adéquate de gestion de cache est nécessaire. Bon nombre de protocoles et algorithmes ont été proposés dans la littérature pour la gestion de cache ainsi que de mise en cache et de remplacement de cache.

L'objectif de notre travail est de présenter une nouvelle architecture qui se base sur un ensemble de modèles existants pour les améliorer. Notre approche consiste à introduire un autre type de gestion de cache et l'adapter aux réseaux de téléphonie cellulaire. Nous voulons répondre à ces problèmes en :

1. Proposant une méthodologie d'exécution des requêtes,
2. Proposer une stratégie de gestion de la mise en cache intelligente,
3. Proposer une politique de remplacement de cache efficace.

1.4 Méthodologie

Pour atteindre les objectifs précités la méthodologie que nous utilisons consiste à la conception d' :

- Une nouvelle stratégie de gestion de cache. Qui permet une mise en cache intelligente des requêtes et les résultats des requêtes
- Un eNodeB intelligent qui joue le rôle de gestionnaire de cache. Il fait une copie des données des utilisateurs mobiles qui sont connectés pour faciliter le processus de coopération entre les utilisateurs mobiles.
- Un système de traitement et d'exécution de requête. Il est doté d'un interpréteur de requête qui distingue la partie d'une requête qui est disponible dans le cache et la partie qui n'est pas disponible.

1.5 Organisation du document

Après avoir introduit le contexte et présenté la problématique de ce mémoire, nous avons précisé nos objectifs. Nous organisons le reste du mémoire comme suit :

Dans le chapitre 2, nous présentons une revue de littérature. Dans cette revue, nous présentons les approches et les systèmes existants qui s'inscrivent dans le même angle de recherche que nous. Nous présentons ensuite une synthèse des principaux travaux liés aux techniques de mise en cache et au traitement des requêtes dans un environnement mobile.

Dans le chapitre 3, nous présentons le traitement des requêtes et les différents types de requêtes mobiles. Nous présentons la méthode de mise en cache, et l'architecture de la solution.

Dans le chapitre 4, nous présentons et détaillons les méthodes proposées pour le traitement des requêtes.

Dans le chapitre 5, nous présentons une évaluation de performances par des études de cas. Nous concluons ce document en dressant un bilan des principaux résultats de notre travail et en dressant une liste des perspectives de recherche envisagées.

1.6 Portée

Notre projet consiste à proposer une approche et une solution architecturale pour l'environnement mobile pour le traitement des requêtes pour les bases de données mobiles distribuées. Notre solution sera axée sur la mise en cache des résultats des requêtes des utilisateurs pour mieux répondre à d'autres requêtes dans le futur.

Elle consiste à :

- Proposer une nouvelle architecture de mise en cache des clients mobiles. Cette architecture sera spécifiquement dédiée aux clients mobiles ayant des téléphones intelligents 3G et LTE
- Proposer un algorithme de remplacement de cache.
- Proposer une stratégie d'exécution de requêtes dans l'environnement mobile qui prend en compte la structure de mise en cache proposée et qui réduit le temps d'attente et la congestion sur le réseau.

Notre travail est limité en ce sens que nous limitons notre stratégie à des simples opérations select. De plus, nous supposons que chaque fois qu'un client envoie une requête à un nœud du réseau, les résultats doivent être trouvés.

CHAPITRE II

ETAT DE L'ART

Dans ce chapitre, nous présentons une revue de littérature et un aperçu des résultats et des algorithmes qui ont été utilisés dans le passé. Nous présentons aussi sur les systèmes qui ont été utilisés pour réaliser l'environnement expérimental de notre approche.

Un système de base de données mobile et distribué est conçu pour s'exécuter dans un environnement mobile et distribué. Pour qu'un tel système soit efficace, il faut faire face aux situations exprimées auparavant, à savoir :

1. Gérer la mise en cache des données,
2. Éviter le plus possible les traitements distants des requêtes,
3. Empêcher l'apparition des goulots d'étranglement dans l'exécution des requêtes.

Dans les systèmes répartis, les facteurs qui participent à la dégradation du système sont les accès à distance aux données et les opérations de synchronisation (Notouom, 1996)

2.1 Cache sémantique

Le *cache sémantique* est un ensemble de principes qui traite le stockage, la gestion et le remplacement des éléments dans le cache. Elle est devenue populaire en raison de

son utilisation efficace et l'utilisation des ressources qui se contribuent dans les processus de traitement des requêtes dans le but de réduction du temps de réponse.

Le coût de traitement des requêtes dépend de la latence du réseau qui peut être réduite par une bonne utilisation du cache sémantique combiné avec le traitement des requêtes.

2.1.1 Modèle de S. Dar et al. (1996)

S. Dar et al (S. Dar et al., 1996), ont proposé un modèle sémantique pour la mise en cache du côté client et le remplacement de cache dans un système de base de données client-serveur. Ils ont comparé cette approche avec les méthodes de la mise en cache de pages dans la mémoire centrale et des stratégies de mise en cache par tuple de données dans les tables. Leur méthode a été plutôt efficace parce que le modèle de mise en cache proposé tire ses avantages de trois idées clés :

- La conservation d'une description sémantique des données dans la mémoire cache permet d'obtenir une spécification compacte pour répondre aux requêtes qui ne sont pas disponibles dans la mémoire cache.
- Une bonne politique de remplacement de cache qui maintient la gestion de l'espace de manière adaptative dans le cache. La partie (b) de la figure 2.1 donne une vue du contenu à supprimer dans le cache.
- Maintenir une description sémantique des données mises en cache pour permettre l'utilisation des fonctions qui intègrent la notion sémantique de localité, pour le remplacement de cache.

Une représentation graphique d'utilisation de ses idées clés est présentée dans la Figure 2.1.

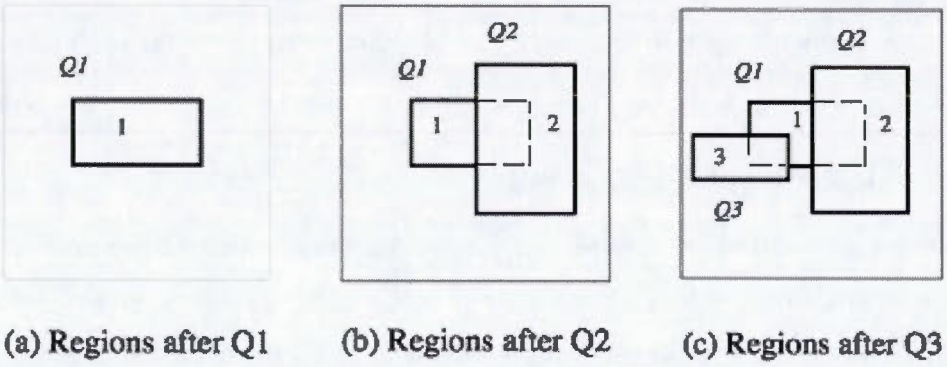

Figure 2.1 Région sémantique d'utilisation (Dar et al., 1996)

Dans ce modèle, le client conserve une description sémantique des données dans sa mémoire cache au lieu de maintenir une liste de pages physiques ou des identificateurs des tuples de données dans les tables. Le traitement des requêtes fait usage de la description sémantique pour déterminer quelles données sont localement disponibles dans le cache et quelles sont les données nécessitant l'accès aux données auprès du serveur. Les données qui ne sont pas disponibles localement constituent le reste de la requête. Une politique de remplacement de cache dicte comment les victimes de remplacement sont choisies

Les faiblesses de cette approche sont :

a) Il n'existe pas un schéma de cache sémantique exhaustif, parce qu'il n'y a pas de schéma de donné ou les données se pas regroupent en fonction de leur type attribut, prédicat, résultat et temps. Ce manque de structure enlève la possibilité de raisonner sur les données en caches

b) La suppression de cache n'est pas efficace. Normalement pour une bonne suppression de cache il faut un schéma de cache exhaustif avec la possibilité de raisonner sur les données cachées et un temps de vie (Time To Live) pour les données.

À défaut d'une mise en cache exhaustive, on peut en déduire que, quelle que soit la politique de suppression de cache elle ne sera pas efficace.

2.1.2 Modèle de Qun et al., (2003)

Qun et al. (Qun et al., 2003) proposent un schéma de gestion sémantique de cache pour accéder à des informations qui dépendent de l'emplacement de l'utilisateur. Ils abordent des aspects sur le comportement mobile et présentent la gestion du cache et la manière d'accéder aux données. Ils proposent également une méthode qui permet de faire la mise en cache des requêtes en fonction de ses composantes : attribut, prédicat, relation et autre.

Cette approche regroupe les données sous forme de graphe et garde en mémoire les résultats des requêtes. Elle utilise l'expression $(Q_R, Q_A, Q_P, Q_L, Q_C)$, où Q_R représente les relations de la requête, Q_A les attributs, Q_P *les prédicats,* Q_L *la position* et Q_C les résultats. La figure 2.2 donne une vue sur l'enregistrement de 3 requêtes (S1, S2 et S3) dans le cache.

S	S_R	S_A	S_P	S_L	S_C	S_{TS}
S_1	Hotel	Hname	$(L_x - 5 \leq hxpos \leq L_x + 5) \wedge (L_y - 5 \leq hypos \leq L_y + 5)$	10,20	2	T_1
S_2	Rest.	Rname, Type	$(L_x - 10 \leq rxpos \leq L_x + 10) \wedge (L_y - 10 \leq rypos \leq L_y + 10) \wedge (6 \leq sched \leq 9)$	-5,15	5	T_2
S_3	Hotel	Hname, Vacancy	$(L_x - 5 \leq hxpos \leq L_x + 5) \wedge (L_y - 5 \leq hypos \leq L_y + 5) \wedge$ (Price<100)	-5,-20	8	T_3

Figure 2.2 Sémantique de mémoire cache de (Qun et al., 2003)

L'approche permet de voir l'interaction d'une requête Q avec le cache représenté par des segments. Si un segment S contient une partie du résultat demandé par la requête Q, alors on divise Q en deux parties : la partie commune qui est la partie retrouvée localement et la partie absente. La partie absente est l'autre partie de la requête qui ne se trouve pas dans le cache mais qu'on doit récupérer à partir d'un serveur distant.

La faiblesse de leur approche, elle prend en compte les données mobiles et des utilisateurs mobiles, tout dépend de la position de l'utilisateur mobile. Quand l'utilisateur et les données sont mobiles, on risque de faire la mise en cache des données qu'on n'aura pas à réutiliser puisque les 2 objets sont mobiles. La mise en cache et la suppression des données dans le cache sont contrôlées et validées par la position des utilisateurs. Tout changement de position affecte les données et les rend désuètes.

2.1.3 Modèle de Bashir et Qadir (2006)

Bashir et Qadir ont présenté une technique sur le traitement et la mise en cache sémantique des requêtes. L'objectif de cette technique est de mettre en cache les requêtes et les projections des requêtes. Dans cette technique ils utilisent un algorithme de comparaison qui est basé sur les opérateurs de base, l'algorithme permet de comparer des prédicats simples. Ils utilisent une approche à 4- niveau hiérarchique d'indexation sémantique (4 - HiSIS) qui permet de faire la correspondance entre la requête et la sémantique.

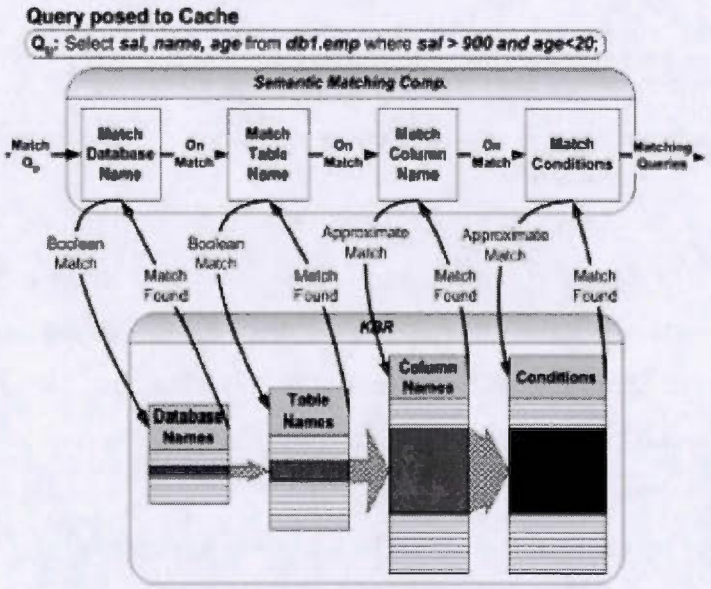

Figure 2.3 4- Niveau d'indexation sémantique hiérarchique (Bashir et Qadir, 2006)

La faiblesse de leur approche est qu'elle n'a aucun processus défini pour rejeter les requêtes erronées. Les règles de correspondance des prédicats sont déterminées que pour les requêtes simples. Il n'y a pas de stratégie définie pour la comparaison disjonctive et conjonctive des prédicats. Il ne fonctionne que pour les requêtes SELECT.

2.1.4 Modèle de Sangwon kang et Kim (2006)

Sangwon kang et Jongwon Kim (Sangwon et Kim 2006) proposent une stratégie de cache sémantique de pré-récupération qui met en cache toutes les données que le client accède. Cette stratégie est fondée sur un modèle client-serveur. Elle fait la mise en cache sémantique et comprend plusieurs stratégies de remplacement de cache. Elle permet de maintenir le résultat des requêtes dans le cache.

Elle permet d'améliorer l'accès aux données en utilisant la mémoire cache. Elle permet également le remplacement de cache. Elle définit un modèle pour traiter les requêtes qui dépendant de l'emplacement de l'utilisateur. Elle analyse les données d'accès et la capacité de la mémoire cache.

Dans cette stratégie, le serveur conserve les fichiers journaux transmises par les utilisateurs, les fichiers journaux sont utilisés pour la prédiction de l'information de pré-chargement sémantique. Le client mobile gère la mémoire cache après qu'une partie de la requête soit vérifiée par le gestionnaire de cache. Si la requête a une partie qui ne peut pas répondre localement, la stratégie formate la requête. Comme il y a une partie de la requête dans le serveur local, le reste de la requête est enregistré dans le référentiel de cache. La mémoire cache remplace son contenu et s'actualise lorsque la pré-lecture de description est transférée au client.

La faiblesse de cette approche est qu'elle utilise une architecture client-serveur ou la majorité des tâches sont exécutées et coordonnées par le serveur. Dans un environnement mobile où le client se déplace d'une station de base à une autre, il est toujours mieux de ne pas centraliser l'exécution. De plus, le journal des transactions se trouve sur les stations de base qui peuvent changer trop souvent risquent de causer un temps de latence dans la transmission de données.

2.1.5 Modèle de Safaeei, et al. (2008)

Dans (Safaeei, et al., 2008), les auteurs proposent un schéma de cache sémantique exhaustif qui est en mesure de répondre à tout type de requêtes telles que JOIN et GROUP.

Item type	Output Attributes (QCL)	Predicates	Pointer To Children nodes	Pointer To parent nodes	Last access time	Pointer to physical location of data

Figure 2.4 Sémantique de mémoire cache de (Safaeei et al., 2008)

Dans leur schéma de cache sémantique, ils proposent une structure de mise en cache des éléments basée sur un modèle graphique formé d'un tableau d'éléments et un algorithme de gestion des éléments du cache. Ils proposent également un algorithme de remplacement des éléments qui sera utilisé pour vider le cache lorsqu'il est saturé. L'algorithme de remplacement fait le meilleur choix des éléments à supprimer en supprimant des éléments résidents calculable. Un élément calculable est un élément qu'on peut déduire à partir des données qui sont dans le cache. Malgré la suppression d'un de ces éléments à partir du cache, il sera toujours calculable à partir des contenus du cache sans nécessiter de communiquer avec le serveur.

Les résultats de la simulation montrent que l'utilisation de ce schéma de cache sémantique améliore grandement la performance de traitement des requêtes. Cette amélioration est mesurable en temps de réponse, bande passante et le nombre de connexions au serveur.

La faiblesse de leur approche est qu'elle ne met pas en cache les requêtes et les résultats dans le schéma de cache, mais des variables pointeurs qui font référence aux données, il n'y a pas de coopération entre les données cachées. La suppression de cache n'est pas efficace, car elle supprime seulement les éléments calculables s'il n'y

a pas d'élément calculable aucune formule n'est proposée pour supprimer les données du cache.

2.1.6 Modèle de N. Qing-jun (2010)

N. Qing-jun propose une approche de gestion de cache. Dans son approche, il considère cinq possibilités de relation entre les segments sémantiques et les requêtes de l'utilisateur mobile. Selon lui, parmi les possibilités qui peuvent se présenter lors du traitement d'une requête, il y a une Possibilité que la réponse d'une requête soit issues complètement des données cachées. Dans cette situation, il n'y a pas d'éléments nouveaux à mettre en cache. Il y a possibilité qu'une certaine partie de la réponse d'une requête soit disponible dans la mémoire cache alors la réponse de la partie probe de la requête est issus des données cachées et l'autre partie du serveur distant. Dans ce cas, on met en cache la partie de la requête qui a été traitée sur le serveur distant. Il y a aussi possibilité où il n'y a aucune donnée du cache qui peut répondre à la requête. Dans ce cas, on fait la mise en cache global de toutes les données de la réponse.

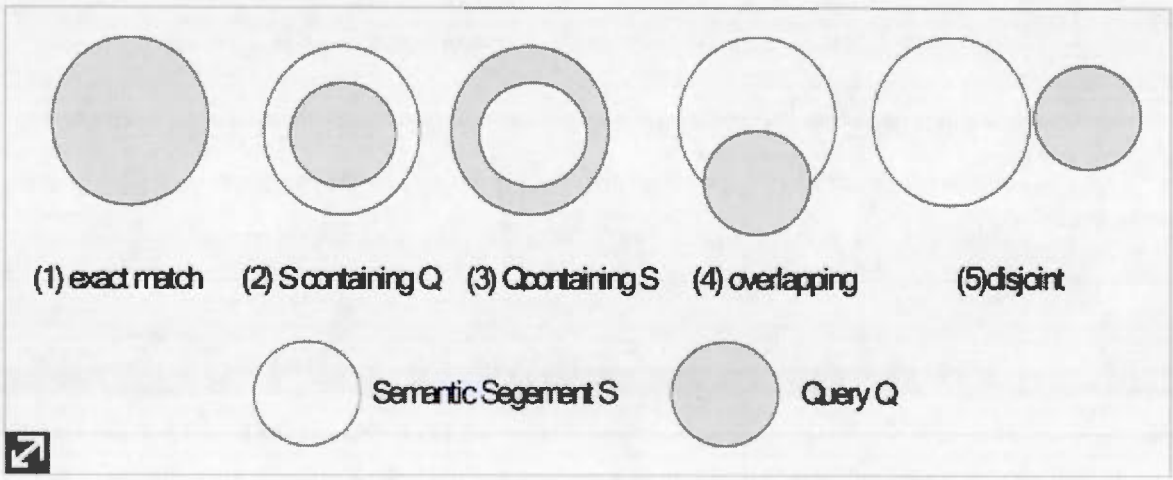

Figure 2.5 Relation entre la requête et segment sémantique de (Qing-jun, 2010)

Dans son approche, il a un système de replacement de cache qui gère les cas de correspondance exacte des contenus et les cas de chevauchement. Lorsqu'il n'y a pas assez d'espace pour stocker des nouvelles données, les données qui ont le plus petit nombre de références sont considérées comme partie à remplacer. Il utilise une durée de vie (TTL) pour chaque objet mise en cache.

La faiblesse de son approche est qu'elle ne propose pas un modèle qui lui est propre. Elle présente le fonctionnement de plusieurs systèmes de mise en cache et critique les paramètres qu'elle trouve inefficaces sans proposer une solution détaillée.

2.1.7 Modèle de Liang Ru -bing et al. (2012)

Dans (Liang et al., 2012), on étend le mécanisme de cache sémantique général en permettant à des clients mobiles de partager leurs caches locaux dans un cache coopératif. Le cache coopératif est l'ensemble des caches locaux des clients.

Ils appliquent une nouvelle approche pour répondre à des requêtes à l'aide d'un mécanisme de cache sémantique coopératif dans l'environnement mobile. Cette méthode permet de réduire le temps de réponse et d'augmenter le débit du serveur de gestion de base de données parce que ce serveur ne reçoit que les requêtes auxquelles le cache ne peut pas répondre.

Ils présentent le fonctionnement du cache sémantique coopératif dans un environnement centralisé et dans un environnement distribué. Ils font la différence entre l'approche centralisée et l'approche distribuée et montrent l'avantage de l'approche distribuée par rapport l'approche centralisée.

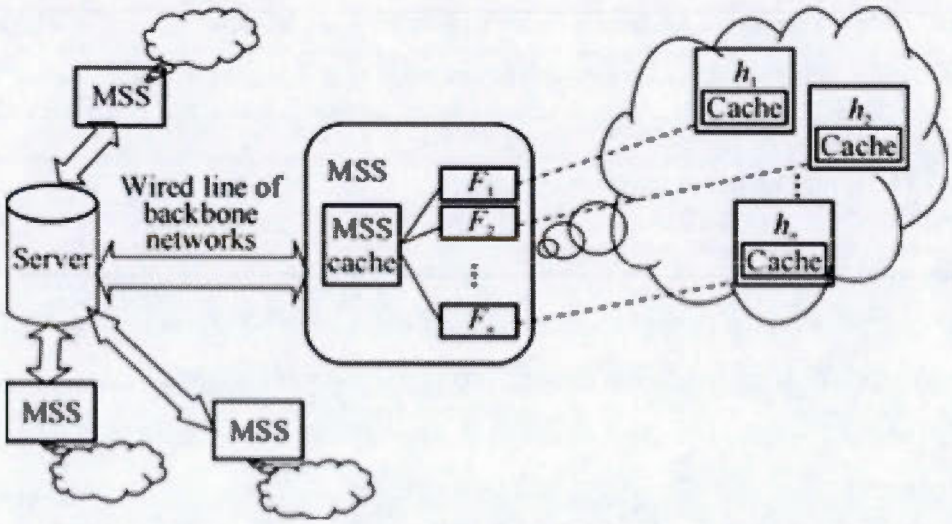

Figure 2.6 Coopération entre les nœuds de (Liang et al., 2012)

Dans cette méthode, ils adoptent un régime centralisé, dans lequel le nœud MSS (Mobile Switching System) joue le rôle de gestionnaire de cache pour le client mobile. MSS est un pont entre l'utilisateur mobile et le réseau câblé. Il permet la performance dans le traitement parallèle et la sauvegarde de la mémoire cache de chaque cache local du client dans une cellule donnée du réseau.

L'approche CoopSC permet aux clients mobiles d'enregistrer des résultats de requête en cache local et de rechercher les requêtes dans le cache de collaboration (MSS) pour obtenir les résultats désirés.

La faiblesse de leur approche est qu'elle ne traite pas en même temps la mise en cache des requêtes et des résultats. Elle propose seulement la collaboration entre les caches sans vraiment mentionner la mise en cache et le remplacement de cache. La figure 2.6 nous montre la coopération entre les nœuds MMS au serveur.

2.2 Traitement des requêtes

2.2.1 Modèle de Marcel Karnstedt, et al. (2003)

Pour surmonter des problèmes de performances et pour réduire les coûts de communication dans le traitement des requêtes, Marcel Karnstedt et al, ont proposé une nouvelle architecture dans laquelle ils décrivent la sémantique du cache basée sur un système d'intergiciel appelé Yacob. Dans cette approche, les entrées de cache sont organisées par régions sémantiques et la mémoire cache elle-même est étroitement couplée avec une ontologie.

Dans leur approche, avec l'utilisation de la mémoire cache on peut essayer de répondre aux requêtes. Le cache renvoie des résultats satisfaisants s'il est en mesure de répondre à la requête, sinon, de nouvelles opérations sont appliquées pour stocker les entrées dans le cache. Dans le cas d'une requête complémentaire non vide ou si aucune entrée de mémoire cache n'a été constatée, la requête est transformée en la traduisant selon un mécanisme de mapping qui traduit la requête et l'envoie à la source correspondante. Les résultats de la requête du cache et/ou les résultats de la source sont ensuite combinés.

La figure 2.7 nous donne une représentation du processus de traitement des requêtes de Marcel Karnstedt

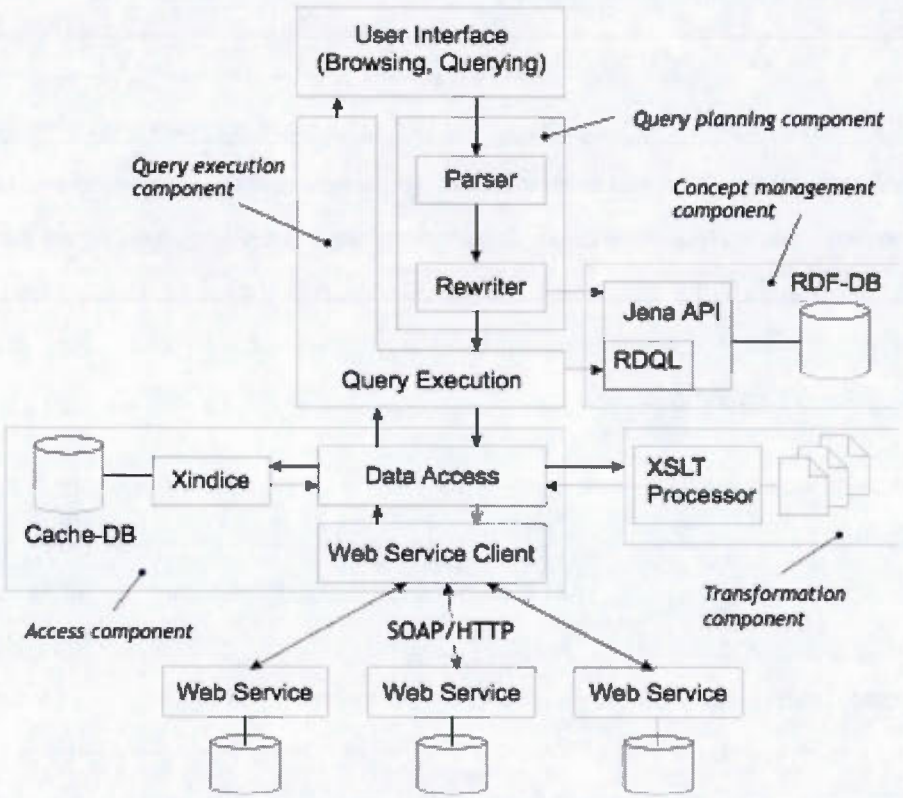

Figure 2.7 Traitement des requêtes de (Marcel et al., 2003)

La faiblesse de cette approche est qu'elle est basée sur 2 sources de données. Si la réponse de la requête n'est pas disponible dans le cache, la 2e partie de la requête se dirige directement vers le serveur. Cette approche ne prend pas en compte la coopération de caches qui stipule que si les résultats ne sont pas disponibles dans le cache local ils peuvent être disponibles dans un cache distant.

2.2.2 Modèle de Amja A. A, et al. (2011)

Dans le modèle de Amja A. A, et al. (2011), une approche a été développée pour un environnement mobile basé sur un réseau 3G. Elle consiste à exécuter des requêtes sur les bases de données mobiles et distribuées. On fait le traitement des requêtes de

type SELECT, UPDATE, DELETE et INSERT, et on utilise la fragmentation horizontale uniquement.

Dans l'approche, ils proposent qu'un client mobile puisse exécuter une requête localement en utilisant des données locales. Sinon, le système peut nécessiter la participation d'un DDN (data directory node) et un ou plusieurs DBN (data base node). Lorsqu'utilisateur mobile envoie une requête, à son DDN plusieurs cas sont considérés :

- Le DDN vérifie dans son répertoire local afin de construire le plan à partir des données qui y sont stockées.
- Le DDN local n'est en mesure d'établir qu'un plan d'exécution de requête partielle de son propre répertoire. Par conséquent, il vérifie avec le DDN à distance pour toute information manquante afin de produire un plan d'exécution complet.

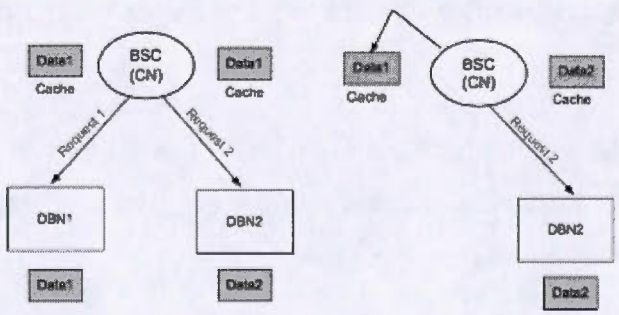

Figure 2.8 Exécution des requêtes en mode déconnecté (Amja et al., 2011)

La faiblesse de leur approche est que cette approche ne prend pas en compte la coopération de cache. La réponse de la requête peut être bien disponible sur un autre nœud sur le même DDN. La technique de coopération entre les caches n'est pas exploitée.

2.2.3 Modèle de Kumar, T.V. Vijay et al. (2010)

Kumar, T.V. Vijay et al. (Kumar, T.V. Vijay et al., 2010) ont proposé une méthode de génération de plans de traitement de requêtes distribuées. Ils déterminent le nombre de plans de traitement de requête possibles en fonction du nombre de sites utilisés par la requête. Ils déterminent ensuite les plans de traitement de requête optimaux parmi tous les plans possibles. Les auteurs ont présenté un plan de traitement de requêtes optimal pour répondre au nombre de plans de requête qui augmente avec le nombre de sites requis. Ils proposent une technique qui permet de réduire l'espace de recherche et d'avoir un modèle de coût pour accéder à la qualité de plans de traitement de requête. Les plans de requête ainsi produits sont optimaux.

La génération d'un plan de requête est basée sur l'heuristique qui définit la proximité relative des données requises pour répondre à la requête de l'utilisateur. La proximité est basée sur le nombre de sites concernés dans un plan de requête. Le plan de requête impliquant moins de sites est considéré et par conséquent est généré. D'autre part, un plan de requête qui vise un grand nombre de sites, n'est pas considéré et n'est pas généré.

La faiblesse de cette approche est que les auteurs se contentent de trouver un plan d'exécution optimal pour la requête en sollicitant tous les nœuds sur le réseau. Les données cachées ne sont pas considérées

2.2.4 Modèle de Komai, Y. et al. (2014)

Dans (Komai et al., 2014), ils proposent deux méthodes de traitement des requêtes pour réduire le trafic sur le réseau et le maintien d'une haute précision du résultat de la requête.

Dans ces méthodes, le nœud émetteur transfère une requête utilisant la formule de géoroutage vers le nœud le plus proche du point spécifié de la requête (un protocole de géoroutage est un protocole basé sur angle de vue adaptatif, pour la transmission

des flux). Le nœud le plus proche de la requête pointe vers d'autres nœuds à proximité du point d'interrogation. Chaque nœud qui reçoit une requête répond avec ses propres informations. Dans ce processus, ils adoptent deux approches différentes : la méthode de l'explosion (EXP) et la méthode spirale (SPI).

Dans la méthode EXP, le nœud le plus proche de la requête inonde les nœuds avec la requête au sein d'une région circulaire spécifique, et chaque nœud qui reçoit la requête répond avec ses informations.

Dans la méthode SPI, le nœud le plus proche de la requête achemine vers l'avant la requête vers d'autres nœuds de manière spirale. Le nœud qui recueille un résultat satisfaisant transmet le résultat au nœud origine de la requête. Les résultats montrent que la méthode SPI réduit le volume de trafic et permet d'atteindre une haute précision des résultats de recherche, en comparaison avec les méthodes existantes.
Dans cette approche, les auteurs se contentent d'acheminer les paquets vers le nœud destinataire en trouvant le chemin le plus court et le plus sûr. Mais en aucun cas, ils ne font attention à la structure des données et la disponibilité des données. Ils ont supposé que les données sont disponibles et interrogent les sources autant de fois qu'une requête est effectuée. Les données en cache ne sont pas considérées. La coopération entre les données des différentes sources n'est pas exploitée.

Tenant compte de l'inefficacité des modèles existants comme : le modèle de A. A. Safaeei, et al. (2008) qui mettait en cache des variables pointeurs qui font référence aux données, mais pas les requêtes et non les résultats dans un schéma de cache. Le modèle de Qun Ren et al., (2003) qui propose une approche basée sur des utilisateurs mobiles et des données mobiles. Or on sait qu'avec les données mobiles on risque de faire la mise en cache des données qu'on n'aura pas à réutiliser pour ne citer que

ceux-là. Selon Poularakis et al. (2014) une bonne stratégie de traitement des requêtes doit être étroitement liée avec un processus de mise en cache. Alors, pour résoudre ces problèmes, nous proposons notre approche. Elle consiste à introduire une nouvelle une stratégie de gestion de cache intelligente et une méthodologie d'exécution des requêtes.

CHAPITRE III

ARCHITECTURE PROPOSÉE

Dans ce chapitre, nous présentons une architecture de la solution des problèmes mentionnés dans le chapitre I. Nous présentons les systèmes de mise en cache, la structure de mise en cache des éléments de la requête et de la réponse de l'utilisateur. Nous présentons le système de remplacement de cache qui est formé de la combinaison de plusieurs techniques de remplacement de cache existantes.

Finalement, nous présentons le gestionnaire de cache coopératif qui effectue la coopération entre les différentes données cachées du côté client, son fonctionnement et son interaction dans la résolution des requêtes des utilisateurs mobiles.

3.1 Le système de mise en cache

Dans l'environnement mobile, la mémoire cache est généralement utilisée pour diminuer la latence d'accès aux données, la tolérance aux pannes, la sécurité des données et pour réduire le trafic sur le réseau. (Godfrey et Gryz, 1997).

La mise en cache des fichiers statiques permet d'économiser du temps au chargement des fichiers ciblés dans le futur. Dans un cache sémantique, la description sémantique ainsi que les données effectives sont stockées dans la mémoire cache. Les requêtes entrantes des utilisateurs sont comparées avec la sémantique des requêtes exécutées précédemment. (Qun et al., 2003), (Dar et al., 1996). La mise en cache des éléments de données fréquemment consultées du côté client est une technique efficace pour

améliorer la performance dans un environnement mobile (Barbará et Imieliński, 1994)

Une mémoire cache est une mémoire qui enregistre temporairement une copie des données provenant d'une autre source afin de diminuer le temps d'accès à ces données. Pour traiter les requêtes des bases de données mobiles et réparties, le recours aux données mises en cache est considéré comme une alternative. Normalement, la mémoire cache est organisée de trois façons : comme une page de mémoire, un *tuple* de mémoire, et un cache sémantique.

- Un *tuple* de mémoire est une mémoire associative partagée via laquelle des processus distincts peuvent communiquer, conserver des données et coordonner leurs actions. C'est une liste de données typées et de longueur arbitraire.
- Une page de mémoire ou une mise en cache de page est à bien des égards analogue à la mise en cache d'un tuple. La différence principale étant qu'avec la mise en cache, le client d'un tuple cache est maintenue en termes d'individu tuple alors que dans la mise en cache de page ils sont des pages entières.
- Le cache sémantique est un tableau de données regroupé par une structure « relation, attribut, prédicat ». Il permet de raisonner sur les données mises en cache.

3.2 Architecture proposée

L'architecture que nous proposons est un système de cache sémantique. C'est une amélioration de la proposition de (Safaeei, et al., 2008), (Qun et Kumar, 2003).

Cette architecture met en cache les requêtes et les résultats des requêtes précédentes et s'en sert pour répondre à de nouvelles requêtes. Elle est basée sur la réutilisation des contenus du cache.

Nous utilisons une architecture dans laquelle on fait la mise en cache et la réutilisation des données cachées afin de réduire le temps d'exécution des requêtes et de faciliter l'accès aux données en tout temps. Pour ceux-là, nous proposons un schéma de cache. Un schéma de cache sémantique est un ensemble de règles pour le stockage, la gestion et le remplacement des éléments dans le cache.

Avec le cache sémantique, les clients mobiles conservent à la fois une description sémantique de la requête et les réponses associées aux requêtes précédentes dans le cache. Cette structure permet de répondre à une nouvelle requête sans avoir besoin de communiquer avec le serveur. Si la requête ne peut être que partiellement répondue, cette requête doit être découpée en deux parties : une partie qui est localement disponible dans le cache et le reste de la requête (Dar et al., 1996). De ce fait, pour construire notre base de données cachée, nous utilisons un algorithme qui fait la mise en cache lors de chaque requête.

L'algorithme effectue un contrôle d'admission de cache sur les résultats de la requête pour décider s'il peut mettre en cache les données de la requête. Il met en cache les données des requêtes dans le cas où toutes les données ou une certaine partie des données de la requête n'existent pas encore dans le cache. L'algorithme doit permettre aux clients de stocker les résultats des vieilles requêtes et de les utiliser toute ou une partie de ces résultats si les paramètres de celles-ci sont proches ou incluent les paramètres de la nouvelle requête pour répondre à ses requêtes.

La figure 3.1 faite une représentation graphique d'un utilisateur mobile est sa base de données cachée.

S	Relation	Attribut	Prédicat	Résultat	Temps d'accès
S1	Employée	(Enamc)	30<age<40	2	T1
S2	Employée	(Enamc, Budget)	Budget >10k	4	T2
S3	Projet	(Enamc, sal)	Age<= 28	3	T3
S4	Employée	(Enamc, age)	Age >45	8	T4

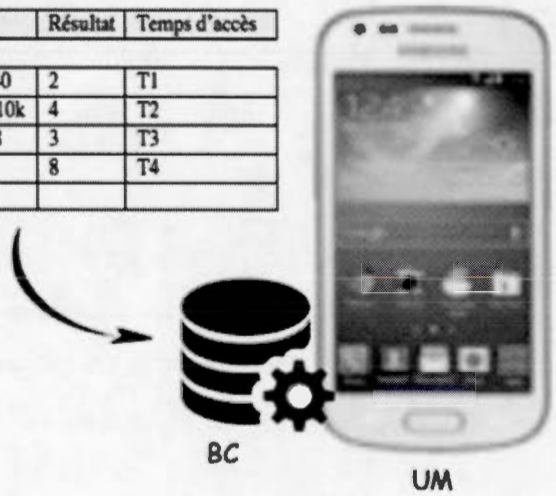

Figure 3.1 Utilisateur mobile et sa base de données cachée

La structure de notre architecture de mise en cache comprend trois grands axes :

1. La mise en cache
2. Le remplacement du cache
3. Le gestionnaire de cache

Dans ce qui suit, nous allons les décrire en détail.

3.2.1 La mise en cache

La mise en cache consiste en une décomposition de la requête et de la réponse de l'utilisateur mobile en différents éléments qui la composent (relation, prédicat, attribut et résultat).

Elle détermine les composants de la requête pour vérifier l'existence de ses éléments dans le cache ainsi que la valeur de chaque élément afin de déterminer si ces éléments doivent être sauvegardés dans le cache.

C'est un schéma exhaustif qui doit être capable de répondre à tous les types de requêtes telles que les jointures (JOIN) et les groupes (GROUP BY).

C'est une représentation des différents éléments que le système va mettre en cache. Elle est divisée en deux parties : a) le cache des requêtes et b) le cache des résultats.

3.2.1.1 Le cache des requêtes

C'est une représentation des différents éléments que peut contenir une requête. Cette structure est présentée sous forme d'un tableau. Elle met en cache toutes les séquences de la requête.

Elle va d'abord diviser la requête en relations, attributs, et prédicats pour ensuite insérer les éléments dans le tableau en fonction de leur groupe. Dans le cas où une requête ne spécifie pas des attributs et/ou des prédicats, elle va mettre en cache la structure de la table ou des tables sur lesquelles la requête a été exécutée. Plusieurs cas peuvent se présenter :

- Cas 1 : Exécution d'une requête quand les attributs, les prédicats et les relations sont précisés. Exemple :

Select e_ID eName, Sal, ***from*** employe ***where*** Age >30

$\mathbf{Q_R}$ = employe, $\mathbf{Q_A}$,= e_ID, eName, Sal, $\mathbf{Q_P}$ = Age > 30

$$Q = < Q_R, Q_A, Q_P, Q_C >$$

Dan ce cas, on définit une requête Q comme : Q_R, Q_A, Q_P, Q_C » où

Q_R = Relation

Q_A = Attribut

Q_P = Prédicat

Q_C = Résultat

Ainsi la décomposition de chaque élément d'une requête se fait pour faciliter la mise en cache et la correspondance à la prochaine requête. Voici un exemple de décomposition de la requête précédente :

S	Relation	Attribut	Prédicat	Résultat
S1	Employe	{e_Id, eName, sal}	age>=30	*c1

Tableau 3.1 Mise en cache d'une requête avec précision.

- Cas 2 : Exécution d'une requête dans le cas où les attributs, les prédicats et les relations ne sont pas précisés. Exemple :

Select * from employe;

Q_R = employé, **Q_A**= ID_Emp, Nom, prénom, age, sal, département **Q_P** = NULL

Dans une telle situation, nous prendrons en compte tous les attributs de la table spécifiée sans se soucie des prédicats.

S	Relation	Attribut	Prédicat	Résultat
S4	Employé	{Nom, prénom, age, sal, département}	NULL	*c4

Tableau 3.2 Mise en cache d'une requête sans spécification des prédicats.

3.2.1.2 La structure des réponses cachées

La mise en cache des éléments de réponse d'une requête est l'un des objectifs de notre architecture. Pour chaque nouvelle requête effectuée auprès d'une base de données distante, notre architecture fait la mise en cache de la requête ainsi que sa réponse. Elle crée une variable pointeur faisant référence à l'adresse de la réponse dans la table des requêtes afin de préciser l'emplacement des éléments de la réponse.

Cas 1 : Exécution d'une requête quand des contraintes ont été appliquées à la requête. Nous avons le résultat qui suit :

```
Select e_ID eName, Sal, from employe where e_Age >30
```

C1		
Id_Emp	**Nom**	**Sal**
PM001	Pierre	49,500
CL001	Claude	51,000
MT001	Michel	35,000
LF001	Louis	42,000
JW001	Julien	48,000
MT002	Matin	32,000

Tableau 3.3 La mise en cache des résultats d'une requête avec précision

Dans ce tableau, nous faisons la mise en cache seulement des attributs qui ont été spécifiés dans la requête. Tout autre élément non mentionné dans la requête n'est pas considéré.

Cas 2 : Quand il n'y a pas de contrainte ni spécification appliquée sur la requête.

```
Select * from employé;
```

C4					
Id_Emp	**Nom**	**Prénom**	**Âge**	**Sal**	**Département**
PM001	Pierre	Martin	30	49,500	Info
OJ001	Orival	Jean	28	53,250	Admin
CL001	Claude	Marcelin	32	51,000	Admin
RF002	Riviére	Frantz	25	60,000	Anal
MT001	Michel	Tremblay	51	35,000	Exec
LF001	Louis	Fritzler	43	42,000	Info
JW001	Julien	Wislain	42	48,000	Analyse
AS001	Augustin	Sandy	26	39,999	Admin
PV001	Pierre	Vanessa	27	28,000	Exec
MT002	Matin	Tasero	50	32,000	Direct

Tableau 3.4 La mise en cache des résultats d'une requête sans précision

Dans ce tableau, nous faisons la mise en cache de la structure complète de la table et les différents éléments qu'il contient sans restriction.

Notre approche fait une mise en cache de toutes les réponses venant d'une source de données externe sauf dans les cas suivants :

- Quand une réponse est déterminée invalide : une réponse est invalide quand la réponse contient que des valeurs résidant dans le cache ou si la requête n'aboutit pas à une réponse;
- Quand la mémoire cache est saturée : dans ce cas, la mise en cache sera impossible. Alors on fait appel au processus de remplacement de cache avant de débuter l'activité de mise en cache.

- Quand la réponse d'une requête n'est pas spécifiée ou comporte un grand nombre d'éléments de données : la réponse est considérée comme une réponse qui risque de perturber le bon fonctionnement de la cache mémoire. Cette réponse ne sera pas cachée.
  Ex. : **select * from** Étudiant;

3.2.2 Le remplacement du cache

Le remplacement de cache est un système qui effectue des calculs de validité et de déductibilité sur les données du cache. Il permet de libérer de l'espace utilisé dans le cache par des données jugées désuètes ou inopportunes dans le but de les remplacer par des données plus susceptibles d'être réutilisées dans le futur.

En raison de la taille limitée de l'espace de stockage du cache, en tenant compte des données à cacher, il est impossible d'envisager une mise en cache sans une politique de remplacement de données (Yin et al., 2003). Par conséquent, le recours à des algorithmes et formules pour libérer de l'espace mémoire en cache est indispensable.

De manière traditionnelle, pour la suppression des données dans le cache on utilise l'algorithme LRU (Least Recently Used) pour remplacer la ligne la moins récemment utilisée ou on utilise l'algorithme LFU (Least Frequently Used) qui remplace le

contenu le moins fréquemment utilisé. Mais, il existe bon nombre de politiques de remplacement de cache qui recourent à différents facteurs pour remplacer les données dans le cache telles que le temps d'accès, le temps d'entrée en cache, la fréquence d'utilisation des données cachées, etc. Dans notre, architecture nous proposons un algorithme qui fait une vérification sur l'espace de stockage disponible dans le cache. Nous utilisons un contrôle d'admissibilité des éléments à mettre en cache qui vérifie la possibilité de suppression de cache en fonction des données calculables ou par un algorithme de remplacement de cache. Notre algorithme vérifie les paramètres suivants :

- La disponibilité d'espace dans le cache avant chaque mise en cache. Afin d'assurer si la mise en cache nous effectuons une surpression de données dans le cache. Si la suppression est obligatoire par manque d'espace, alors nous faisons une vérification pour déterminer la présence des données calculables.

- Les données calculables ou déductibles. Un élément est défini déductible ou calculable c'est quand même après la suppression de cet élément nous pouvons le reconstitué à partir des différents éléments de données qui existent dans la base de données.

Dans tous autres cas, si l'espace de stockage est saturé et il ne contient pas de donnée calculable l'algorithme LRU (Last Recently Used) sera utilisé pour remplacer les données dans le cache.

L'algorithme suivant est présenté en vue de donner une idée de la justification de la méthode proposée.

```
CacheItemreplacement(C)
        */Cachesup = élément à supprimer*/
        */Cachesel = élément calculable*/

{
      */pour toute donnée enregistrée dans le cache */
For all Si, Si ∈ C
            */s'il existe des données et elles sont calculables*/
If (Si ≠ Null AND Calculable (Si))
        Then {
        */sélectionnez les données calculables qui existent dans le cachesel */
        cachesel ← Si;
        */si aucune solution calculable n'existe dans le cache */
          If (count(Cachesel)==0) then

        */alors, supprimer les données les moins récemment utilisées */

            Cachesup ← LRU(C);
          Else
        */si une ou plusieurs solutions calculables existent dans le cache */
          If (count(Cachesel)>1) then
        */alors, supprimer les solutions calculables les moins récemment utilisées */

            Cachesup ← LRU(Cachesel);
        */si une ou plusieurs solutions calculables existent dans Cachesup */
          If (count(Cachesup)>1) then

            Cachesup ← LRU(Cachesup);
    }
  }
```

Figure 3.2 Algorithme de remplacement de cache

La figure 3.4 donne une représentation de notre système de remplacement de cache. Dans la partie A, les éléments R1, R2 et R3 sont cachés par ordre respective t1, t2 et t3. Dans la partie B, le système vérifie les données calculables. Dans la partie C, la suppression des données calculable se fait pour libérer de l'espace dans le cache.

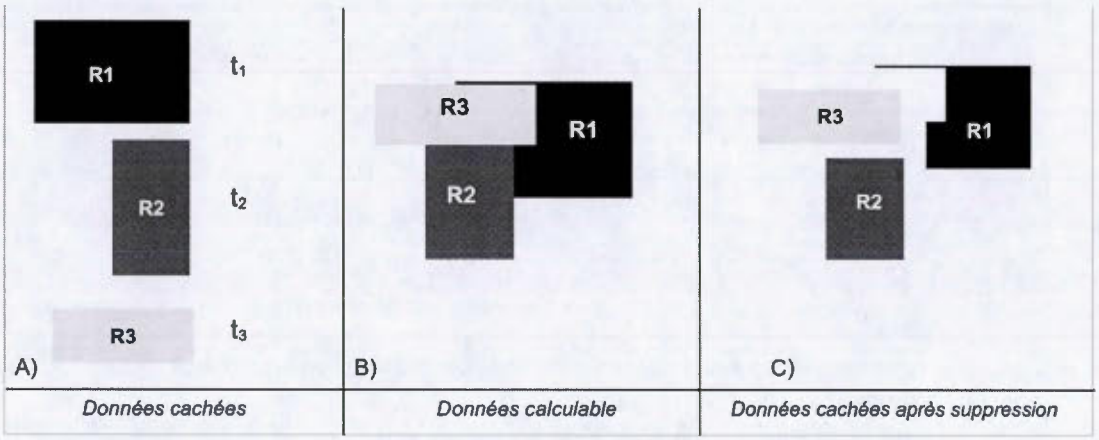

Figure 3.3 Remplacement de cache

3.2.3 Le gestionnaire de cache

Le gestionnaire de cache coopératif c'est un centre de réplication de données créer au niveau du BTS/eNodeB. Son rôle est de faire une réplication des données des utilisateurs mobiles qui sont connectés à la station de base.

Selon les principes de fonctionnement de l'architecture du réseau LTE, un utilisateur mobile est lié à un nœud eNodeB et peut se connecter à un et un seul de ces nœuds pour communiquer avec les autres utilisateurs. Sur ce, dans notre architecture, nous proposons que chaque utilisateur mobile qui se connecte à un eNodeB doive partager son cache et qu'une réplication de ses données cachées doit se fait sur la station de base. La réplication de données des différents nœuds de la cellule crée une base de données cachée au niveau de l'eNodeB. La composition de toutes ces données va former le centre de gestion de cache (CGC). L'approche d'un centre de gestion de cache a été utilisée par : (Lübbe et al., 2011). (Olston et Widom, 2002), (Elfaki et al., 2013). Le centre de gestion de cache est construit à partir des données disponibles dans le cache de chaque utilisateur et il est utilisé comme cache distant pour répondre aux requêtes des utilisateurs mobiles. Il permet à tous les utilisateurs d'avoir accès

aux données cachées par les autres utilisateurs. La disponibilité des données sur un utilisateur de la cellule contribuera à répondre aux requêtes effectuées dans la cellule.

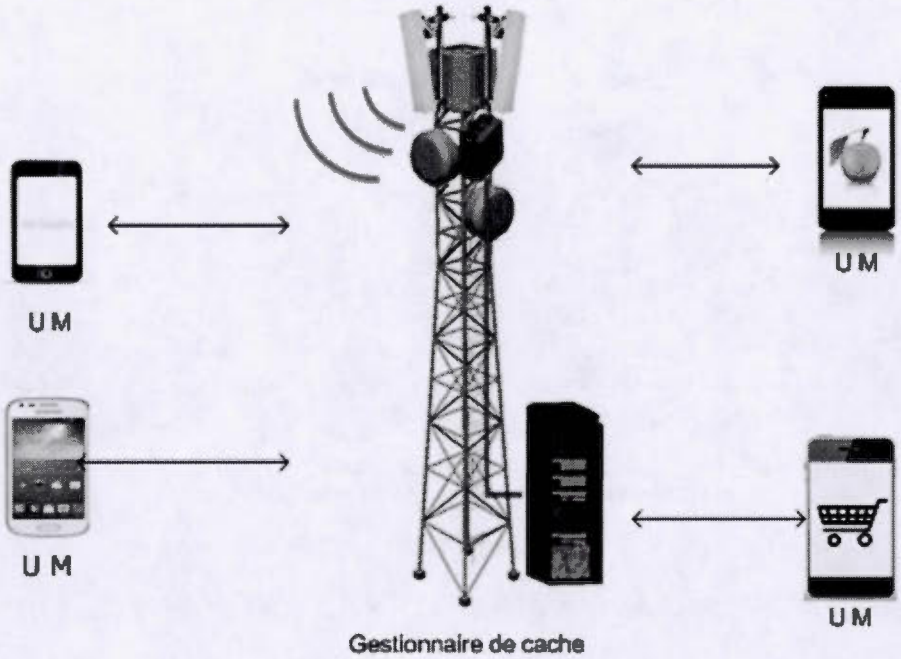

Figure 3.4 Centre de gestion de cache

Nous proposons d'utiliser une réplication de données au niveau de la station de base dans le but d'augmenter la disponibilité des données dans la cellule afin que même dans le cas où l'utilisateur mobile est partiellement connecté ses données soient toujours disponibles. Nous proposons d'utiliser un système de réplication à faible coût et qui garde en mémoire les données tant que l'utilisateur est connecté à la même station de base. Quand un utilisateur mobile change de cellule, un transfert de responsabilité est fait et la suppression de données répliquées de l'utilisateur est faite. Notre architecture fonctionne de manière centralisée : l'eNodeB sélectionné par le client mobile joue le rôle de gestionnaire de cache centralisé. Dans l'exécution d'une requête, la priorité est accordée au cache local. La réponse de toutes requêtes non

satisfaites localement doit passer d'abord par le centre de gestion de cache avant même de tenter toute possibilité de propagation de la requête.

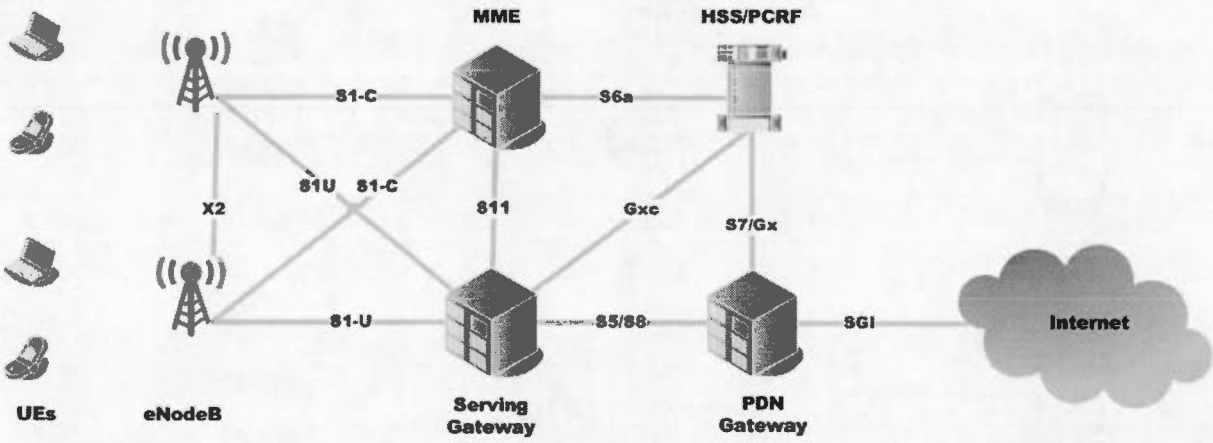

Figure 3.5 Architecture du réseau LTE (rcrwireless news mai 2014)

CHAPITRE IV

LE TRAITEMENT DES REQUÊTES

Le traitement des requêtes dans l'environnement mobile est vaste et assez complexe. Ces dernières années ont vu la popularité des téléphones intelligents et les tablettes. Le trafic mobile de données généré en 2014 était près de 30 fois la taille de l'ensemble du réseau Internet mondial en 2000, et le trafic mobile mondial continuera de croître à un taux de croissance annuel composé de 57 % de 2014 à 2019. En d'autres termes, il va augmenter de 10 fois et atteindre 24,3 Exaoctets (Eo) par mois d'ici 2019 (Cisco Visual Networking Index). Cette énorme quantité de trafic de données dégrade la qualité du service et crée une énorme pression sur le spectre limité de réseaux cellulaires. [AT&T Throttling Their Smartphone Customers]

Dans ce chapitre, nous présentons l'architecture de traitement des requêtes, la description des différents éléments de cette architecture, le traitement des requêtes et les différentes possibilités d'exécution des requêtes. Nous faisons également une présentation sur les différentes stratégies d'exécution de requêtes en tenant compte des différentes sources de données.

4.1 Le traitement des requêtes

Le traitement d'une requête permet de solliciter une base de données en vue d'obtenir un élément de réponse pour cette requête. La base de données sollicitée lors d'une requête peut être centralisée ou distribuée. Une base de données distribuée est une collection de bases de données localisées sur différents sites, généralement distants, mises en relations les uns avec les autres à travers un réseau et perçue pour

l'utilisateur comme une base de données centralisée. Elle permet de rassembler des données, disséminées dans le réseau sous forme d'une base de données intégrée.

4.1.1 Éléments de l'environnement de traitement des requêtes

Dans le traitement des requêtes dans un environnement mobile, on fait face à des entités comme les données, les utilisateurs, etc. Ces entités-là peuvent être statiques ou mobiles. Par conséquent, 4 situations peuvent se présenter :

- Base de données mobile et utilisateur fixe : c'est la catégorie des requêtes émises à partir de terminaux fixes qui interrogent des données liées à des objets mobiles;
- Base de données mobile et utilisateur mobile : catégorie des requêtes émises à partir de terminaux mobiles qui interrogent des données liées à des objets mobiles.
- Base de données fixe et utilisateur mobile : catégorie des requêtes émises à partir de terminaux mobiles qui interrogent des données liées à des objets fixes.
- Base de données fixe et utilisateur fixe : catégorie des requêtes émises à partir des terminaux fixes qui interrogent des données liées à des objets fixes.

Dans notre architecture, nous nous concentrons sur les bases de données fixes et distribuées avec des utilisateurs mobiles. Nos données sont fixes du fait que nos données sont enregistrées sur des serveurs fixes ou des préenregistrées sur des BTS et que nos données n'ont rien à voir avec la position géographique du client. Précisons que le déplacement d'un utilisateur à travers d'autres cellules n'affectera pas la valeur de retour des requêtes sauf que le temps de traitement peut varier en fonction de la disponibilité des données. Ca veut dire les utilisateurs sont mobiles et que la résolution des requêtes ne dépend pas de la position de l'utilisateur.

Selon le mode de fonctionnement des utilisateurs mobiles, dans l'environnement LTE au moment de déplacement d'un utilisateur dans une zone de couverture, il est desservi même BTS quelque soit sa vitesse de déplacement à travers les cellules et quelques soit sa position (Madria et al., 2002). Quand un utilisateur mobile arrive dans une situation où il doit changer de cellule un transfert de responsabilité est fait entre un eNodeB courant et le nouveau eNodeB qui prend en charge l'utilisateur. Une fois connecté, l'utilisateur est sous le contrôle de ce BTS jusqu'à ce qu'il le quitte ou qu'il soit en mode d'inactivité. Ce mode est caractérisé par l'absence d'échange d'informations. Un utilisateur peut être dans différents états : connecté, semi-connecté et déconnecté. En tenant compte du fait que les utilisateurs mobiles peuvent changer d'état, cela implique que le traitement des requêtes doit s'adapter à ces différents états.

En tenant compte de ses différents facteurs, nous pouvons déduire que le coût de traitement d'une requête dans l'environnement mobile dépend de plusieurs paramètres qui peuvent faire augmenter ou diminuer le temps de traitement voire même à l'insatisfaction des réponses. Pour pallier à ces problèmes, nous présentons une nouvelle approche. L'objectif n'est pas d'éliminer ces contraintes, car elles sont inévitables et font partie intégrante de l'environnement mobile, mais de trouver un moyen de maximiser les performances dans l'environnement mobile et de donner des résultats satisfaisants. Dans notre approche nous nous contentons de proposer un mécanisme qui retourner les réponses précises aux requêtes et de réduire le temps de réponse.

En utilisant la sémantique sur les éléments stockés dans le cache, les clients mobiles sont capables de raisonner pour déterminer si une requête peut être totalement répondue à partir du cache ou quelle partie de la requête peut être répondue, et quelles sont les données manquantes. (Qun et al., 2003).

L'idée de la mise en cache sémantique est que le client conserve dans le cache la description sémantique et les résultats des requêtes précédentes (Qun et al., 2000). Une telle approche est très bénéfique lors qu'un utilisateur mobile est en mode semi-déconnecté. L'appareil mobile sera en mesure de répondre à un certain nombre de requêtes et sera obligé de traiter les requêtes en fonction des données stockées dans le cache. Les politiques de routage et de mise en cache doivent être conçus conjointement (Poularakis et al., 2014) afin que chaque requête et sous-requête puissent être dirigée vers la base de données concernée. Un interpréteur de requêtes sera en mesure de distinguer la partie cachée dans une requête (Remainder) et la partie manquante (Probe) (Marcel et al., 2003), (Franklin et al; 1996). Un distributeur de requêtes (Map Query) sera en mesure de distribuer chaque sous-requête aux sources de données qui peuvent le traiter. Ça peut être le cache local, le cache distant ou une base de données distante.

4.2 Architecture de traitement des requêtes proposée

Cette architecture a été choisie pour résoudre les problèmes que nous avons mentionnés dans les chapitres précédents. Elle est composée des éléments suivants : l'Utilisateur mobile (UM), le processeur des requêtes (Q. Processor), la base de données cachée (BC) le Gestionnaire de cache (GC) et la base de données Serveur (BS). Notre architecture de traitement des requêtes est représentée dans la Figure 4.1

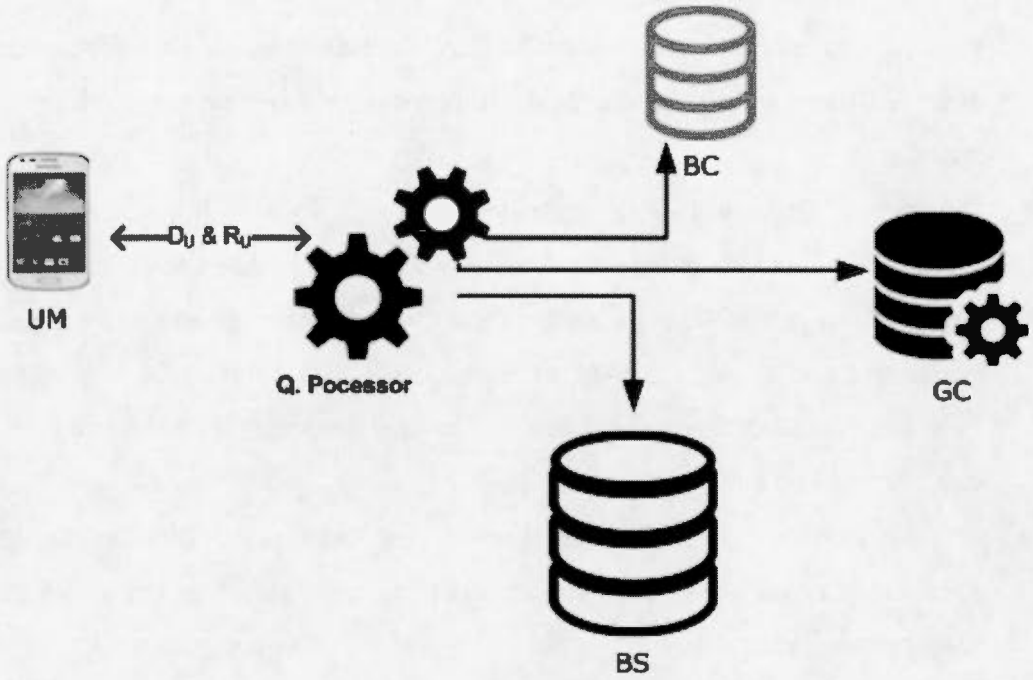

Figure 4.1 Architecture de traitement des requêtes

Au niveau du processeur des requêtes (Q processor), il y a une suite d'activités de vérification et de contrôle. Elles sont montrées dans le diagramme de la figure 4.2.

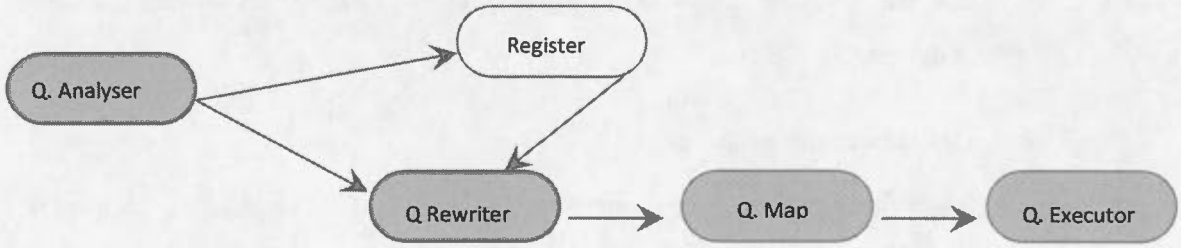

Figure 4.2 Digramme d'activité de traitement des requêtes

- Q. Analyzer : vérifie le contenu de la requête, il détermine les différents composants (les prédicats, les attributs et les relations) en vue d'interpréter la

requête pour les autres composants de l'architecture. Il utilise l'algorithme de (Ali et al., 2010) conçu sur un schéma de graphe sémantique basé sur l'indexation des requêtes.

- Register : c'est l'interface de communication avec la base de données cachée. Il vérifie le contenu de la base de données en cache et garde une description ajout des contenus présents dans le cache. Quand il reçoit une requête de l'analyseur, il compare les relations, attributs, prédicats de la requête avec les éléments de la base de données cachée en vue de déterminer si le cache peut répondre la requête partiellement ou entièrement.
- Request Rewriter : il décompose la requête en fonction des données disponibles dans le cache et dans les bases de données distantes. Ensuite, il effectue la réécriture de la requête.
- Request Map : il distribue les sous-requêtes vers les sources d'exécution. Les requêtes qui vont être traitées localement sont envoyées à la base de données cachée et les requêtes qui vont traiter sur les serveurs distants seront envoyées aux bases de données distantes.
- Request Executor : il exécute la requête auprès de la base de données spécifiée en vue d'obtenir le résultat attendu. Quand les résultats des sous-requêtes sont arrivés, il fait la fusion des différentes parcelles des résultats en un seul résultat et l'achemine vers le RU

4.3 Le traitement des requêtes

Notre architecture d'exécution de requêtes prend en compte la requête du client et fait une analyse sur la requête dans le but de dissocier le *probe query et le remainder query* de la requête afin de pouvoir réécrire la requête de manière optimisée pour réduire le temps d'exécution. Cette technique a été utilisée dans le passé par Marcel (Marcel et al., 2003).

La mobilité exige qu'un utilisateur mobile doive être en mesure d'accéder aux données souhaitées à partir de n'importe quelle cellule (Madria et al., 2002). Pour se faire, un nouveau lien de communication est établi à chaque fois qu'un utilisateur mobile_change de cellule. Ce processus de transfert de cellule est transparent et est responsable du maintien de la connectivité. Dans un tel processus, chaque eNodeB va stocker des informations telles que le profil de l'utilisateur, les fichiers de connexion, et les droits d'accès de l'utilisateur mobile.

Dans ce contexte, une demande (D) est créée lorsqu'un utilisateur mobile effectue une requête pour un élément de donnée. Il y a 3 sources possibles pour satisfaire la demande d'un client :

- La base de données cachée (BC) peut répondre partiellement ou complètement à la demande du client à partir des données disponible dans le cache.
- Le gestionnaire de cache (GC) qui est une union de toutes les données cachées par les différents nœuds du réseau. Il a la capacité de répondre à une requête partiellement ou complètement.
- La base de données du côté serveur (BS) où on récupère le reste des tuples manquants ou bien la totalité des données de la requête si aucun contenu de la requête n'est disponible dans le cache ni dans le gestionnaire de cache.

Ces sources de données sont accessibles sur la base du classement de la demande. Notre algorithme d'exécution détermine les parties d'une requête qui peuvent être répondues dans le cache local en utilisant le schéma de graphe sémantique basé sur l'indexation des requêtes de l'algorithme de (Ali et al, 2010). Le gestionnaire de cache et/ou le serveur de base de données sont sollicités afin de retourner le résultat de la requête finale.

La requête de l'utilisateur venant de l'appareil mobile et s'achemine vers l'Analyseur de requêtes (Q. Analyser). L'Analyseur des requêtes vérifie le contenu de la requête et décompose les éléments de la requête en attributs, prédicats et relations. Il utilise l'algorithme de (Munir et al., 2010) conçu sur un schéma de graphe sémantique basé sur l'indexation des requêtes. Il envoie la requête au Registre et au Request Rewriter. Le Registre vérifie la disponibilité des données de la requête dans le cache selon la structure du graphe de mise en cache et communique les résultats au Request Rewriter.

Si le contenu du cache peut répondre à la requête, la requête est acheminée vers le Map Q. Le Map Q l'achemine vers l'exécuteur des requêtes (Q. Executor). Le Q Executor l'exécute auprès de la base de données cachée. Si une partie de la requête est satisfaite par le cache, le Registrer informe le Request Rewriter des résultats. Le Request Rewriter réécrit la requête en fonction des résultats du Registre et achemine les nouvelles sous-requêtes vers le Q Map. Dans le Q. Map, les éléments du Registre vont former une seule sous-requête. Le reste peut former une ou plusieurs sous-requêtes. Le Q Map va acheminer les requêtes au Q. Executor. Le Q Executor achemine les sous-requêtes aux sources appropriées en fonction des spécifications faites par le Q Map et ramène les résultats des sous-requêtes qu'il va regrouper en un résultat unique auprès de l'utilisateur. Ce processus est décrit dans l'algorithme ci-dessous (Figure 4.3).

```
*/Q= Requête de l'utilisateur
C = Base de données cachée
Cm = Gestionnaire de cache
Db = Base de données
Result = Résultat de la requête
Flagcache = donnée à cacher */
*/pour toutes requêtes */
```

```
FOR all Q ε (C Cm Db)
*/si généralement Q est dans C */
IF (G (Q) = C) then {
        Result ß QC
        }
ELSE {
 */si généralement une partie de Q est dans C */
 IF (G (Q) ≈C) then {
        Result ß QC,, QCm;
        flagcache ß QCm;
        }
        ELSE
        reslut ß QC, ,QCm, QDb;
        flagcache ß QCm, QDb ;
        }
Result;
}
```

Figure 4.3 Algorithme de traitement des requêtes

4.4 Traitement des requêtes

Nous présentons notre stratégie pour l'exécution des requêtes. Pour alléger l'explication, nous présentons le traitement des requêtes SQL de type **select.** Plusieurs cas se présentent :

- Requête locale : une requête locale est une requête qui affecte une ou plusieurs tables dans une base de données qui peut être satisfaite par la base de données cachée.

- Requête distante : une requête distante est une requête qui affecte une ou plusieurs bases de données qui ne sont pas disponibles localement. L'exécution d'une telle requête nécessite l'appel des bases de données distantes.
- Requête mixte : une requête mixte est une requête qui affecte à la fois des données localement disponibles et des données distantes. L'exécution d'une telle requête nécessite donc l'appel des bases de données cachées et/ou de bases de données distantes.

Selon les explications des données précédemment, nous avons :

- Requête multirelationnelle locale
- Requête multirelationnelle distante
- Requête multirelationnelle mixte

4.5 Stratégie d'exécution des requêtes

Dans les stratégies de traitement des requêtes, l'existence d'une requête mono-relationnelle ou multi-relationnelle n'a pas d'importance parce que le schéma d'exécution ne changera pas.

4.5.1 Exécution locale : Stratégie d'exécution vers la BC

```
SELECT Temp, ville
FROM ruche
WHERE date = CURDATE AND Ville = Montreal
```

Si les éléments de réponse de ce type de requête sont disponibles dans la base de données cachée, alors nous aurons le schéma d'exécution de la figure 4.4.

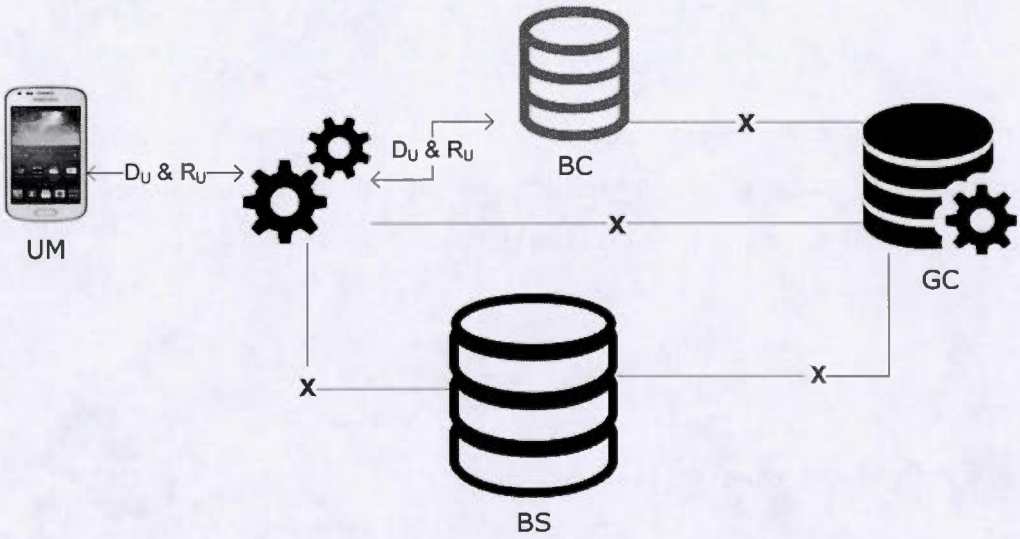

Figure 4.4 Requête affectant seulement la BC

La demande de l'utilisateur (DU) est envoyée d'abord vers la base de données cachée (BC). Celle-ci a la possibilité de répondre complètement à la demande alors une réponse est acheminée à l'utilisateur (RU). Selon les mêmes principes dans la méthode spirale (SPI) de Komai et al., 2014, le nœud qui recueille un résultat satisfaisant transmet le résultat au nœud de délivrance de la requête. Il n'y a pas de propagation de la requête quand la demande est entièrement satisfaite. Il n'y a pas de communication avec les autres nœuds GC et BS.

Montréal -12° C

4.5.2 Exécution distante : Stratégie d'exécution vers le gestionnaire de cache uniquement

```
SELECT Temp, ville
FROM ruche
WHERE date = CURDATE AND Ville = Vancouver
```

Dans cet exemple, nous supposons que la base de données cachée (BC) est vide, mais le gestionnaire de cache contient les données suffisantes pour répondre à la requête. Alors on aura le diagramme d'exécution suivant :

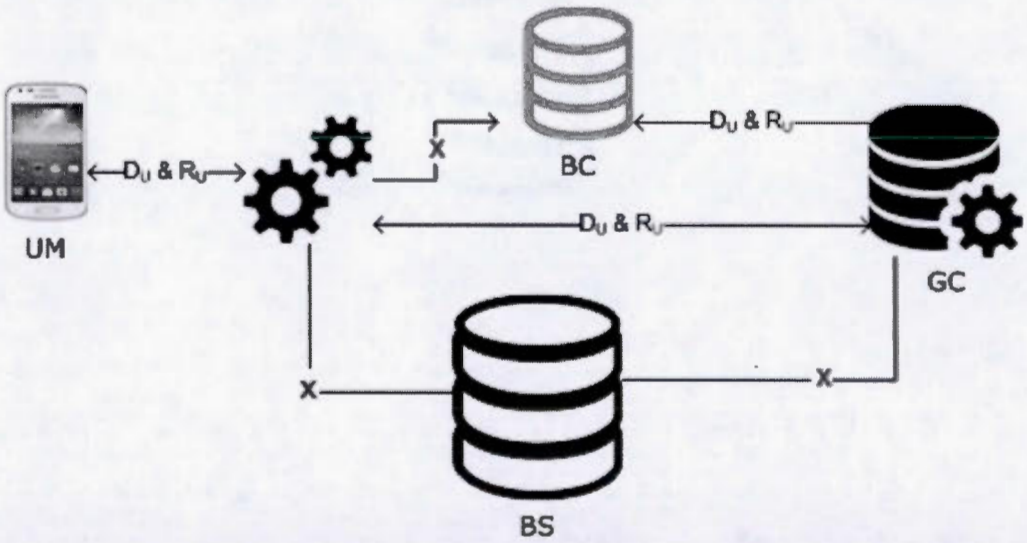

Figure 4.5 Requête affectant seulement le GC

Une demande DU est créée pour l'utilisateur mobile. La demande est envoyée vers la base de données cachée pour trouver une réponse. La base de données cachée étant vide, elle ne peut pas répondre à la demande. L'absence de réponse de BC suscite la redirection de la requête auprès du gestionnaire de cache qui à son tour va répondre à la requête. Comme la base de données du gestionnaire de cache parvient à répondre complètement la requête, alors la réponse est envoyée à l'utilisateur, il n'y a pas d'autre propagation. La base de données serveur n'est pas sollicitée.

Vancouver -5 °C

4.5.3 Exécution locale et distante : Stratégie d'exécution quand la réponse est à la fois dans BC et le GC

```
SELECT Temp, vile
FROM ruche
WHERE ville = Montreal AND Vancouver;
```

Supposons que dans cet exemple la base de données cachée (BC) contient une partie de la requête et l'autre partie de la requête se trouve dans les bases de données distantes plus précisément dans le gestionnaire de caches. Alors nous aurons le processus d'exécution suivant :

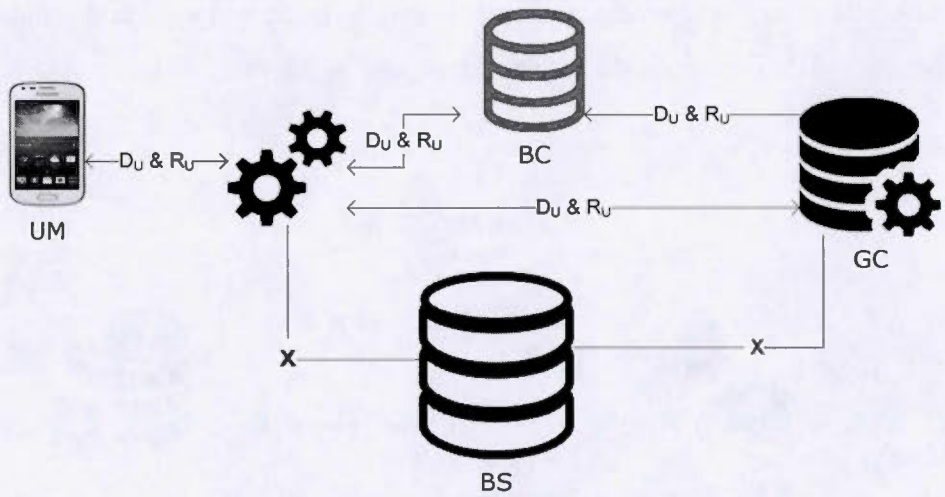

Figure 4.6 Requête affectant la BC et le GC

Une demande DU est créée pour l'utilisateur mobile. La demande est envoyée vers la base de données cachée (BC) pour trouver une réponse. La base de données cachée ne parvient pas à satisfaire la requête au complet, mais une partie de la demande a été satisfaite. La partie non satisfaite de la requête va être redirigée auprès du gestionnaire de cache. Celui-ci va tenter de répondre à la requête. Si la base de données du gestionnaire de cache parvient à répondre complètement à cette sous-

requête, alors la réponse est envoyée à l'utilisateur. Il n'y a pas d'autre propagation vers la base de données serveur (BS), car la requête est complètement satisfaite et la base de données serveur n'est pas sollicitée.

Montréal -12 °C
Vancouver -5 °C

4.5.4 Exécution locale et distante mixte : Stratégie d'exécution quand la réponse est à la fois dans la BC, le GC et le BS

Dans cet exemple, nous supposons que les informations que contiennent les bases de données cachées (BC) et le gestionnaire de cache (GC) ne peuvent pas satisfaire complètement la requête. Par contre, une partie de la requête doit être exécutée auprès de la base de données serveur qui est l'unique source de donnée contenant l'information. Alors, on aura le diagramme d'exécution suivant :

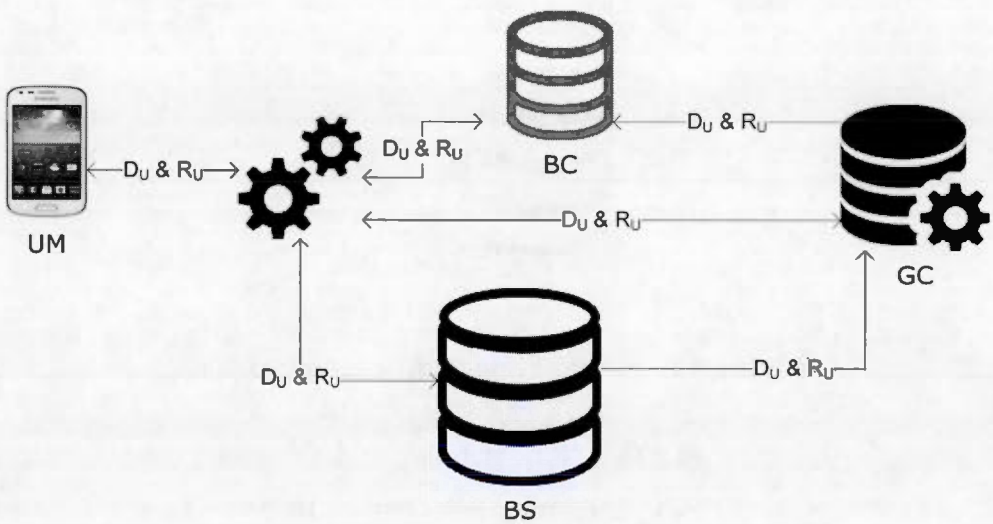

Figure 4.7 Requête affectant la BC, le GC et la BS

Une demande (DU) est créée pour l'utilisateur, la demande est envoyée vers de la base de données cachée (BC) pour trouver une réponse. La base de données cachée ne peut pas répondre à la demande complètement, mais une partie de la demande a été traitée (RU). La partie non satisfaite de la demande va être redirigée auprès du gestionnaire de cache qui à son tour va tenter de répondre à la requête. Si la base de données du gestionnaire de cache ne peut pas répondre elle aussi qu'à une partie du reste de la requête, alors une partie de la réponse est envoyée à l'utilisateur (RU). Le reste de la requête se dirige vers la base de données serveur qui va traiter à son tour le reste de la demande et envoie la réponse à l'utilisateur.

CHAPITRE V

ÉVALUATION DES RÉSULTATS

Nous avons proposé une stratégie de recherche qui sera capable de réduire le trafic sur l'infrastructure de communication et de résorber les goulots d'étranglement. Cette approche améliore la performance des requêtes des utilisateurs sur le réseau mobile de type LTE. Son rôle est de répondre efficacement aux requêtes des utilisateurs à l'aide des contenus déjà calculés ou précompilés. Cela évite de refaire le travail à chaque itération. Dans ce chapitre, nous allons montrer la justification de notre approche. Les objectifs de ce chapitre sont : (i) évaluer les performances de notre proposition, et (ii) étudier les impacts

L'étude des performances des systèmes de communications au niveau des réseaux mobiles (LTE) représente une tâche complexe où des outils de simulation spécialisés doivent être utilisés. La plupart des fournisseurs d'équipements de communication mobiles ont mis en œuvre leurs propres simulateurs. Par ailleurs, d'autres simulateurs développés peuvent être achetés au moyen d'une licence commerciale, mais leurs codes sources ne sont pas accessibles au public. Les simulateurs réseau offrent beaucoup d'économie, de temps et d'argent pour l'accomplissement des tâches de simulation et ils sont également utilisés pour que les concepteurs des réseaux puissent tester les nouveaux protocoles ou modifier les protocoles existants d'une manière contrôlée et productrice.

Lors d'une simulation d'un protocole ou d'un algorithme plusieurs phases sont indispensables comme : (i) la définition du problème (ii) la construction du modèle (iii) la simulation et (iv) l'analyse des résultats en vue de prendre une décision. Dans le cadre de notre projet, le simulateur open source le mieux adapté et capable de traiter le comportement des équipements dans l'environnement mobile est le ns-3. Vu les limites et le manque de modules dédiés au traitement des requêtes dans l'environnement mobile, nous nous contentons de présenter des études de cas.

5.1 Facteur de performance

Dans le but d'effectuer une étude de performance de notre stratégie de mise en cache et de traitement des requêtes, certains facteurs peuvent affecter les résultats. Dans les réseaux à faibles débits et fortes latences comme les réseaux sans fil, le transfert des résultats peut être important. Ainsi, nous calculons une estimation du temps de traitement des requêtes (T_{Tr}), une estimation du temps de transfert des résultats (T_{Trans}) et une estimation du coût de traitement de la requête (C_{Tr}).

- Temps de transfert

C'est le temps écoulé entre l'envoi de la demande de l'utilisateur ((T_S) vers la base de données et le temps de réception des données de réponse après traitement (T_R). Le total du temps d'envoi et de réception noté par $T_{Trans} = T_S + T_R$.

Le calcul du temps de transfert peut varier en fonction de la disponibilité des données. Si les données sont disponibles sur plusieurs bases de données, le temps de transfert sera la somme des n temps d'envoi vers n sources de données et de n temps de réponse. De ce fait, la formule pour calculer le temps de transfert sera : $T_{Trans} = \{(T_{S1} + T_{S2} .... + T_{Sn}) + (T_{R1} + T_{R2} ....+T_{Rn})\}$. La figure 5.1 présente une perspective de temps de transfert de n sources de données.

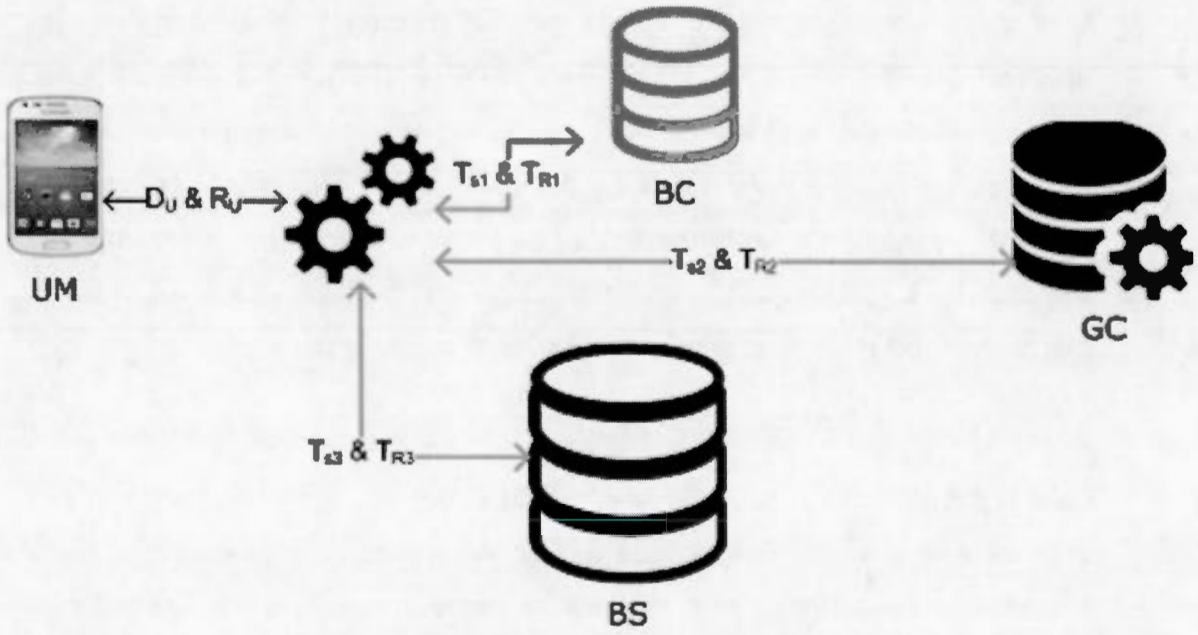

Figure 5.1 Transfert des données vers les sources de données

Le temps d'envoi (T_{S1}) et le temps de réponse (T_{R1}) sont le temps que prend le processus pour transférer les données entre l'UM et la base de données cachée. Le temps d'envoi et de réception vers le gestionnaire de cache est T_{S2} & T_{R2}. Le T_{S3} & T_{R3} est le temps pour l'envoi et la réception des données vers la base de donnée serveur.

- Disponibilité des données

Selon le plan d'exécution des requêtes, chaque nœud a la possibilité de répondre à sa propre requête ou à une partie de la requête. Quand la réponse est déduite directement du nœud émetteur de la requête le temps de chargement est beaucoup plus rapide que si la réponse venait du GC et encore plus rapide que si la réponse devait venir du serveur de donnée distante BS..

Le temps de traitement croît de manière proportionnelle avec la durée de transfert des données. Le traitement des données distant génère un coût de traitement plus grand. Selon l'emplacement de l'élément de réponse, nous déterminons les coûts de traitement.

- Données disponibles dans le cache.

Dans ce cas, nous supposons que les données sont disponibles localement (BC), quand le cache peut répondre la requête de l'utilisateur la vitesse de transfert est négligeable ou elle est proportionnelle à la vitesse de traitement de l'appareil mobile. Dans ce cas, il n'y a pas de transfert externe, le débit du réseau n'est pas considéré et il n'y a pas de consommation des données mobile. La figure 5.2 illustre une situation ou les données sont stockés dans le cache et que le temps te transfert de données est négligeable.

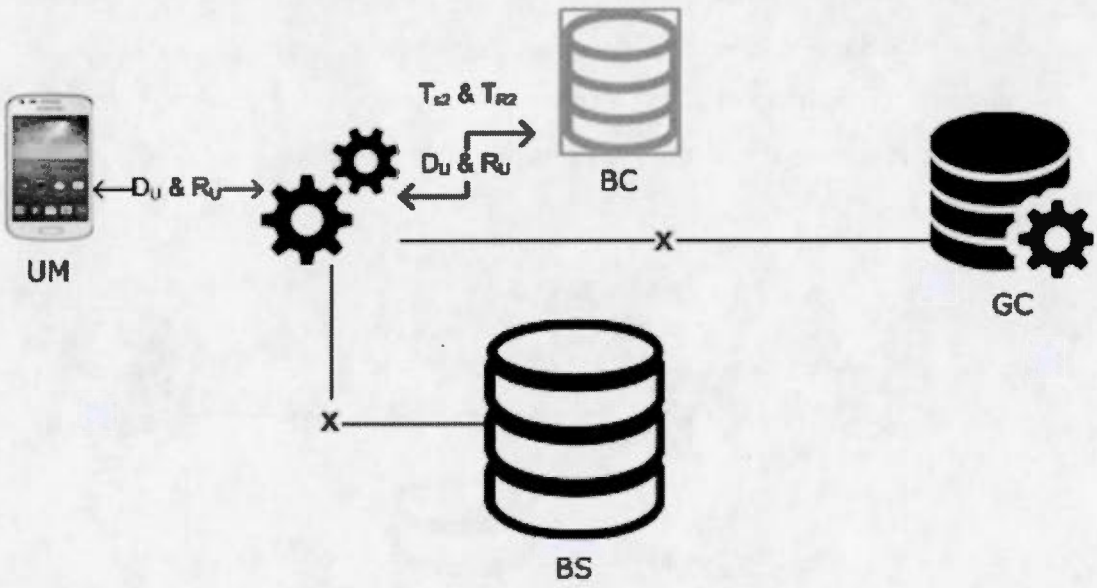

Figure 5.2 Traitement d'une requête dans la donnée cachée.

La figure 5.2 est une représentation d'un traitement de requête avec des données internes. La requête est exécutée directement sur la base de données cachée. Les autres sources de données ne sont pas sollicitées. Ce qui permet un coût de traitement très avantageux par rapport au modèle de traitement des données mobiles actuelles. Dans cet exemple, la performance du coût de traitement est estimée à 95 %, le temps de lecture du cache, le temps d'envoi et le temps de réception sont estimés à 5 %.

- Données disponibles dans le cache et le GC

Dans cette situation, nous supposons que les données susceptibles de répondre à la requête se trouvent entre la BC et le GC. Dans une telle situation, la requête doit récupérer les informations des 2 sources de données afin de répondre pleinement à la demande. Le débit maximal de transfert de données dans le réseau LTE est 100 Mb/s. En utilisation réelle, la vitesse d'envoi peut atteindre 15Mb/s et la vitesse de réception 30Mb/s. (Guangxiang et Jucai, 2010) alors quand les données sont disponibles dans le eNodeB, le coût de traitement des données va augmenter avec l'ajout du temps d'envoi et de réception de la demande de l'utilisateur vers l'eNodeB (GC). Le processus d'exécution est illustré par la figure 5.3.

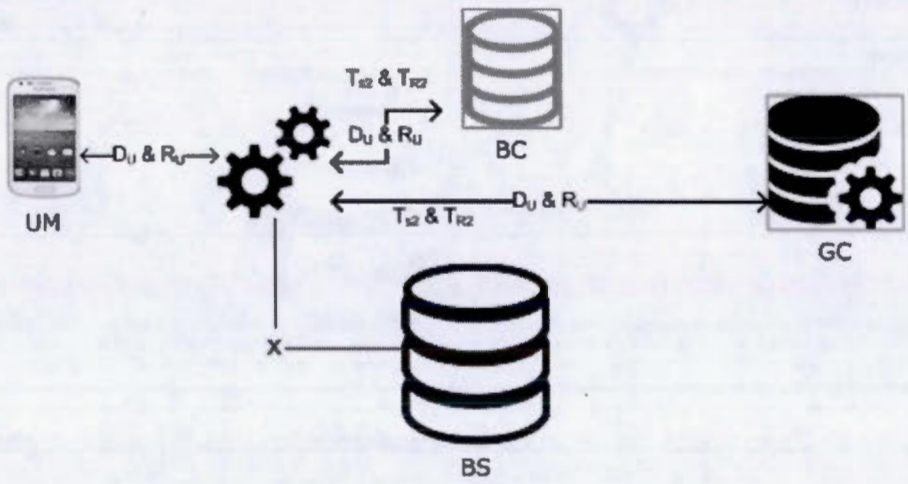

Figure 5.3 Traitement d'une requête entre 2 sources de données.

Dans cet exemple, nous pouvons constater qu'il y a un temps de transfert pour chaque source de données. Il y a aussi un temps de traitement avec une vitesse de transfert de 15 Mb/s en envoi et 30 Mb/s en réception. Cela va augmenter le coût de traitement de la requête, mais encore une fois, le coût de la requête ne sera pas trop élevé, car les données cachées dans le GC sont considérées comme un moyen plus rapide d'accéder aux données non disponibles sur l'utilisateur mobile.

- Données disponibles à la fois sur la BC, GC et le BS.

Supposons que les données sont distribuées dans plusieurs sources de données. Ça augmente le coût de traitement de la requête en ajoutant un temps de transfert pour chaque source de données et les coûts des traitements distants. Dans une telle situation, les n sources de données reçoivent leurs sous-requêtes, effectuent leur traitement et acheminent les résultats de leur traitement vers l'utilisateur mobile.
Pour chaque source de données, nous aurons un temps d'envoi (T_S) et un temps de réponse (T_R). La figure 5.4 illustre une stratégie d'exécution de requête avec 3 sources de données (BC, GC et BS).

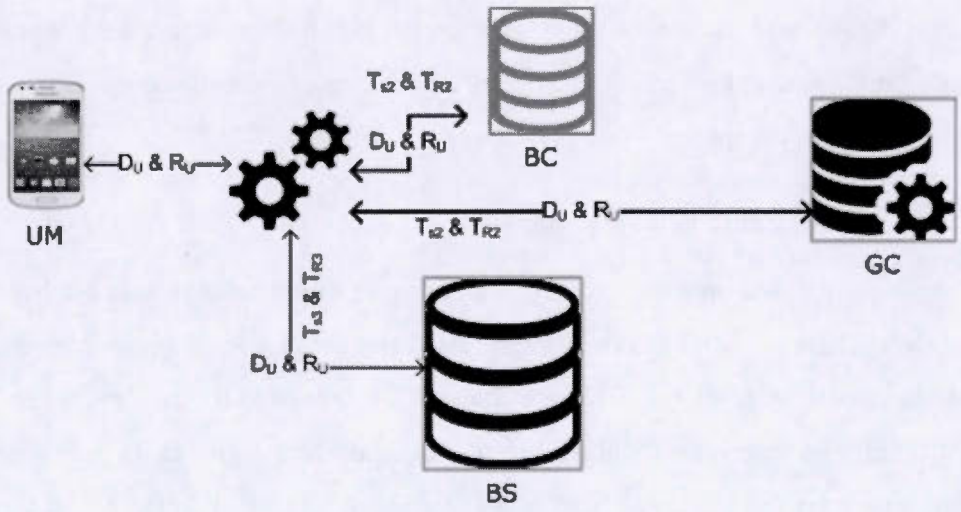

Figure 5.4 Traitement d'une requête entre 3 sources de données

- Utilisateur mobile

Dans l'analyse de traitement des requêtes dans l'environnement mobile, plusieurs raisons peuvent atténuer le temps de traitement d'une requête auprès d'un utilisateur mobile. Parmi ses différentes raisons, on a :

- La capacité énergétique.

Les gadgets électroniques ont tendance à conserver l'énergie pour prolonger l'autonomie. Quand un équipement mobile atteint un niveau de piles inférieur ou égal à 20 %, la puissance de calcul et de connectivité de cet appareil n'est pas le même. Dans le but de rester plus longtemps fonctionnel, l'appareil réduit certaines fonctionnalités qui affectent sa puissance de fonctionnement. Selon Apple, le Power mode de l'iPhone réduit la vitesse du CPU, du GPU performance de la connectivité, etc. (Apple, 2015).

- La couverture réseau.

Les compagnies de télécommunication cellulaire ont tendance à privilégier certaines zones urbaines. Une zone contenant une population dense a une forte chance d'avoir une bonne couverture réseau. Par contre dans certaines zones, elles font des ententes avec d'autres compagnies pour la couverture.

- La puissance de l'appareil mobile

La puissance de l'appareil croît de manière proportionnelle avec la qualité de service et de traitement. Nous avons effectué des tests de vitesse sur plusieurs équipements mobiles sur le réseau LTE nous avons constaté des différences par rapport aux différents équipements. Malgré que les équipements sont sur le même réseau, des baisses de performance sont enregistrées d'un appareil à un autre. Le tableau 5.1 et le graphe de la figure 5.5 donnent une vue sur les résultats obtenus.

Appareils	PING	Téléchargement	Téléversement
iPhone 6	19	101,4	26,9
iPhone 5S	36	47	10,7
Samsung G Note 5	68	55	10
Samsung G S4	43	21,2	9,2

Tableau 5.1 Résultat des différents tests de latence.

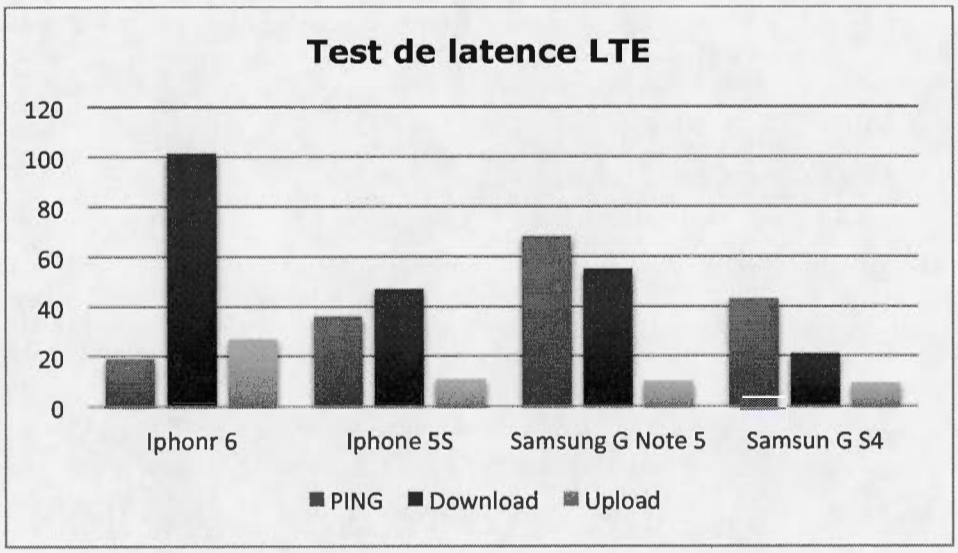

Figure 5.5 Graphe des tests de latence

Dans ce graphe, on peut constater la variation entre les différents équipements mobiles. Par exemple, on peut voir que le iPhone 6 à une vitesse de téléchargement 2 fois plus que l'iPhone 5s qui est sa version précédente. Nous avons aussi constaté la même chose entre le Samsung Galaxy Note 5 et le Samsung Galaxy S4.

Dans le but d'avoir une étude détaillée sur les tests de vitesse, nous avons utilisé l'application « Speedtest » pour effectuer des tests de latence sur le réseau Vidéotron en utilisant un iPhone 5s. Ces tests sont faits à des intervalles proches, mais à la même position. Nous avons constaté une variation sur les différents résultats de la simulation. À chaque simulation, nous constatons un écart sur les débits en amont et en aval et le temps de traitement. La figure 5.6 nous donne une idée sur les différentes variations.

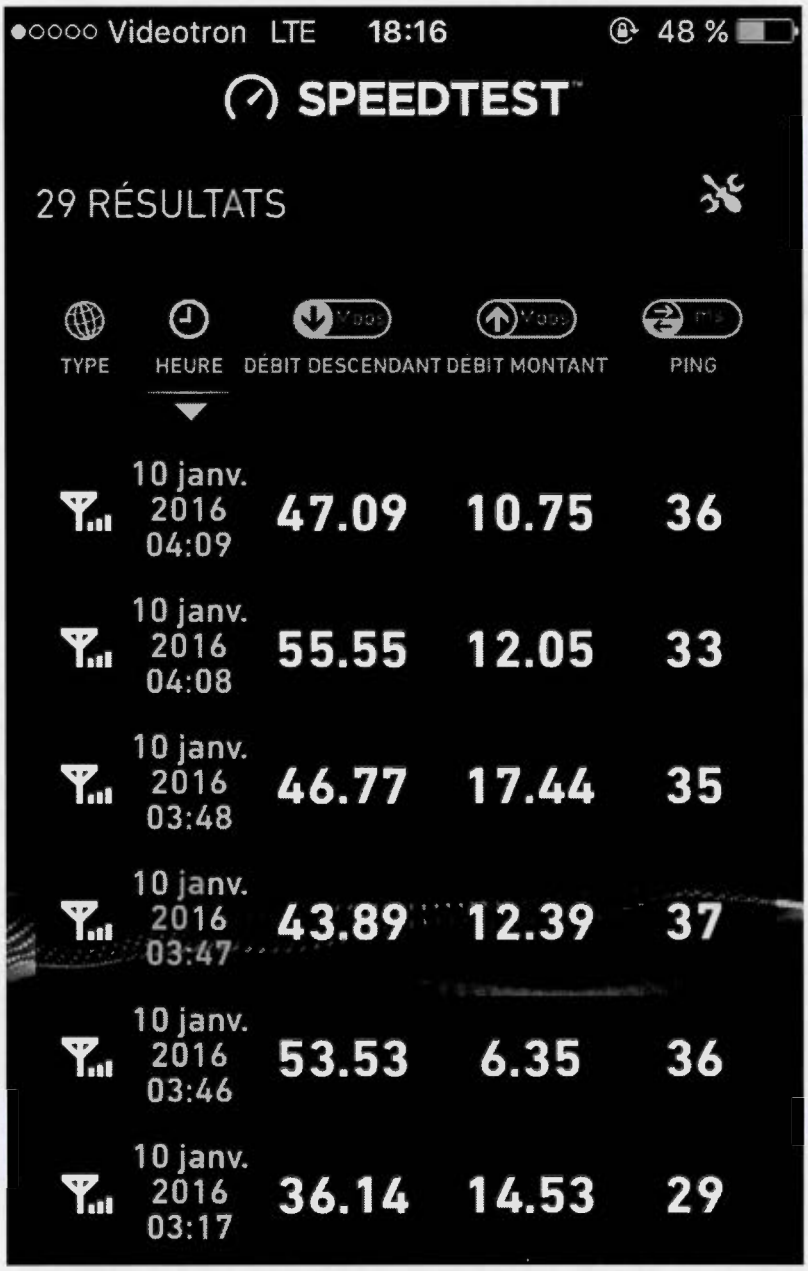

Figure 5.6 Résultat des tests de latence iPhone 5s

5.2 Les avantages de notre approche

Avec notre méthode, nous pouvons constater les résultats suivants selon divers critères :

1. La disponibilité des données même en mode déconnecté l'utilisateur mobile a la possibilité de répondre à certains nombre de requêtes
2. Le temps de traitement, avec la disponibilité des données, les requêtes sont traités en majeure partie dans le cache local ou distant.
3. La gestion économique des données mobile : avec la résolution des requêtes dans le cache ça évite la consommation des données mobile.
4. Amélioration de l'autonomie d'énergie : la consommation d'énergie croît de manière proportionnelle avec la durée de traitement des requêtes et l'utilisation de la bande passante du réseau. Avec le traitement des requêtes dans le cache, nous aurons une plus grande autonomie d'énergie.

CONCLUSION GÉNÉRALE

Dans ce mémoire, nous nous sommes intéressés à la mise en cache et aux traitements des requêtes dans l'environnement mobile afin d'optimiser le traitement des requêtes dans une base de données mobile et distribuée.

Nous avons montré que l'utilisation d'un cache sémantique intelligent et réparti permet d'améliorer le débit transactionnel d'une base de données mobile dans l'environnement réseau LTE. Avant d'introduire notre modèle, nous avons présenté une revue des principales approches existantes en mettant l'accent sur les modèles qui reflètent le mieux notre approche. Nous avons apporté des arguments pour la nécessité d'un modèle simple. Notre modèle permettrait de faire la mise en cache des requêtes et leurs résultats. Il permet d'utiliser en retour ces résultats pour répondre aux futures requêtes dans l'environnement mobile.

Dans le Chapitre 3, nous présentons la syntaxe et la sémantique opérationnelle de notre architecture de mise en cache. Ensuite, nous avons montré l'utilisation des expressions de notre modèle à travers plusieurs exemples. Ces derniers montrent que nos algorithmes permettent une description du comportement de chaque requête dans l'environnement des bases de données mobile.

Dans le Chapitre 4, nous proposons un mécanisme de routage des requêtes pour garantir une exécution rapide et cohérente des requêtes sur les différentes sources de données. Nous introduisons plusieurs relations qui permettent d'identifier la source contenant la réponse de la requête tout en favorisant en tout premier lieu une exécution auprès de la base de données cachée (BC). Par des techniques classiques, nous montrons l'exécution des requêtes sur les différentes sources.

Dans le chapitre 5, nous présentons des résultats de notre approche. Nous avons énuméré et évalué les facteurs de performance. Nous avons montré qu'avec la disponibilité des données dans le cache, le temps de traitement des requêtes sera réduit de manière sensible nous avons effectué des calculs sur des résultats en fonction des normes de l'environnement du réseau LTE.

RÉFÉRENCES

Ahmad, M., Qadir, M. A., et Sanaullah, M. (Decembre, 2008). Query processing over relational databases with semantic cache: A survey. In *Multitopic Conference, 2008. INMIC 2008. IEEE International* (pp. 558-564). IEEE.

Ahmad, M., Qadir, M. A., Sanaullah, M., et Bashir, M. F. (février 2009). An efficient query matching algorithm for relational data semantic cache. In *Computer, Control and Communication, 2009. IC4 2009. 2d International Conference on* (pp. 1-6). IEEE.

Ali, T., Qadir, M. A., et Ahmad, M. (Juin, 2010). Translation of relational queries into Description Logic for semantic cache query processing. In Information and Emerging Technologies (ICIET), 2010 International Conference on (pp. 1-6). IEEE.

Alonso, R., et Ganguly, S. (Septembre, 1993). Query optimisation for energy efficiency in mobile environments. In *Proceedings of the 1993 International Workshop on Foundations of Models and Languages for Data and Objects, Aigen, Austria.*

Amja, A. M., Obaid, A., et Seguin, N. (Decembre, 2011). A distributed mobile database architecture. In *2011 IEEE Asia-Pacific Services Computing Conference* (pp. 62-69). IEEE.

Barbará, D., et Imieliński, T. (Mai, 1994). Sleepers and workaholics: caching strategies in mobile environments. In *ACM Sigmod Record* (Vol. 23, No. 2, pp. 1-12). ACM.

Barbará, D. (1999). Mobile computing and databases a survey. *Knowledge and Data Engineering, IEEE Transactions on, 11*(1), 108-117.

Bashir, M. F., et Qadir, M. A. (2006, December). Hisis: 4-level hierarchical semantic indexing for efficient content matchingover semantic cache. In 2006 IEEE International Multitopic Conference (pp. 211-214). IEEE.

Buneman, O. P., Davidson, S. B., et Watters, A. (1991). A semantics for complex objects and approximate answers. Journal of Computer and System Sciences, 43(1), 170-218.

Dar, S., Franklin, M. J., Jonsson, B. T., Srivastava, D., et Tan, M. (Septembre, 1996). Semantic data caching and replacement. In *VLDB* (Vol. 96, pp. 330-341).

E. Pitoura et B. Bhargava, "Maintaining consistency of data in mobile distributed environments," Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on, Vancouver, BC, 1995, pp. 404-413.

Ebling, M. R. (1997). *Evaluating and improving the effectiveness of caching for availability* (Doctoral dissertation, Ph. D. Thesis, Department of Computer Science, Carnegie Mellon University).

Elfaki, M. A., Ibrahim, H., Mamat, A., Othman, M., et Safa, H. (2014). Collaborative caching priority for processing requests in MANETs. Journal of Network and Computer Applications, 40, 85-96.

Forman, G. H., et Zahorjan, J. (1994). The challenges of mobile computing. *Computer*, *27*(4), 38-47.

Godfrey, P., et Gryz, J. (Aout, 1997). Semantic Query Caching for Hetereogeneous Databases. In *KRDB* (pp. 6-1).

Guangxiang, X., et Jucai, C. (2010). Study on overcutting-bolting & grouting-backfilling concrete to control the floor heave of deep mine roadway. Journal of China Coal Society, 35(8), 1242-1246.

Imielinski, T., et Badrinath, B. R. (Aout, 1992). Querying in highly mobile distributed environments. In *VLDB* (Vol. 92, pp. 41-52).

Imielinski, T., et Badrinath, B. R. (1994). Mobile wireless computing : challenges in data management. *Communications of the ACM*, *37*(10), 18-28.

Kang, S. W., Kim, J., Im, S., Jung, H., et Hwang, C. S. (Juin, 2006). Cache Strategies for Semantic Prefetching Data. In *Web-Age Information Management Workshops, 2006. WAIM'06. Seventh International Conference on* (pp. 7-7). IEEE.

Karnstedt, M., Sattler, K. U., Geist, I., et Höpfner, H. (Octobre, 2003). Semantic Caching in Ontology-based Mediator Systems. In *Berliner XML Tage* (pp. 155-169).

Kistler, J. J., et Satyanarayanan, M. (1992). Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems (TOCS)*, *10*(1), 3-25.

Komai, Y., Sasaki, Y., Hara, T., et Nishio, S. (2014). NN Query Processing Methods in Mobile Ad Hoc Networks. Mobile Computing, IEEE Transactions on, 13(5), 1090-1103.

Kumar, TV Vijay, Haider, Mohammad, et Kumar, Santosh. Proposing candidate views for materialization. In : *Information Systems, Technology and Management.* Springer Berlin Heidelberg, 2010. p. 89-98.

Le Mouël, F. (2003). *Environnement adaptatif d'exécution distribuée d'applications dans un contexte mobile* (Doctoral dissertation, Université Rennes 1).

Liang, R. B., et Qiong, L. I. U. (2012). Answering queries using cooperative semantic cache in mobile computing environments. *The Journal of China Universities of Posts and Telecommunications*, *19*(3), 54-59.

Lübbe, C., Brodt, A., Cipriani, N., Großmann, M., et Mitschang, B. (Juin, 2011). DiSCO : A distributed semantic cache overlay for location-based services. In Mobile Data Management (MDM), 2011 12th IEEE International Conference on (Vol. 1, pp. 17-26). IEEE.

M. Ahmad, M. A. Qadir et M. Sanaullah, « Query processing over relational databases with semantic cache: A survey," Multitopic Conference, 2008. INMIC 2008. IEEE International, Karachi, 2008, pp. 558-564.

Madria, S. K., et Bhargava, B. (Juillet, 1998). A transaction model for mobile computing. In *Database Engineering and Applications Symposium, 1998. Proceedings. IDEAS'98. International* (pp. 92-102). IEEE.

Madria, S. K., Mohania, M. K., et Roddick, J. F. (1998). A query processing model for mobile computing using concept hierarchies and summary databases. *Foundations of Database Organisation*, 146-157.

Madria, S. K., Mohania, M., Bhowmick, S. S., et Bhargava, B. (2002). Mobile data and transaction management. *Information Sciences*, *141*(3), 279-309.

N. Qing-jun, « Answering semantic caches in integration systems, » 2010 2d International Conference on Signal Processing Systems, Dalian, 2010, pp. V3-461-V3-463.

Notouom, H. (1996). Réalisation de la mémoire partagée dans les systèmes répartis. Diplôme de maîtrise en sciences appliquées (h4. Sc. A).

Olston, C., et Widom, J. (Juin, 2002). Best-effort cache synchronization with source cooperation. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data (pp. 73-84). ACM.

Poirier, Sylvain (2013). « Extensions parallèles pour le langage Nit » Mémoire. Montréal (Québec, Canada), UQÀM, Maîtrise en informatique.

Poularakis, K., Iosifidis, G., Sourlas, V., et Tassiulas, L. (Avril, 2014). Multicast-aware caching for small cell networks. In Wireless Communications and Networking Conference (WCNC), 2014 IEEE (pp. 2300-2305). IEEE.

Ren, Q., et Dunham, M. H. (Aout, 2000). Using semantic caching to manage location dependent data in mobile computing. In *Proceedings of the 6th annual international conference on Mobile computing and networking* (pp. 210-221). ACM.

Ren, Q., Dunham, M. H., et Kumar, V. (2003). Semantic caching and query processing. *Knowledge and Data Engineering, IEEE Transactions on, 15*(1), 192-210.

Safaeei, A. A., Haghjoo, M., et Abdi, S. (Novembre, 2008). Semantic cache schema for query processing in mobile databases. In *Digital Information Management, 2008. ICDIM 2008. Third International Conference on* (pp. 644-649). IEEE.

Stefano Spaccapietra. Base de données réparties. Ecole Polytechnique Fédérale de Lausanne, Mars 1998

Swaroop, V., Shanker, U. (Septembre, 2010). Mobile distributed real time database systems: A research challenges. In *Computer and Communication Technology (ICCCT), 2010 International Conference on* (pp. 421-424). IEEE.

Yin, L., Cao, G., et Cai, Y. (Janvier, 2003). A generalized target-driven cache replacement policy for mobile environments. In Applications and the Internet, 2003. Proceedings. 2003 Symposium on (pp. 14-21). IEEE.

Zhang, X., et Wang, W. (2010). Carrier aggregation for LTE-advanced mobile communication systems. IEEE Communications Magazine, 89.

AT&T (2014, Juin). Throttling Their Smartphone Customers to Deal with Spectrum Shortage. Récupéré à http://u2l.info/1nkLbB

Cisco Visual Networking Index (2014) Global Mobile Data Traffic Forecast Update, 2014–2019. Récupéré à http://u2l.info/1NoUTo.

Apple (2015, decembre) Mode faible consommation sur les iPhones. Récupéré à https://developer.apple.com/library/ios/documentation/Performance/Conceptual/EnergyGuide-iOS/LowPowerMode.html.

Rcrwireless (2014, 9 mai) les composants du réseau LTE. Récupéré à http://www.rcrwireless.com/20140509/wireless/lte-mme-epc