Table des Matières

Introduction

Dans le domaine de l'informatique, une application est définie comme étant composée ou constituée d'un ensemble de services, où chaque service est constitué d'un ensemble de primitives et assu re une fonctionnalité particulière de l'application. Le web, littéralement toile désigne le réseau maillé formé par l'Internet et le contenu multimédia qu'il comporte.

Le Web 2.0 se démarque concrètement du web classique par de nouvelles approches : un rapport modifié à l'écriture, à la lecture, au partage, au signalement, à la description de ressources (documentaires ou non). La chose la mieux partagée est la mise en commun d'informations où l'internaute n'est plus un consommateur mais un producteur d'information. Les applications web 2.0 comme les applications communautaires (blog, Partage de photos, mondes virtuels….) voient leur nombre d'utilisateurs et leur charge de traitement croître de manière exponentielle. Par conséquent le contrôle centralisé des accès aux données est exposé à des goulots d'étranglement.

Pour faire face à cette croissance, de nouvelles architectures sont conçues pour bénéficier des ressources disponibles à large échelle sur Internet. Les architectures client/serveur initialement déployées en cluster sont remplacées par les solutions pair-à-pair.

Les architectures pair-à-pair permettent de concevoir des solutions efficaces de gestion des données partagées à très grande échelle. Les données sont organisées dans un espace logique dans lequel l'accès direct à un point précis de cet espace est possible.

Ce mémoire s'intéresse au contrôle décentralisé des accès aux données.

L'objectif visé est de concevoir une solution efficace pour garantir le traitement cohérent des transactions lorsque les applications et les données sont réparties à grande échelle.

Le mémoire s'articule autour de quatre chapitres. Le premier chapitre présente une étude des applications web 2.0, plus particulièrement de Second Life, un jeu de mondes virtuels. Il présente les opérations de manipulation et de localisation des données et une caractérisation de ces transactions cela grâce à une métrique quantifiant la localité et la fréquence des accès. Le deuxième chapitre porte sur les applications pair-à-pair basées sur les diagrammes de Voronoi. Le troisième chapitre présente l'algorithme que nous avons proposé pour délimiter dynamiquement les données et créer des zones de concurrence en fonction des transactions lancées par les utilisateurs. Enfin le quatrième chapitre présente l'implémentation et les résultats que nous avons obtenus lors de la validation de notre algorithme.

Chapitre1

Etude des applications web 2.0 : les Transactions web 2.0

1. Introduction

Parmi les applications web 2.0, nous avons les blogs, le partage de photos sur internet comme Picasa, les jeux virtuels 3D comme Second life, triple life etc. Toutes ces applications offrent des services à travers lesquels on peut localiser et manipuler des données. Plus précisément ces applications permettent de manipuler des objets web (avatar, blog, photos,) en utilisant les attributs de ces objets. Cependant, pour étudier les applications web 2.0 nous nous limiterons aux jeux virtuels et principalement à Second Life .Ce choix est basé sur le fait que nous avons considéré que les applications de jeux virtuel peuvent incorporer les blogs, le partage de photos et beaucoup d'autres applications du web 2.0. Donc un travail réalisé sur Second Life peut être généralisé à toutes ces applications.

2. Architecture et Fonctionnement de Second Life

2.1. Description de l'architecture de Second Life

L'architecture de Second Life est une architecture client/serveur [13] représentée sur la figure ci-dessous :

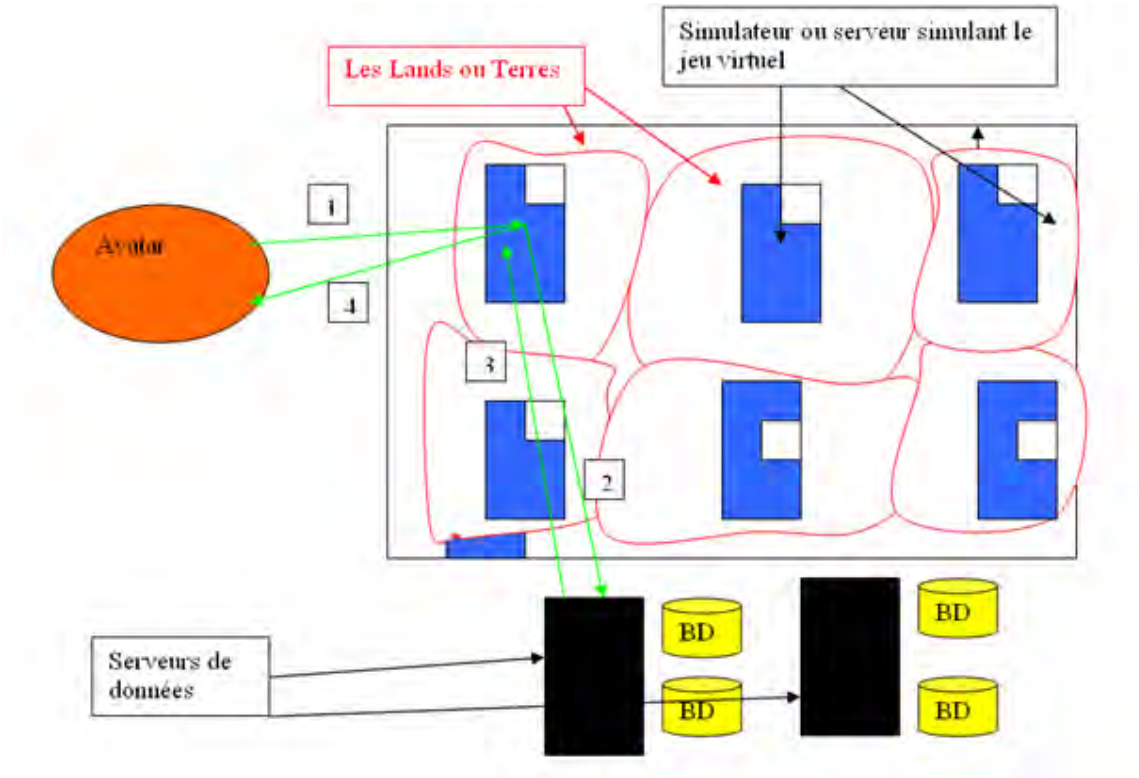


Figure 1 : Architecture de Second Life

2.2. Les différents composants de l'architecture et leur fonctionnement

2.2.1. Les différentes composantes de l'architecture

L'application de monde virtuel Second Life est composée d'une banque de serveurs dont le principal est le simulateur. Ce simulateur est accompagné d'autres serveurs tels que le serveur de données, le serveur de login, le serveur d'espace. En plus de ces serveurs nous avons des clusters de bases de données que sont les clusters de bases de données centralisées, les clusters de bases de données des inventaires ….

Un avatar représente chaque utilisateur et lui permet lors de sa session de visiter le monde virtuel. Le monde virtuel est constitué d'un ensemble d'espaces virtuels. Dans Second Life, chaque utilisateur connecté possède un espace virtuel où il peut créer ses objets et organiser des activités. Cet espace virtuel est appelé Land et peut être partagé par plusieurs joueurs ou avatars. Chaque land appartient à un seul utilisateur qui peut être connecté ou non et qui décide des droits d'accès à son land. C'est dire qui accorde des privilèges d'accès ou pas aux utilisateurs.

2.2.1. Les fonctions de chaque composante de l'architecture

Nous allons donner les fonctions des différents composants de l'architecture de Second Life :

- ✓ Les fonctions du simulateur.

Le simulateur constitue le serveur principal du jeu virtuel. Il gère le stockage de l'état des objets et des régions (ensembles de lands) en interagissant avec les bases de données .Ensuite il transmet les données aux lands et les objets aux clients sur ces lands. Il transmet les données images sur une file de priorité. Les chats et message instantanés sont aussi exécutés par les simulateurs ainsi que la simulation physique.

Les simulateurs ont en charge :

- l'exécution de l'application sur les machines
- la détection des collisions
- la trace de chaque élément du monde virtuel
- l'envoi du lieu des objets aux utilisateurs (avatars)
- l'envoi des mis à jours aux utilisateurs quand ils en ont besoin

Le simulateur peut simuler au plus un taux de 45 images/sec. Les simulateurs communiquent à travers le protocole UDP.

- ✓ Les fonctions des autres serveurs

Le monde virtuel est constitué d'une banque d'autres serveurs, chaque serveur a la charge de gérer les objets, les terrains, les avatars et de s'assurer que les clients connectés au serveur sont mis à jour correctement. Parmi les serveurs nous avons

- Le serveur de login

Il procède à la vérification du login et du mot de passe, détermine la région de connexion par rapport à la région lors de la dernière connexion. Il identifie le simulateur en charge de cette région et vérifie si l'utilisateur est autorisé à se connecter là bas .Dans le cas contraire, il demande au simulateur de cette région de faire patienter la connexion en attendant qu'il trouve la région où l'utilisateur peut se connecter.

- Le serveur d'espace

Il gère le routage des messages en se basant sur une grille de localisation en x, y. Les simulateurs l'utilisent pour s'enregistrer, mais aussi pour trouver leurs voisins

- Le serveur de données

Ce serveur gère les connexions avec les bases de données centrales à savoir la base de données de log, la base de données de recherche, la base de données des inventaires… .Chaque simulateur est accompagné d'un serveur de données .Ce serveur de données améliore les requêtes pour les simulateurs. Il y a plusieurs autres serveurs comme le serveur des Linden dollars (L$) qui gère les transactions financières des utilisateurs. Le Linden dollar (L$) est la monnaie dans le monde virtuel

L'utilisateur peut procéder à une demande de connexion, à la localisation des objets du jeu. Il peut aussi demander des informations sur les objets que son avatar rencontre.

- ✓ Les clusters de base de données

Du coté serveurs, nous avons aussi plusieurs clusters de bases de données permettant le stockage des objets du jeu. Parmi ces clusters, les principales sont :

- Le cluster de bases de données central

Il stocke des informations persistantes sur Second Life, incluant les profils des résidents (id, nom, date-de-création, région, sexe, position(x, y)), les groupes(id, nom…), les régions (id, nom, position en x, y, propriétaire, date de création, les utilisateurs ayant un accès,ou si l'accès est gratuit, les objets ,les droits accordés aux utilisateurs dans cette région…), les parcelles, les transactions financières de SL (L$).

- Le cluster de base de données des inventaires

Dans second life un inventaire désigne l'ensemble des objets que possède un utilisateur .Ces objets peuvent être des voitures, des habilles, des maisons … .L'arbre représentant l'ensemble des objets des utilisateurs du jeu est stocké sur un cluster de base de données appelé le cluster de base de données des inventaires .Et nous pouvons spéculer qu'une table ou relation de cet cluster stocke des lignes de cette forme.

Tableau 1 : Les attributs des objets du jeu Second Life

ID objet	Nom objet	Position(x, y) sur la grille	Date création	Propriétaire	créateur	Statut (en vente ou pas	région

- Le cluster de bases de données des login

Ce groupe de machines garde les informations sur les utilisateurs de Second Life c'est-à-dire le login, et le mot de passe de chaque utilisateur, la région (id, nom région, position) ou land de cet utilisateur, les régions où il est autorisé à se connecter.

- Le service de données transitoires

Nous avons aussi un cluster de machines qui hébergent sur leur mémoire vive les états transitoires des données. Les transactions sur ces machines nous permettent d'avoir des informations telles que les utilisateurs en ligne, les groupes, les chats … Ces données ne sont pas estoquées sur une base de données mais sont constamment rafraîchies par les simulateurs .En effet un simulateur met à jour la présence des agents toutes les minutes et d'autres enregistrements comme celui qui a rejoint tel groupe …

Apres l'architecture de l'application et le fonctionnement des différentes composantes de l'architecture, nous allons étudier les opérations de manipulation et de recherche de données dans Second Life.

3. Opérations de Manipulation et de Localisation des données dans Second Life

Les opérations de manipulations et de localisation des données se déroulent au niveau des clusters de bases de données centralisées de Second Life. Ces opérations sont regroupées sous forme d'unités appelées transactions.

Plus précisément une transaction est une unité de mise à jour composée de plusieurs opérations successives qui doivent toutes être exécutées ou pas du tout. En effet c'est une séquence d'opérations liées comportant des mises à jours ponctuelles d'une base de données devant vérifier les propriétés d'atomicité, de cohérence, d'isolation et de durabilité (ACID). Dans second life nous avons deux types de transactions :

- Les transactions de lecture

Ces transactions sont constituées d'opérations de lecture et ne modifient donc pas les objets des bases de données .Les opérations de connexion, de recherche et d'édit de données constituent ces transactions.

- Les transactions de mise à jour (lecture-ecriture).

En considérant que les transactions modifiant la position des avatars sont exécutées au niveau du service de données transitoires et que seules les dernières positions des avatars sont stockées sur les bases de données centrales, nous pouvons affirmer que les transactions de mise à jour de second life sont les transactions d'inventaires. Elles portent sur les objets du monde virtuel et sont réalisées par les utilisateurs lors de leur session .Les transactions financières qui se font en Linden $ font partie de ces transactions .Le Linden $ est la monnaie du jeu et permet d'acheter des objets .Ces transactions de lecture-ecriture sont constituées d'opérations qui se font dans des clusters de bases de données centralisées.

3.1. Les opérations de manipulation des données

Les différentes opérations de manipulation de données que l'utilisateur peut effectuer avec les objets du jeu sont :
-créer un objet ;
-détruire un objet ;
-déplacer un objet ;
-partager un objet avec un autre joueur ;
-changer le statut d'un objet (le mettre en vente) ;
-Acheter un objet ;
Nous considérons que les transactions de mise à jour sont constituées principalement par ces opérations et par objet nous entendons les avatars, les objets des avatars tels que les voitures, les lands …. En nous limitant à trois bases de données dans notre étude, à savoir la base des login, celle des inventaires, et la base de données centrale qui servent pour les recherches, nous allons expliquer les différentes actions générées dans les bases de données par ces opérations. Et ensuite nous allons expliquer les transactions de recherche ou la procédure de localisation des objets.

- Les opérations de manipulation des objets de second life

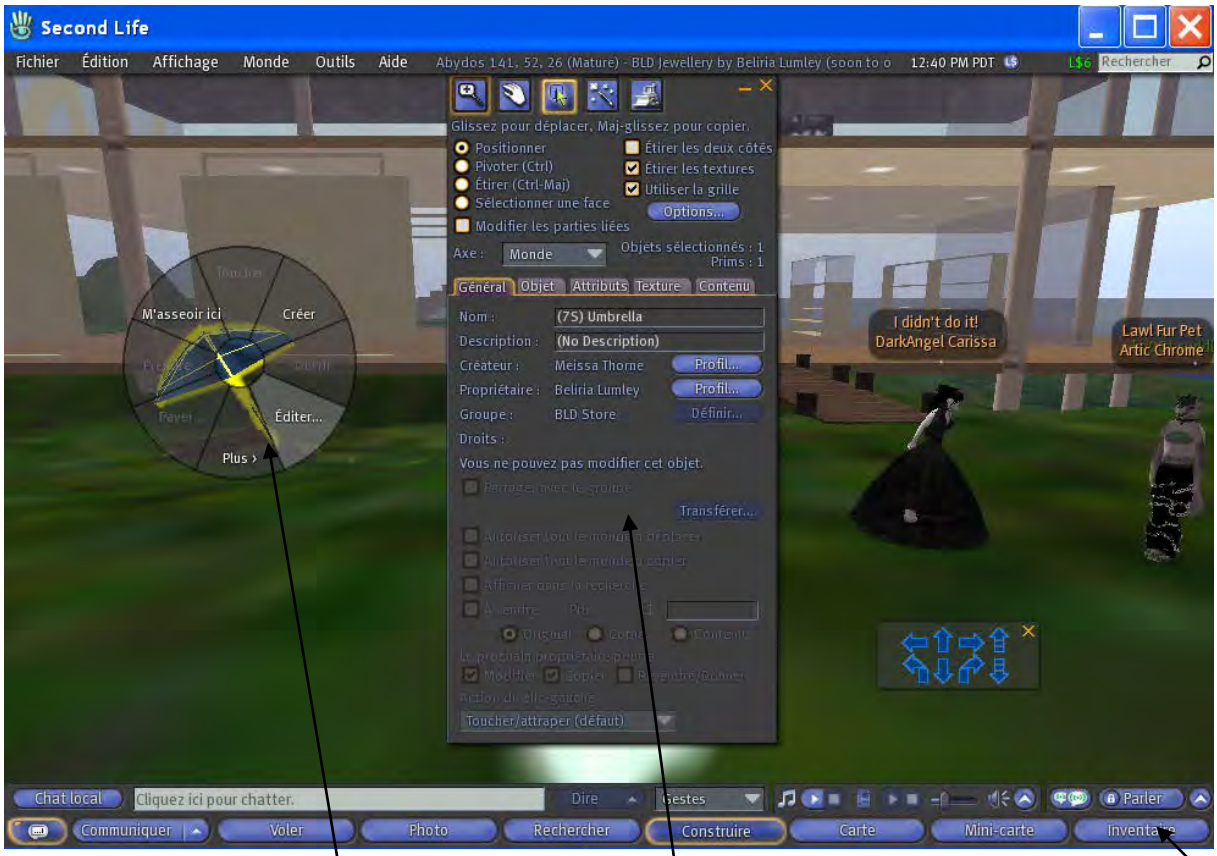
Ils existent plusieurs opérations de manipulation des objets qui se traduisent par des écritures sur les bases de données. Par exemple quand on crée en tant qu'utilisateur un objet, il y a des opérations d'écritures générées dans les différentes bases de données. Dans la base de

données des inventaires, une opération d'écriture, précédée d'une opération de lecture, va mettre à jour notre inventaire pour ajouter un nouvel objet (id, nom, position, etc.) dans nos objets. Dans la base de données centrales une opération d'écriture, précédée d'une opération de lecture, va ajouter ce nouve au objet (id, nom, propriétaire, région, position(x, y) etc.) parmi les objets du jeu.

Quand nous détruisons un objet aussi, une opération d'écriture va permettre d'effacer dans la base de données des inventaires, cet objet de notre collection. Si nous avions donné cet objet aux autres joueurs du même groupe que nous, ces joueurs auront toujours l'objet dans leur inventaire. Et donc une mise à jour sur l'attribut id_avatar de ces joueurs va se répercuter sur l'attribut représentant les propriétaires de l'objet au niveau de l'objet (Propriétaire).

Si nous changeons la position d'un objet, une opération d'écriture met à jour l'objet c'est-à-dire change son attribut position dans les bases de données.

La figure 2 ci-dessus montre au niveau de l'application cliente comment sont effectuées toutes ces opérations.



Balise permettant d'éditer un objet lors d'une opération d'inventaire

Balise utilisée pour manipuler un objet lors de la création de l'objet

Lister notre inventaire

Figure 2 : les différentes opérations de manipulations des données possibles

3.2. Les opérations de localisation des objets dans Second Life

N'oublions pas que nous considérons comme objets, les avatars, les régions (ensemble de lands), les groupes d'utilisateurs etc. Et rappelons encore que dans Second Life chaque objet a des attributs tels que son id, son nom, sa position sur la grille en (x, y), sa région, son créateur, son ou ses propriétaires, les utilisateurs autorisés à accéder à l'objet, etc. Au niveau de l'application cliente la localisation d'un objet se fait selon les étapes suivantes. D'abord une recherche par mot clé est effectuée par l'utilisateur pour voir la région où se trouve l'objet ou la position de l'objet sur la grille. Ensuite, il peut s'y « téléporter » pour pouvoir accéder à cet objet. On voit donc que, pour localiser un objet, des opérations de lectures sont faites au niveau du cluster de base de données central et du cluster des inventaires pour déterminer: si l'objet existe d'abord ensuite sa position sur la grille, la région où il se trouve et son propriétaire.

En résumé, nous pouvons donc dire que les transactions de second life sont généralement des transactions de lecture (localisation, connexion, lecture des information sur l'objet) ou des transactions de mise à jour ou transaction d'inventaire. Et parmi les transactions d'inventaire, on a les transactions financières en L$. Toutes ces transactions sont constituées d'opérations de lectures et d'écritures sur les clusters de bases de données.
Cependant, Second Life étant une application communautaire, il s'en suit que les transactions ou opérations des utilisateurs dans les bases sont fortuitement tributaires de cet aspect communautaire. C'est à dire chaque utilisateur a son groupe d'amis et ses objets accessibles à ses amis, sa région et les régions qu'il fréquente le plus souvent. Donc il est clair que les transactions de chaque utilisateur sont délimitées en fonction de certains critères surtout de proximité et/ou d'affinités.
Ainsi pour bien prendre en compte ces types de transactions, nous avons besoin de définir cette proximité pour en déduire les concurrences d'accès. Cette notion de proximité est appelée localité des accès dans Second Life.

3.3. Localité des accès aux objets du monde virtuel Second Life

Au niveau de l'application client de Second Life, un utilisateur peut accéder à une liste contenant les groupes, les contacts et les lands de l'utilisateur. Il peut donner au niveau de l'application cliente des droits d'édit sur ses objets à ses amis et aux membres de ses groupes.

Dans second life, le droit d'édit correspond au droit de création, modification et suppression sur les objets du jeu.

Supposons qu'un utilisateur se trouvant dans une région A, possède des objets dans cette région et d'autres dans une région B. Si cet utilisateur octroie des droits d'édits à des utilisateurs sur ses objets alors il y'aura une transaction lancée à partir de la région de l'utilisateur qui va modifier les objets se trouvant dans ces deux régions différentes pour permettre aux utilisateurs d'accéder à ses objets. Donc il devient difficile de parler, à partir de là, de localité des accès dans Second Life, étant donné qu'il peut arriver qu'une transaction lancée à partir d'un site *i* manipule des données d'un autre site *j*. Dans ce cas on aura, en parcourant les tuples de la transaction et en récupérant les attributs positions des objets non avatars de la transaction, des positions correspondant à plusieurs sites.

Cependant, une étude approfondie de l'application cliente de Second life, nous a permis de nous rendre compte qu'un utilisateur situé sur un land se trouvant dans une région donné ne peut pas manipuler des données situées dans d'autres lands. Et ne peut pas accéder à des objets situés sur un autre land. Il doit s'y téléporter (rejoindre la région) pour pouvoir y faire des manipulations.

De plus le but des utilisateurs de l'application n'est pas de partager des droits sur leurs objets avec les autres utilisateurs (ce qui peut créer une transaction manipulant des données éparpillées dans les sites), mais de posséder des objets et lands pour eux et d'y tirer des profits .Ils bénéficient de leur objets en les vendant à d'autres utilisateurs ou en créant ainsi un club (lieu de chat) attractif dans leur land.

En outre dans second life, du fait de l'aspect communautaire, la majeure partie des contacts d'un joueur fréquentent les même lands, les mêmes clubs et ont leur land situé à coté du land de ce joueur c'est-à-dire dans la même région que la région du joueur.

Nous en déduisons donc que les transactions manipulant des données reparties sur plusieurs sites sont négligeables devant les transactions manipulant des données localisées sur un espace limité (un land ou région). Et par conséquent nous pouvons parler de localité des transactions dans second life.

3.4. La limitation des accès des utilisateurs à une faible portion de données

Dans Second Life les accès sont les transactions ou opérations de manipulations des objets du monde virtuel (land, voiture, habits,…) que réalisent les utilisateurs .Les accès des joueurs aux objets génèrent des accès aux bases de données centrales .Et avant de voir dans quelle mesure les accès des joueurs peuvent être limités à une faible portion de données, nous allons essayer d'expliquer la notion d'accès à une base de données dans le cas général.

3.4. A Définition d'un accès à une base de données dans le cas général

Nous pouvons considérer la plus petite entité d'une base de données comme étant la granule .Et la granule peut être une page, une relation ou un tuple d'une relation .Un accès à une base de données signifie un accès à une ou plusieurs granules de la base de données, et nous avons plusieurs types d'accès parmi lesquels on peut citer

- L'accès informationnel (requête d'interrogation et réponse)
- L'accès transactionnel (mises à jour rapides en temps réel)
- L'accès décisionnel (exécution de requêtes complexes sur de gros volumes de données).

Nous nous limitons dans la suite de cette étude de second life aux accès transactionnels étant donné qu'avec Second Life les accès se font en ligne et sont de courte durée. Et que nous avons négligé les transactions qui portent sur plusieurs données se trouvant au niveau de l'espace logique dans des lands différents.

3.4. B Les conditions de la limitation des accès à une faible portion de données se trouvant sur une zone relativement petite.

Les activités principales des joueurs sont les activités commerciales et les rencontres dans les clubs .Ces activités se déroulent dans des clubs ou régions ou land possédés par des joueurs .Et chaque joueur a son ou ses groupes de contacts ou a mis et possède un inventaire (ensemble d'objets divers du jeu).

Quand l'activité d'un joueur lors de sa session se déroule entièrement dans une zone ou le nombre d'utilisateurs partageant cette activité est limité alors les accès générés par cet utilisateur se limiteront à une faible portion de données délimitées dans cet espace limité. En effet un utilisateur ne peut accéder qu'aux objets se trouvant sur le land où se trouve son avatar.

Par exemple, supposons que dans Second Life, un utilisateur soit un vendeur d'objets (voitures, maisons, produits) dans un land qui n'est visité que par un nombre limité d'utilisateurs lors de sa session. Alors il est évident que l'instance d'activité mené lors de cette session ne pourra générer qu'un nombre limité de transactions et donc d'accès aux données dans les bases de données centralisées et celles des inventaires.

Si un utilisateur participe à des rencontres entre joueurs dans un club donné alors il n'aura d'interaction qu'entre lui et les joueurs qu'il connaît. Et pour une des sessions données le nombre de transactions générées par ce joueur dans les bases de données centralisées, que ces transactions soit des transactions de recherche ou d'inventaires, est limité par le nombre d'objets possédés par ce groupe de joueurs dans ce club. Il s'en suit que ce nombre de transactions générées tend vers l'infiniment petit si le nombre de contacts du joueur tend vers zéro. Et ici encore, les accès se limiteraient à une faible portion de données délimitées dans un espace réduit.

4. Caractérisation des accès des utilisateurs de Second Life, Métrique pour quantifier la localité et la fréquence des accès et cas d'utilisations de cette métrique

4.1. Quelques définitions de base pour caractériser les accès de Second Life

Hypothèse de base

Chaque transaction est lancée dans un land et se limite aux objets de ce land. Le land est une partie de « l'espace géographique » .L'espace géographique est un espace cartésien bidimensionnel constitué de plusieurs points. Chaque objet est localisé par sa position spatiale

dans cet espace. Nous donnons quelques définitions qui nous permettrons de caractériser les accès des utilisateurs:

Définition 1. Position d'une transaction de Second Life

La position d'une transaction lancée à un instant t est le **barycentre** de l'ensemble des objets auxquels accède la transaction. Quand une transaction porte sur un seul objet on lui associe la position spatiale de cet objet. La position d'une transaction est notée P (Ti).

Définition 2. Distance entre deux transactions

La distance entre deux transactions T_1 et T_2 est la distance euclidienne entre les positions spatiales des deux transactions. Elle est notée $D(T_1, T_2) = d(P(T_1), P(T_2))$

Définition 3. Transactions ε - proches

Soit ε un entier strictement positif.
Deux transactions sont ε -proches si la distance entre leurs positions est inférieure à ε.
Il est évident que plus deux transactions sont proches, plus est grande la probabilité qu'elles portent sur les mêmes données.
Une transaction est ε -proche à une position P si la distance entre la position de la transaction et la position de P est inférieure à ε .

Définition 4. Découpage de l'espace logique de Second Life en zones pour localiser les accès

Dans le cas des applications web 2.0 plus précisément avec les jeux virtuels, l'espace géographique est un square [0,1]*[0,1] constitué d'un ensemble de points Pi. Nous considérons le milieu du square, le point O de coordonnées (1/2, 1/2) comme le point initial de l'espace. En prenant les médiatrices des cotés du square qui passent par O, nous obtenons quatre carrés au sein du square. Nous pouvons noter les centres de ces carrés les points O^1_i

$_{(1<=i<=4)}$. En prenant encore les médiatrices des cotés des quatre carrés formés qui passent par ces points $O^1{}_{i\ (1<<i<<4)}$ nous obtenons quatre autres carrés au sein de chacun des quatre carrés. Les centres de ces quatre nouveaux carrés seront appelés les points $O^2{}_{i\ (1<<i<<4)}$. . Nous pouvons continuer ainsi ce découpage jusqu'à n et obtenir les points $O^n{}_{i\ (1<<i<<4)}$.

Le nombre de découpages augmente avec la densité de l'occupation de l'espace logique par les avatars qui sont en ligne.

Le coté du square mesure 1. Le point initial O a pour coordonnées $X^0 = 1/2$ et $Y^0 = 1/2$. Et à chaque découpage, nous obtenons quatre points de coordonnées $X^n{}_{i\ (1<=i<=4)}$, $Y^n{}_{i\ (1<=i<=4)}$. Avec n le numéro du découpage.

$X^{n+1}{}_{i\ (1<=i<=4)}$, $Y^{n+1}{}_{i\ (1<=i<=4)}$ les coordonnées des centres $O^{n+1}{}_{i\ (1<=i<=4)}$ obtenus au n+1 nième découpage.

Nous avons au découpage numéro n+1 :

$$X^{n+1}{}_i = X^n{}_i \pm (1/2)^{n+1} \qquad (1)$$

$$Y^{n+1}{}_i = Y^n{}_i \pm (1/2)^{n+1}$$

Le schéma ci-dessous illustre le découpage effectué

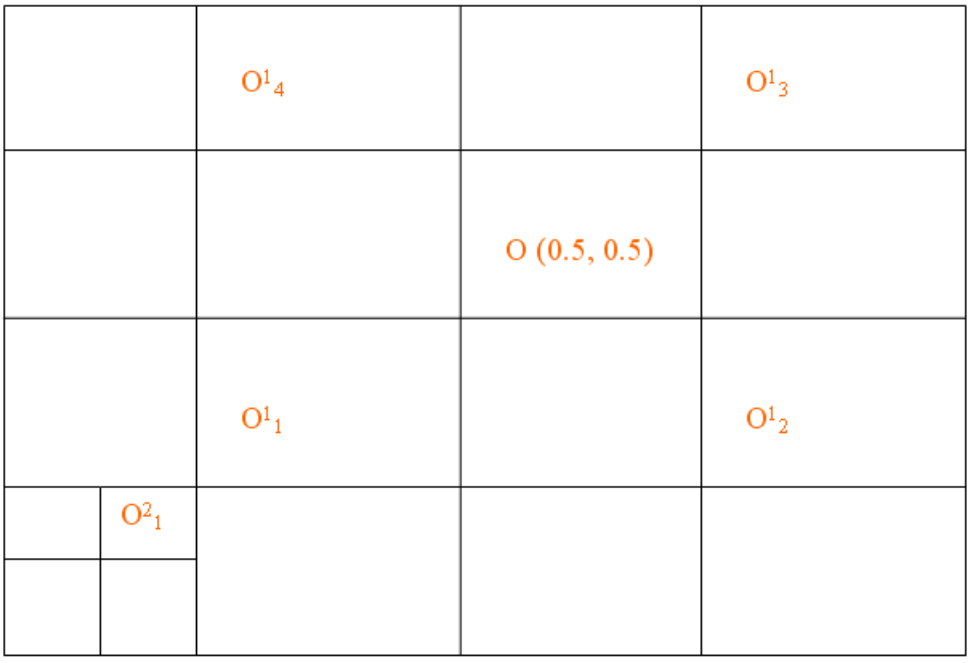


Figure 3 : Découpage de l'espace logique de Second Life

Définition 5. Zone de positions

Une zone spatiale est un carré constitué d'un centre $O^n_{i\ (1<=i<=4)}$ de coordonnées (X_n, Y_n) et en général d'un coté de $1/2^n$ où n est le nombre de découpages effectués. Chaque point du square appartient à la zone qui contient sa position spatiale. On représente une zone de positions sous la forme

$Z_n = (O^n_{i\ (1<= i <=4)}\ (X^n_{\ i\ (1<= i <=4}, Y^n_{\ i(1<= i <=4)})\ ,\ 1/2^n)$. (2)

Définition 6. Localité d'une transaction

Chaque transaction web 2.0 est caractérisée par sa position dans l'espace géographique.

La localité d'une transaction correspond à la zone à laquelle appartient la position de la transaction.

4.2. Quantification de la localité des accès et de leur fréquence

Pour une période donnée D compris entre deux instants t1 et t2, nous pouvons allouer à chaque zone Z_n l'ensemble TR_n des transactions trx telles que la position de trx appartienne à Z_n. Cela est possible étant donné que nous connaissons la position et l'instant de chaque transaction.

$TR_n = \{trx / position\ (trx) \in Z_n\}$

Pour chaque zone Z_n, nous pouvons connaître le nombre $p\ (Z_n)$ de transactions lancées dans la zone pendant la période D. Ce nombre correspond à la cardinalité de l'ensemble TR_n et nous permet de quantifier la localité des accès à la zone Z_n.

Nous pouvons définir une métrique

Loc-acc (Z_n, D) = Erreur ! Signet non défini.**{p (Zn), (p (Z_n)/ D)}**

Où (p (Zn)) est le nombre d'accès dans Z_n durant la période D et le rapport $p\ (Z_n)/D$ la fréquence des accès à Z_n durant la période D.

Pour chaque position K(x, y) moyenne d'un ensemble de données d'une zone, nous pouvons savoir le nombre des accès lancés et qui sont $\mathcal{E}$-proches à cette position. Cette information pourra être utilisée si nous avons besoin de découper une zone en un ensemble de petites zones. Dans ce cas, notre métrique sera appliquée au point K(x, y) ainsi :

Loc-acc (K(x, y), D) =Erreur ! Signet non défini. **{p (K(x, y), (p (K(x, y)/ D) }** (3)

Où **p (K(x, y)) = Card (TR_n) avec TR_n = {trx telle que |position (tr)-K(x, y)| < $\mathcal{E}$ }**

p (K(x, y) est le nombre d'accès $\mathcal{E}$-proches à K(x, y) et le rapport p (K(x, y)/D nous donne la fréquence des accès $\mathcal{E}$-proches à K en supposant les accès $\mathcal{E}$-proches à K(x, y) ont presque la même position.

Etant donné qu'avec les applications web2.0, les accès se limitent à la zone où ils sont lancés c'est-à-dire à une faible portion de données parmi les données ayant des attributs positions localisés dans cette zone, nous pouvons éviter de centraliser le contrôle des accès aux données dans l'architecture de base de données pair-à-pair que nous avons.

L'idée est la suivante :

D'abord nous allons utiliser l'attribut position des données pour les mettre sur une base de données pair-à-pair, ensuite nous allons nous baser sur la métrique Loc-acc quantifiant la localité et la fréquence des accès pour compartimenter de manière dynamique les données et allouer à chaque compartiment de données un serveur et un gestionnaire de transactions. Ainsi nous aurons des zones de concurrence avec des données fortement accédées et des zones délaissées. Et en fonction des accès, les zones seront modifiés (Verrouillage du dictionnaire de données et modification de ce dernier) à chaque fois que le test se fera et un serveur leur sera alloué.

4.3. Cas d'utilisations de notre métrique Loc-acc dans des situations concrètes

Dans Second Life les crash sont le plus souvent dus à un nombre de transactions très élevé à tel point que le serveur de données ne peut plus satisfaire aux demandes des utilisateurs .Pour pallier à ce problème, pour chaque zone nous calculons la métrique Loc-acc .Ainsi en comparant la métrique de chaque zone aux métriques des autres zones, on pourra mettre en attente les transactions des zones dont les Loc-acc dépasseraient une limite précisée. Durant ce temps d'attente, on pourra exécuter les transactions dans les zones qui ont des Loc-acc en dessous de la limite de Loc-acc fixée .Et après ces zones peuvent être redémarrées. Ainsi donc en utilisant le Loc-acc on peut résoudre le problème des crashs. Un autre cas d'utilisation de la Loc-acc peut être l'amélioration des temps d'attentes pour les transactions conflictuelles. En effet des utilisateurs ayant des données liées, se trouvant dans des zones différentes peuvent lancer des transactions de mis à jour de leur inventaire. Un problème de scheduling au niveau de la base de données centrale peut arriver lors de l'exécution de ces transactions. Donc parmi les zones possédées par un même utilisateur de second life, par exemple un développeur, si certaines voient leur Loc-acc dépasser la limite fixée, alors ces zones peuvent

être deconnectées pour laisser aux autres zones de cet utilisateur qui ont des Loc-acc normales terminer leurs transactions.

La métrique Loc-acc peut permettre dans les architectures client/serveur d'éviter des situations de crise comme les crashs, les durées trop longues dues aux conflits entre transactions. Mais cette gestion des transactions doit se faire d'une manière transparente. Nous allons donc faire migrer l'architecture centralisée de Second Life dans une architecture pair-à-pair où nous pourrons nous servir de notre métrique Loc-acc pour gérer les accès sans que ne nécessite d'arrêter de déconnecter des zones ou lands

Conclusion

Avec le découpage proposé de l'espace géographique, nous avons obtenu un réseau maillé avec des zones Z_n. Chaque zone a une position centrale (X^n_i, Y^n_i) et un ensemble de positions déterminées par des coordonnées cartésiennes. Le nombre d'accès dans chaque zone p (Zn) varie d'une zone à l'autre, et les accès d'une même zone ont la plus grande probabilité de concerner les mêmes données et se limitent aux données de cette zone d'après notre hypothèse de base.

Notre architecture pair-à-pair devra donc être structurée en zone et chaque zone devra regrouper les données les plus proches. Chaque zone est gérée par un nœud correspondant au centre de la zone et va contenir des nœuds objets ou nœuds de données et des nœuds avatars. Les données seront affectées aux nœuds de données du graphe en fonction de leurs coordonnées.

Ce type d'architecture est l'architecture d'un réseau pair à pair basé sur les diagrammes de Voronoi. En effet, un diagramme de Voronoi d'un ensemble d'objets est une séparation en zones disjointes de l'espace de nommage.

Chaque objet est associé à une zone, qui contient tous les points qui sont plus proche de lui que de tout autre point.

Les voisins de Voronoi sont les objets dont la zone partage une frontière commune.

Nous allons ainsi donc nous baser sur les bases de données pair-à-pair structurées en diagrammes de Voronoi pour proposer un traitement décentralisé des accès de Second Life. Mais avant cela, nous commencerons par faire une présentation détaillée de ces digrammes de Voronoi

Chapitre2

Les Applications pair-à-pair (P2P) basées sur les diagrammes de Voronoï

1. Introduction

La gestion de données suscite un intérêt important pour les concepteurs de réseaux et d'applications réparties à large échelle. Particulièrement, des réseaux et protocoles pair-à-pair, permettant des recherches de ressources ou objets répartis sur un ensemble de noeuds (instances d'une application), ont été proposés afin de pallier le problème de passage à l'échelle. Ces protocoles se proposent de lier les nœuds logiques, et de laisser chacun gérer le maintien et le service d'un sous-ensemble d'objets de l'application. Chaque noeud ne connaît qu'un ensemble restreint de *voisins*, avec lesquels il peut communiquer, et la communication s'effectue par routage ou inondation au sein du réseau défini par cette relation de voisinage.

On distingue deux grandes familles de réseaux pair-à-pair, les réseaux structurés et les réseaux non structurés. Dans les réseaux non structurés, les relations entre noeuds ne suivent pas de structure prédéfinie [11]. De la même manière, la responsabilité d'un objet n'est pas nécessairement réservée à un nœud particulier du réseau, et plusieurs d'entre eux peuvent posséder des copies d'un même objet. Ces réseaux ont l'avantage d'être simples mais ne permettent pas d'assurer l'obtention d'une donnée pour toute requête, même si une telle donnée est présente dans le réseau. En effet, les mécanismes de recherche typiquement utilisés au sein de ces réseaux, tel l'inondation ou les marches aléatoires, ne peuvent garantir l'exhaustivité des résultats que si l'ensemble des noeuds du réseau est interrogé. À l'inverse, un réseau structuré impose que les liens entre les nœuds logiques suivent une organisation particulière, qui met en oeuvre une structure de données. Elle permet d'améliorer la performance des recherches de données dans le réseau et elle impose le placement des objets sur un des noeuds participant à la structure, permettant d'assurer l'exhaustivité des résultats des requêtes. Les réseaux structurés ont été principalement conçus autour des tables de hachage distribuées (DHT), où la responsabilité d'un objet est confiée à un noeud du réseau, selon une mesure de proximité dans un espace de nommage (un anneau [24], une liste chaînée [11] ou un espace euclidien [23] sont des espaces de nommage caractéristiques). Les identifiants dans cet espace de nommage des noeuds physiques ainsi que ceux des objets partagés sont obtenus grâce à une fonction de hachage uniforme appliquée à leur descripteur

(adresse IP, nom du fichier, . . .). Ceci permet d'assurer une répartition uniforme du nombre d'objets hébergés par chacun des noeuds. Toutefois, ces fonctions de hachage ne préservent pas l'ordre sur les identifiants. Deux noms d'objets proches ne donnent pas deux identifiants consécutifs dans l'espace de nommage. La recherche de données se fait donc selon le formalisme de recherche par clé, en connaissant à l 'avance l'identifiant de l'objet ; des requêtes par plages de valeurs ne sont pas permises directement par la structure du réseau.

Contrairement aux approches citées précédemment, qui proposent de lier des noeuds physiques hébergeant des copies de données, avec les réseaux pair-à-pair basés sur les diagrammes de Voronoi, ceux sont les objets partagés eux-mêmes qui sont liés, au sein d'un espace de nommage reflétant la sémantique de l'application. Ils existent deux exemples de tels réseaux à savoir VORONET [10] et RAYNET [18]. Le premier à savoir VORONET est utilisé pour les transactions de type OLTP et le second RAYNET pour les requêtes persistantes. Comme les transactions web 2.0 sont de type OLTP. Nous allons donc utiliser VORONET.

VORONET fait partie des réseaux pair-à-pair basés sur des diagrammes de Voronoï. C'est un réseau objet à objet, où chaque objet est décrit par un ensemble d'attributs. Chaque objet est placé de manière directe, sans utiliser de fonction de hachage uniforme, dans un e space euclidien de dimension 2. Les relations de voisinage dans le réseau sont définies par adjacence dans le diagramme de Voronoï des objets dans le plan. Ainsi, les objets avec des valeurs d'attributs similaires sont adjacents dans le réseau ce qui permet un support naturel aux requêtes par plages de valeurs. De plus, Voronet n'impose pas de placement aux objets sur les noeuds physiques : les objets sont maintenus par les nœuds physiques qui les partagent. La répartition des objets dans l'espace peut être éparse, car dictée par l'application.

Dans la suite, nous allons présenter les notions fondamentales de Voronet à savoir les diagrammes de Voronoï [8], le modèle <<petit-monde>> de Kleinberg [20], la construction de Voronet, et les mécanismes de routage et d'interrogation.

2. Notions fondamentales

Dans cette partie, nous présenterons dans un premier temps les diagrammes de Voronoï et la triangulation de Delaunay, puis le modèle << petit monde >> de Kleinberg

2-1. Définition d'un diagramme de Voronoï

Un diagramme de Voronoï est une partition de l'espace *V (P)* pour un ensemble fini de points P = {p_1, . . ., p_n} auquel est associé une mesure de distance d. Si d (p1, p2) est la distance euclidienne entre p1 et p2, chaque point pi est associé à une zone R (pi) définie comme suit :

R (pi) = {p | d(p, p_i) ≤ d(p, p_j), $\forall$ j ≠ i} (la zone R(p_i) est constituée de tous les points du plan plus proches de p_i que de tout autre point). La partition de l'espace {R (p1), . . ., R (pn)} forme le diagramme de Voronoï de P. On nomme *arête de Voronoï* la frontière commune à deux régions, et *sommet de Voronoï* le point d'intersection de trois arêtes ou pl us .La *triangulation de Delaunay D (P)* est le graphe associé au diagramme de Voronoï. Soit C ($\Delta p_1 p_2 p_3$) le cercle circonscrit au triangle formé par les points p1, p2 et p3. La triangulation de Delaunay est alors l'ensemble des triangles { $\Delta pipjpk$ | $\forall$ p_l ∈ **Erreur ! Signet non défini. Erreur ! Signet non défini.Erreur ! Signet non défini.***P* \ {pi, pj, pk} : pl ∉ C ($\Delta pipjpk$)}.

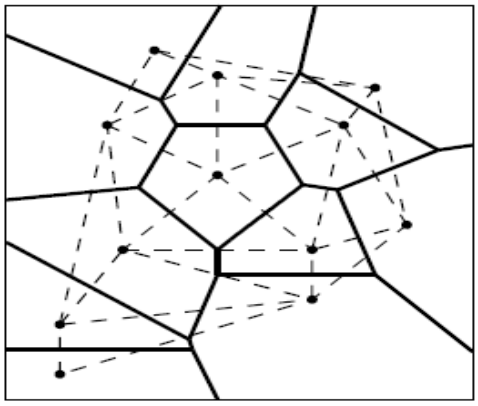
La relation entre V (P) et D (P) est directe :

Erreur ! Signet non défini. $\forall$ i, j : (**Erreur ! Signet non défini.** $\exists$ {R(pi),R(pj)} ∈ V(P), R(p i) ∩ R(pj) ≠ Φ ⇔ $\exists$ pk, $\Delta p_i p_j p_k$ ∈ D(P)).

La figure 1(a) montre un exemple de diagramme de Voronoï et de triangulation de Delaunay pour un ensemble de points dans le plan. Dans le plan, le graphe de Delaunay est un graphe planaire et présente une distribution des degrés indépendante de |P| : la moyenne des degrés des sommet est inférieure à 6, donc en O (1). Toutefois, dans des cas extrêmes (points situés exactement sur un cercle dont le centre est lui même un point par exemple), le degré d'un sommet peut être borné par (|P|−1).

FIG 4- a)

FIG 4 - b)



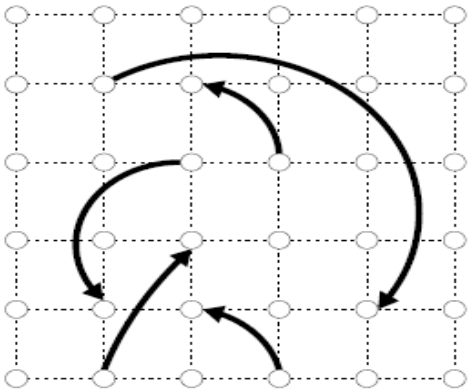


Figure. 4 – a (gauche) : Un exemple de diagramme de Voronoï (lignes pleines) et de la triangulation de Delaunay associée (lignes discontinues) ; Figure.4- b. (droite) : Un graphe en topologie de grille, augmenté de quelques liens longs << à la Kleinberg >>.

2.2. Modèle << petit-monde>> de Kleinberg

Les modèles dits << Petit-monde >> permettent de comprendre les raisons qui font que la plupart des réseaux d'interaction présentent à la fois des chemins courts (diamètre faible et d'ordre logarithmique) et la propriété de navigabilité (un processus de routage glouton (chaque nœud transmet la requête à son voisin qui est le plus proche du point de destination) est assuré de trouver le sommet le plus proche de la cible demandée, en un nombre d'étapes poly logarithmique en la taille du réseau [20, 21]). J. Kleinberg a proposé un modèle de construction de graphes qui présente ces propriétés de chemins courts et de navigabilité [20]. Le modèle s'appuie sur un tore n × n, au sein duquel chaque sommet possède une arête vers ses quatre voisins directs, ainsi qu'un *lien longue distance* permettant d'assurer les propriétés << petit-monde >>. La longueur l de ce lien longue distance (distance euclidienne entre le sommet qui le crée et le sommet distant choisi) est générée avec une probabilité $Pr[l] \propto l^{-s}$ (dans notre travail, s=2). La distribution de ces longueurs suit une loi de puissance :

Il existe un petit nombre de << grands >> liens longs et une majorité de << petits >> liens longs qui assurent la convergence du routage et une division de l'espace parcouru à chaque saut d'ordre logarithmique. La figure 1(b) montre un exemple de réseau de Kleinberg, pour lequel seuls quelques liens longue distance ont été représentés pour préserver la lisibilité. Dans [22], l'auteur montre que choisir s = 2 pour le plan assure les propriétés de chemin court et de navigabilité, et une généralisation à un espace à d dimensions est possible en choisissant s = d, et permet des chemins découverts en $\theta(\log^2$ **Erreur ! Signet non défini.**$\frac{n}{k})$, où k est le nombre de liens longue distance.

3. Voronet : un réseau objet à objet

3-1. vue d'ensemble

Voronet lie les objets de l'application (ensemble *O*) dans un espace euclidien qui est directement l'espace de désignation de ces objets. Chaque objet est représenté par exactement *a* attributs : l'espace dans lequel les objets sont placés est de dimension a. Dans ce travail nous nous limitons à *a* = 2 étant donné que les coordonnées des objets web sont des coordonnées cartésiennes et que ceux sont ces objets qui seront stockés au niveau de notre espace logique, nous avons jugé préférable que notre espace logique soit un espace bidimensionnel.

Chaque objet o possède une vue partielle du système réparti, constituée de trois ensembles de voisins, les voisins directs, les voisins longues distances et les voisins proches.

Voisins directs

Les voisins directs ou voisins de Voronoï sont les objets reliés à *o* dans la triangulation de Delaunay (les objets *o′* dont la zone R (*o′*) partage une arête de Voronoï avec R (*o*)). Comme le degré d'un sommet dans le graphe de Delaunay, le nombre de voisins directs moyen est en O (1), et ne dépend pas du nombre d'objets ni de leur distribution. La création et la maintenance de ces liens se font lors de l'ajout ou du retrait d'objet dans le réseau logique

Voisins longues distances

Les voisins longues distances sont nécessaires pour assurer un routage efficace et pour garantir la complétude des réponses. L'ajout d'un ou plusieurs liens longue distance par objet permet d'assurer la navigabilité et la découverte de chemins poly logarithmiques dans le réseau, quelque soit la distribution des objets dans l'espace. La taille de cet ensemble de voisins est fixe. Les liens longs de chaque objet *o* pointent sur les voisins longues distances de cet objet.

Voisins proches

Les voisins proches permettent d'assurer la convergence du routage quelque soit la distribution des objets dans l'espace. Cet ensemble est constitué de tous les objets o′ tels que d (o, o′) < d_{min}, avec d_{min} =**Erreur ! Signet non défini.** $\frac{1}{\pi * N_{max}}$, où N_{max} est une constante représentant le nombre maximal d'objets pour lequel le routage en O (log^x(|O|) est garanti. d_{min} est très petit par rapport à la taille de l'espace, et pour toute distribution raisonnable dans le plan, le nombre de voisins proches est en O(1) pour $|O| \leq N_{max}$. Si une zone de l'espace est particulièrement peuplée, alors les objets au centre de cette zone peuvent avoir un nombre important de voisins proches, mais ceci permet de conserver les propriétés théoriques d'efficacité du routage indépendante de la distribution des objets dans l'espace. Toutefois, il est possible d'imposer une borne supérieure sur la taille de cet ensemble

Figure 2 Exemple de vue pour un objet o dans Voronet

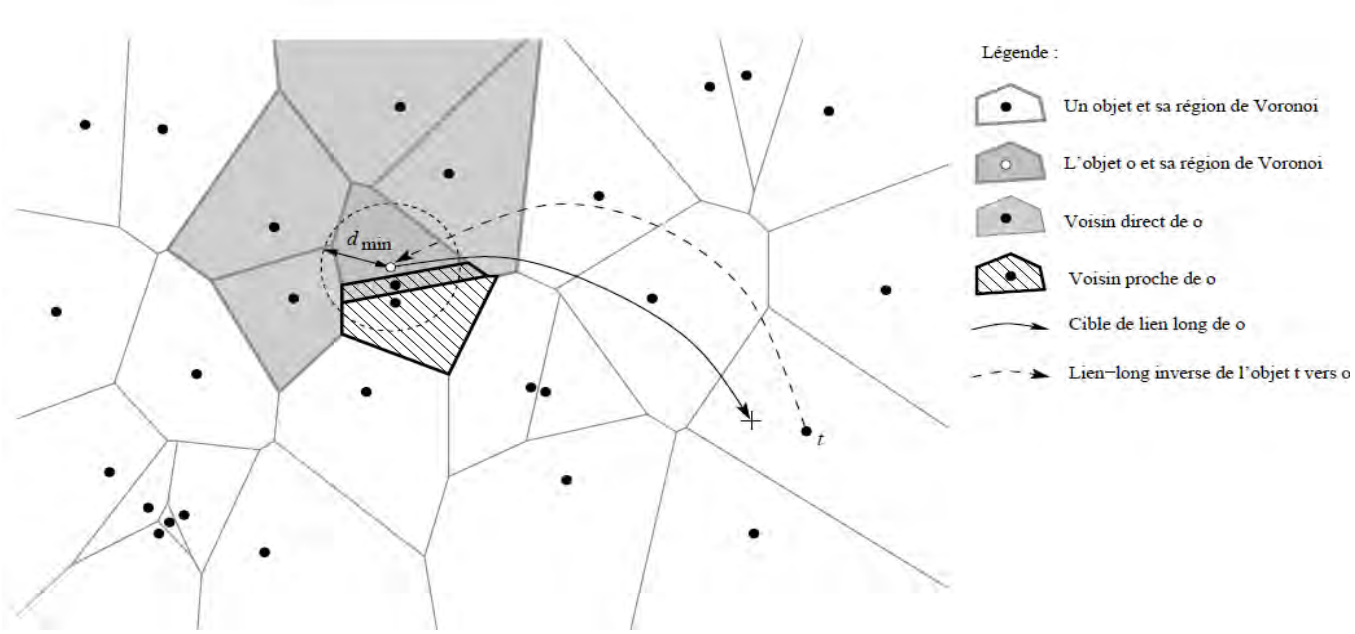


Figure 5 : vue d'ensemble du système par un objet

3-2. Construction du réseau logique

La construction des ensembles de voisins directs consiste à générer le diagramme de Voronoï des objets s'insérant dans le réseau. Cette génération se fait de manière incrémentale et l'action primordiale est de générer les propriétés d'adjacence des zones, plus que de générer de manière exacte les zones elles-mêmes.

3-2.1. Insertion d'un objet

Un objet o′ qui s'insère dans le réseau logique doit connaître un objet quelconque du réseau, auquel il demande un routage vers sa position dans l'espace, déterminée par la valeur de ses attributs. Le routage s'achève de manière déterministe sur l'objet o tel que o′ ∈ R (o).

La figure 3 montre en partie gauche le déroulement de ce routage. Cet objet o est responsable de l'insertion de o′ dans le réseau logique, et de la modification de zone et ensemble de voisins pour lui-même ses voisins directs, comme le montre l'exemple de la partie droite de la figure 3. La méthode la plus simple pour calculer la zone de Voronoï du nouveau point consiste à partir de l'intersection de la bissectrice du segment [o, o′] et de la zone R (o). Successivement, pour chaque zone R (o″) voisine de R(o), l'intersection entre R(o″) et la bissectrice du segment [o′, o″] est calculée, jusqu'à ce que l'algorithme retombe dans la zone R (o). Toutefois, cette méthode est sensible aux propagations d'erreurs d'arrondis. La figure 4.a montre un cas d'erreur possible : les calculs d'intersection effectués avec des réels cumulent leurs approximations, et la zone générée n'est pas cohérente. Afin de pallier les problèmes posés par les propagations d'erreurs de calcul, Voronet utilise un algorithme de calcul de zones fondé sur la méthode de Sugihara et Iri. Le principe est illustré par la figure 4.b : l'objet o, en charge de l'insertion du nouvel objet o′, crée un arbre indiquant les sommets de Voronoï à supprimer (nœuds internes de l'arbre) ainsi que les arêtes de Voronoï à supprimer (arêtes internes) ou à scinder pour créer la nouvelle zone (arêtes liant les feuilles). Le sommet s de cet arbre est, parmi l'ensemble des sommets définissant la zone de o, celui qui est numériquement le plus sûrement inclus dans la nouvelle zone R (o′) (pour lequel l'inclusion dans le cercle circonscrit du triangle correspondant aux trois zones reliées par le sommet de Voronoï est la plus sûrement établie).

La construction de l'arbre se fait ensuite selon les propriétés combinatoires du diagramme de Voronoï. Enfin, les arêtes de l'arbre situées au dernier niveau sont scindées pour assurer que chaque nouvelle arête constituant la zone R (o′) et jouxtant la zone R (o″) soit située sur la droite médiatrice de [o′, o″]. Ainsi, il n'y a pas de propagation d'erreurs, et si la zone peut ne pas être exactement calculée, les relations d'adjacence entre zones sont exactes, ce qui est suffisant pour la construction du réseau.

3-2.2. Gestion des liens proches

Lemme1

L'ensemble des liens proches est construit à partir des liens proches des voisins directs du nouvel objet o′. En effet, une fois les voisins directs déterminés, les voisins proches (dans un rayon de dmin) sont soit les voisins directs eux-mêmes, soit leurs voisins proches.

L'objet qui insère o′ lui fournit un premier ensemble de liens proches et, si nécessaire, lui indique le sous-ensemble de ses v oisins auxquels o′ doit demander ses liens proches (ceci n'est nécessaire que si le disque de rayon d_{min} centré en o′ n'est pas inscrit dans R (o′) **Erreur ! Signet non défini.** ∪ R(o)).

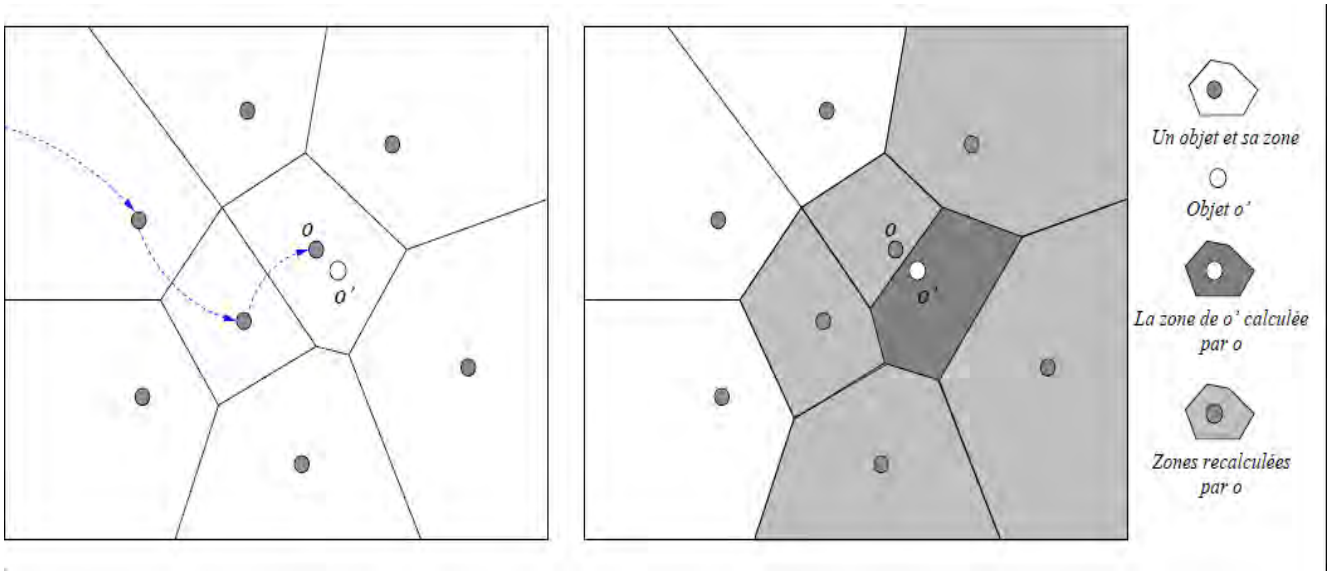


Figure 6 : Insertion d'un objet o′ par l'objet o (o′ ∈ R (o)).

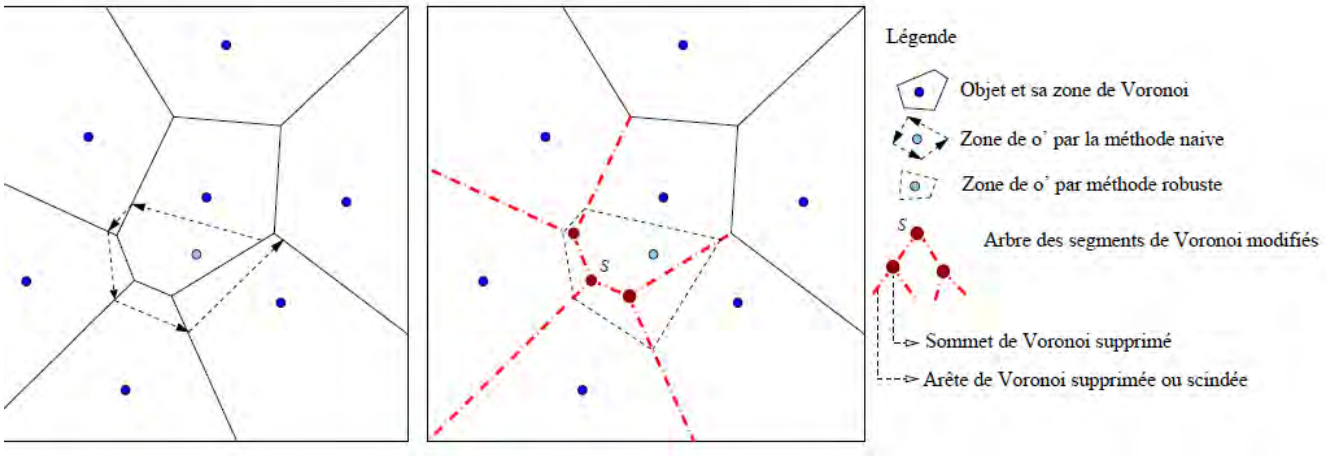


Figure 7 : Calcul d'une nouvelle zone : a. (gauche) méthode naïve avec propagation d'erreurs ; b. (droite) méthode robuste par construction de l'arbre des arêtes de Voronoï modifiées.

3-2.3. Suppression d'un objet

La suppression d'un objet est similaire : il suffit que l'objet o′ qui souhaite quitter le réseau logique calcule quelle portion de sa zone R (o′) doit être confiée à chacun de ses voisins directs. Le calcul de ces zones s'effectue en générant le diagramme de Voronoï partiel de la zone R (o′). L'objet o′ envoie ensuite une notification à chacun de ses voisins, qui modifient leurs zones en y ajoutant la portion qui leur est donnée par o′ et mettent à jour leur ensemble de voisins directs en conséquence.

3-2.4. Le mécanisme des liens longues distances

Le lien longue distance a pour objectif de permettre un routage efficace entre tout couple (objet, point) dans l'espace.

Lorsqu'un objet o a terminé son insertion locale (voisins directs et voisins proches), il choisit une cible pour son lien long. La longueur l de ce lien longue distance est choisie avec une probabilité $Pr[l] \propto l^{-s}$ (s = 2 dans le plan). La distance minimum est fixée à d_{min}, puisqu'un lien plus petit que d_{min} n'a aucun intérêt. La direction θ prise par le lien est choisie de façon totalement uniforme ($Pr[\theta] \propto 1$). Du choix de l et de θ résulte un point dans le tore p_{dest}. Il est important de noter que ce point ne correspond pas nécessairement à un objet dans le réseau. Toutefois, $p_{dest} \in R(o_{dest})$. Cet objet o_{dest} est choisi comme destination du lien longue distance, tout en gardant comme destination le point p_{dest}. Une illustration est donnée par la figure 2 : l'objet o tire le point dénoté << Cible de lien long de o >> et le voisin effectif est t. À tout moment, la propriété suivante est vérifiée: le voisin longue distance est toujours, parmi l'ensemble des objets O, l'objet o_{dest} tel que $p_{dest} \in R(o_{dest})$. Afin de maintenir cette propriété, lorsque o choisit sa destination, il route un message de création de lien long vers p_{dest} : un lien entre o et o_{dest} est créé, et o_{dest} conserve un *lien long inverse* de o_{dest} vers o. Lorsque o_{dest} insère un nouvel objet o′, il lui délègue le sous ensemble de ses liens longs inverses tels que $p_{dest} \in R(o')$. L'objet o′ notifie alors les objets à la source de ces liens longue distance de leur changement de voisin.

3.3. Le routage dans Voronet

Dans Voronet, c'est le routage glouton qui est utilisé. Le routage glouton est un processus qui consiste à faire transiter un message dans le réseau entre une source et une destination : chaque objet retransmet le message à son voisin le plus proche de la destination, sans

conserver d'informations ni dans le message ni sur l'objet lui-même. La topologie du réseau doit assurer qu'un message arrive exactement à destination, et qu'il n'y ait pas de cycle dans la chaîne de retransmission. Afin de mettre en oeuvre les principes << petit-monde >> à savoir les chemins courts et la navigabilité, il est nécessaire que le réseau sous-jacent permette à tout routage glouton le succès en un nombre d'étapes fini et sans cycle. C'est le cas du diagramme de Voronoï sur lequel est basé Voronet. Dans un diagramme de Voronoï sur un ensemble de N points distribués uniformément dans le plan, le routage glouton réussit en O (√N) étapes. Dans la topologie générale du diagramme de Voronoï, la destination est un point p dans la zone d'un objet existant. Alors, soit l'objet pointé divise la distance restante par $\frac{5}{6}$, soit la destination est très proche de la zone de Voronoï de l'objet o pour lequel p $\in$ R (o). Dans ce dernier cas, il est possible ([6]) de rejoindre directement R(o), la zone destination. Dans le cas contraire, la division par $\frac{5}{6}$ de la distance restant à parcourir peut être vue comme une << super-étape >> .Nous montrerons dans les parties qui suivent que le nombre de ces super-etapes H est borné : $H \leq \frac{\ln \sqrt{2}\pi N_{\max}}{\ln\left(\frac{6}{5}\right)}$ et a une espérance E majorée par $\alpha * \ln^2\left(N_{\max}\right)$. Nous ne détaillerons pas toutes les preuves mais elles sont toutes disponibles dans [10].

3.3.1. Choix d'un lien long vers un voisin éloigné

Quand un objet joint le réseau, il doit trouver un voisin éloigné .Cela est réalisé en utilisant la fonction Choose-LRT décrit par l'algorithme 1

Tableau 2 : Algorithme 1 : Algorithme pour trouver le voisin éloigné

Choose-LRT ()

Choisir a avec une probabilité uniforme dans [ln (dmin), ln ($\frac{1}{\sqrt{2}}$)]

Choisir δ avec une probabilité uniforme dans [0, 2π]

Prendre δ = (e^a cos (θ), e^a sin (θ))

Prendre LRT = *CurrentObject.coordinates*+ δ

return

Le lemme suivant donne la distribution de probabilité créée par la fonction Choose-LRT

Lemme 2

En utilisant la fonction Choose-LRT ,la probabilité que LRT(x) appartienne à une petite surface *dS* située à une distance d d' un point x est donnée par $\frac{dS}{K*d^2}$ où K=

$$\frac{1}{2\pi \ln\left(\frac{\pi|o|cibles}{\sqrt{2}}\right)} \qquad (4)$$

Un résultat intéressant peut être déduit de ce lemme, qui donne une borne inférieure de la probabilité de choisir LRT(x) dans un disque donné

Lemme 3

La probabilité pour LRT(x) d'être choisi dans un disque de centre y et de rayon f*r, où r=d(x,y) est minorée par $\frac{\pi * f^2}{K\left(1+f^2\right)}$ (5)

Cette propriété sera utilisée pour prouver l'efficacité de l'algorithme de routage utilisé dans Voronet.

3.3.2. Complexité de Routage dans Voronet

Cet algorithme de routage est la structure de base de toutes les algorithmes de routage utilisés dans Voronet .Que ce soit un routage pour acheminer un message vers un objet ou pour

trouver une nouvelle région où un objet sera ajouté, ou pour trouver un objet responsable d'un nouveau lien vers un voisin éloigné .Dans cette fonction, la cible est située au point cible sur le graphe de Voronoï. La fonction DistanceToRegion (x) calcule la distance entre un point x et la région de Voronet de l'objet courant (noté CurrentObject dans l'algorithme) .Si cette distance mène à un point z alors d(x, CurrentObject) = d(x, z) avec z**Erreur ! Signet non défini.** appartenant à la région de l'objet courant (R (CurrentObject)), et DistanceToRegion donne en sortie le point z.

Si x appartient à R(x) alors DistanceToRegion donne x.

Enfin GREEDYNEIGHBOUR(x) est le voisin le plus près (en utilisant la norme euclidienne) de x parmi les voisins de l'objet courant :{VN(x), CN(x), LRN(x) …}

Tableau 3 Algorithme2 : algorithme pour le routage commençant au point x

ROUTE(*x, Target*)
z = DistanceToRegion (*Target*)
if *d*(*z,Target*) > 1/3 *d*(*Target, CurrentObject*) and
d(*Target, CurrentObject*) > *d*min then
Continuer le processus ROUTE(*x, Target*) sur l'objet retourné
Par GREEDYNEIGHBOUR (*Target*)
Else
Ajouter un objet fictif sur le réseau au point z
Ajouter un objet sur le réseau au point *Target*
Réaliser quelques opérations selon l'opération
Qui devra être réalisée au point z
Détruire l'objet fictif créé en *z*
(Selon l'action, enlever l'objet au point *Target*)
Return

Cet algorithme est correct si nous montrons que lorsqu'il se termine nous pouvons ajouter z au réseau puisqu'il est assez proche de l'objet courant et ensuite que la cible est assez proche de z et peut être ajouté au réseau .Ceci est résumé dans le lemme ci-dessous

Lemme 4

Supposons que

$d(\text{DistanceToRegion}(Target), Target) \leq d(Target, CurrentObject)/3$

ou

$d(Target, CurrentObject) \leq d\min$

Alors z et Target peuvent être ajoutés successivement à l'overlay c'est-à-dire au réseau

Le lemme 5 qui suit démontre que le coût du routage est poly logarithmique en la taille du réseau et nous démontrer cela

Lemme 5

Le nombre d'appels à la fonction GREEDYNEIGHBOUR dans l'algorithme 2est de l'ordre de O (ln2 |*O*|Target).

Preuve

Considérons une étape dans l'algorithme 2 :

Soit d=d(CurrentObject,Target).

D'après le lemme 3 la probabilité de choisir LRT (CurrentObject) dans un disque de centre Target et de rayon **Erreur ! Signet non défini.**d/6 est bornée inférieurement par

$$\frac{1}{98\ln\left(\frac{\pi|o|T\arg et}{\sqrt{2}}\right)}$$

Soit X le nombre d'appels à la fonction GREEDYNEIGHBOUR avant d'atteindre un objet s tel que LRT(s) appartienne au disque de centre Target et de rayon $\frac{d}{6}$

L'espérance de x est donnée par

$$E(X) = \sum_{i=1}^{+\infty} \Pr[X \geq i] \qquad (6)$$

$$E(X) \leq \sum_{i=1}^{+\infty} \left(1 - \frac{1}{98\ln\left(\frac{\pi|O|t\arg et}{\sqrt{2}}\right)}\right)^{i-1} \qquad (7)$$

Or $\sum q^j = \frac{1}{1-q} si|q| \leq 1$

En posant q= $\left(1 - \frac{1}{98 \ln\left(\frac{\pi |O| t \arg et}{\sqrt{2}}\right)}\right)$ on obtient donc

$$E(X) \leq 98 \ln\left(\frac{\pi |O| t \arg et}{\sqrt{2}}\right) \quad (8)$$

Supposons maintenant que nous avons atteint un obj et courant qui est tel que LRT (CurrentObject) appartienne au disque de centre Target et de rayon $\frac{d}{6}$, alors nous pouvons démontrer voir [6] que :

- Soit GREEDYNEIGHBOUR (CurrentObject) satisfait

d (GREEDYNEIGHBOUR(*CurrentObject*),*Target*)

$\leq \frac{5}{6}$ *d* (*CurrentObject , Target*)

Ou bien

- la condition suivante est remplie

d(*z* = DISTANCETOREGION(*Target*),*Target*)

$\leq \frac{1}{3}$ *d* (*Target, CurrentObject*).

Donc, en continuant le routage après $98 \ln\left(\frac{\pi |O| t \arg et}{\sqrt{2}}\right)$ appels de la fonction GREEDYNEIGHBOUR, l'algorithme s'arrête parce que la condition

d (*z*=DISTANCETOREGION(*Target*),*Target*)

$\leq \frac{1}{3}$ *d* (*Target, CurrentObject*) est remplie ou bien la distance entre l'objet courant (CurrentObject) et la cible Target est divisée par $\frac{6}{5}$

Considérons qu'une telle séquence d'appels de la fonction GREEDYNEIGHBOUR est une super étape H.

Comme après chaque super étape H, soit l'algorithme s'arrête ou bien la distance entre l'objet courant et la cible Target est divisée par $\frac{6}{5}$, le nombre de supers étapes est borné supérieurement par

$$\frac{\ln\left(\frac{\pi|O|t\arg et}{\sqrt{2}}\right)}{\ln\left(\frac{6}{5}\right)}$$

La distance initiale est plus petite que $\frac{1}{\sqrt{2}}$ et l'algorithme s'arrête des que la distance est plus petite que d_{min}

Et par linéarité de l'espérance, l'espérance du nombre d'étapes N de l'algorithme 2 est donnée par

$$E(N) \leq \frac{\ln\left(\frac{\pi|O|t\arg et}{\sqrt{2}}\right)}{\ln\left(\frac{6}{5}\right)} \times 98\ln\left(\frac{\pi|O|t\arg et}{\sqrt{2}}\right) \quad (9)$$

$$E(N) \leq \alpha \ln 2\left(|O|_{t\arg et}\right) \quad (10)$$

Ainsi donc le coût du routage est poly logarithmique en la taille du réseau

3.4. Les transactions web 2.0 et les systèmes pair-à-pair basés sur les diagrammes de Voronoi

3.4.1. L'efficacité du routage des transactions web2.0 avec les diagrammes de Voronoi

Avec ces réseaux, c'est le routage glouton qui y est utilisé pour la transmission des messages. Le routage glouton est un processus de routage où chaque nœud transmet le message à son voisin le plus proche de la zone de destination. Le routage dans de tels réseaux reste efficace si le nombre de nœuds logiques se trouvant dans les zones découpées est borné c'est-à-dire ne

tend pas vers l'infini. Ce qui est le cas avec les applications web 2.0 où le nombre d'objets par utilisateurs c'est-à-dire le nombre de nœuds logiques par utilisateurs est borné [17]. Donc le routage reste efficace si nous utilisons ces réseaux.[2]

3.4.2. La possibilité des requêtes par plages de valeurs dans ces réseaux

En plus de ce routage efficace, Ces réseaux basés sur les diagrammes de Voronoi possèdent d'autres critères qui font d'eux une architecture idéale pour le traitement des transactions web 2.0.

En effet les accès web 2.0 ou transactions web 2.0 sont en général regroupés dans de petites portions de l'espace. Ce qui augmente la probabilité que les objets, sur lesquels porte une transaction web 2.0, partagent un attribut en commun. Par exemple un objet o et un objet v peuvent être tel que $o_x = v_x$ et dans ce cas, ces réseaux permettent d'avoir des requêtes par plage de valeurs.

Une requête par plage de valeurs implique de contacter l'ensemble des objets dans une zone rectangulaire définie par les plages de valeurs recherchées.

C'est-à-dire quand une requête porte sur plusieurs objets se trouvant dans une même zone, il est possible avec ces réseaux d'utiliser un mécanisme de division de l'espace à l'initiative du premier objet concerné rencontré lors du r outage .L'espace de la requête est divisé en plusieurs sous-espaces et une sous requête est adressée à chacune des sous-espaces.

3.4.3. Modification de la topologie du réseau en fonction des accès

Avec les réseaux basés sur les diagrammes de Voronoi, on pe ut avoir plusieurs types de nœuds comme les nœuds de données (nœuds contenant les données des avatars), des nœuds routeurs (nœuds traitant les accès des avatars) et des nœuds avatars représentant les utilisateurs connectés.

Il nous est possible de changer la position d'un nœud de données en fonction des avatars qui lancent des accès vers ce nœud.

2 les réseaux basés sur les diagrammes de Voronoi

Nous voulons dire par là qu'il est possible de modifier de façon dynamique la structure logique du réseau en fonction des accès des avatars. Et pour cela il suffit simplement de changer la position de certains nœuds de données.
Par exemple, si les avatars qui accèdent le plus aux données de la zone Z_k appartiennent à la zone Z_j, il est possible de rapprocher les zones Z_k et Z_j. Ce qui réduit le coût du routage.

Conclusion

En résumé, Voronet est un réseau objet-à-objet, dans lequel chaque objet est décrit par un ensemble d'attributs. Les objets sont placés dans un espace euclidien. Ils sont liés de manière à ce que les objets avec des attributs proches soient eux-mêmes proches dans le réseau, en conservant un nombre de voisins par objet d'ordre constant. De plus, des mécanismes de routage efficace « petit-mode » généralisent le modèle de Klein Berg sur la grille à Voronet. Voronet est destiné à être le support de mécanisme de recherches complexes, comme des requêtes par plage de valeurs. L'approche suivie ouvre des perspectives intéressantes pour une mise en forme effective des principes petit-monde et pour les réseaux objet-à-objet.
Dans la suite nous allons modifier l'architecture et le fonctionnement de Voronet pour y implémenter notre algorithme pour découper les données et le protocole de routage que nous avons proposé.

Chapitre4

GESTION DECENTRALISEE DES TRANSACTIONS WEB 2.0

Introduction

Dans les deux chapitres précédents, nous avons étudié les accès des utilisateurs de Second Life une application web 2.0. Cette étude nous a permis de caractériser les accès aux données, de pouvoir regrouper les accès par zone en nous basant sur leur position sur un espace cartésien. Ce découpage en zone nous a mené aux systèmes de bases de données pair à pair basés sur les diagrammes de Voronoi à savoir Voronet. Voronet comme nous l'avons déjà vu offre de nombreux avantages :

Le coût du routage avec VORONET est poly logarithmique en la taille du réseau, on peut faire des requêtes par plages de valeurs et l'exactitude est garantie…..

Cependant VORONET ne permet pas d'éviter de centraliser le contrôle des accès aux données. Nous allons modifier donc le fonctionnement de VORONET pour pallier aux insuffisances de la centralisation du contrôle des accès.

4.1. L'architecture pair-à-pair proposé pour l'application *Second Life,*

4.1.1. Les composantes de l'architecture

Les réseaux pair-à-pair basés sur les diagrammes de Voronoi sont constitués d'ensemble de cellules adjacentes. Chaque cellule gérée par un nœud et contenant d'autres nœuds. Les nœuds sont affectés aux cellules en fonction de leur position et on peut décider des relations d'adjacences entre les nœuds.

Notre architecture est donc un diagramme de Voronoi où, les cellules correspondent aux zones que nous avons découpées lors du découpage de l'espace logique de Second Life en zones dans le chapitre 2. Chaque zone est gérée par un noeud routeur (TMN). Chaque zone

contient des nœuds de données (DN) et des nœuds avatars (AN) et chaque zone a deux zones répliques. Cette réplication sera utilisée pour équilibrer la charge lors du routage des accès. Les données sont affectées aux nœuds DN en fonction de leur position.
Un SGBD gère les données de plusieurs zones et est stocké sur les nœuds physiques des zones. Chaque SGBD a un dictionnaire de données local qui est verrouillé lors de la délimitation des données.

4.1.2. Description détaillée des différents nœuds de l'architecture

4.1.2. A. Les nœuds avatars (AN : Avatar Node)

Ces nœuds représentent les avatars .Leur position dans l'overlay est stockée sur la mémoire vive du nœud physique contenant le nœud TMN et ces TMN s'échangent des informations pour effectuer l'opération de (Handover) qui est réalisée quand un avatar change de zone.

4.1.2. B. Les nœuds de données (DN : Data_node)

Ces nœuds contiennent les objets des avatars et leur position est en générale fixe sur l'overlay .Ils peuvent être déplacés par les avatars .Chaque nœud avatar contient une liste de fichiers descripteurs, chaque fichier portant sur un nœud objet. Les données de ces nœuds objets sont stockées sur les bases de données des zones.

4.1.2. C. Les nœuds gestionnaires de transaction (TMN : transaction management Node)

Dans notre architecture, ceux sont ces nœuds qui sont structurés en un diagramme de Voronoï. Chacun de ces nœuds est un gestionnaire de transactions .Chaque gestionnaire de transactions se trouve dans une zone et est entouré des quatre zones de ses voisins directes qui sont aussi des gestionnaire de transactions. Et chaque nœud gestionnaire de transaction possède des fichiers descripteurs portant sur les nœuds DN appartenant à sa cellule ainsi que sur les avatars appartenant à sa cellule. Chaque gestionnaire de transaction a trois liens longs vers des voisins éloignés .Parmi ces liens longs, les deux pointent vers des zones répliques de la zone du gestionnaire de transactions. C'est à dire contenant les répliques des nœuds DN de la zone du gestionnaire de transactions.

Chacun des TMN (gestionnaire de transactions) a un médiateur (couche logicielle intermédiaire) architecturé ainsi et lui permettant d'appliquer notre protocole pour décentraliser le contrôle des accès aux données.

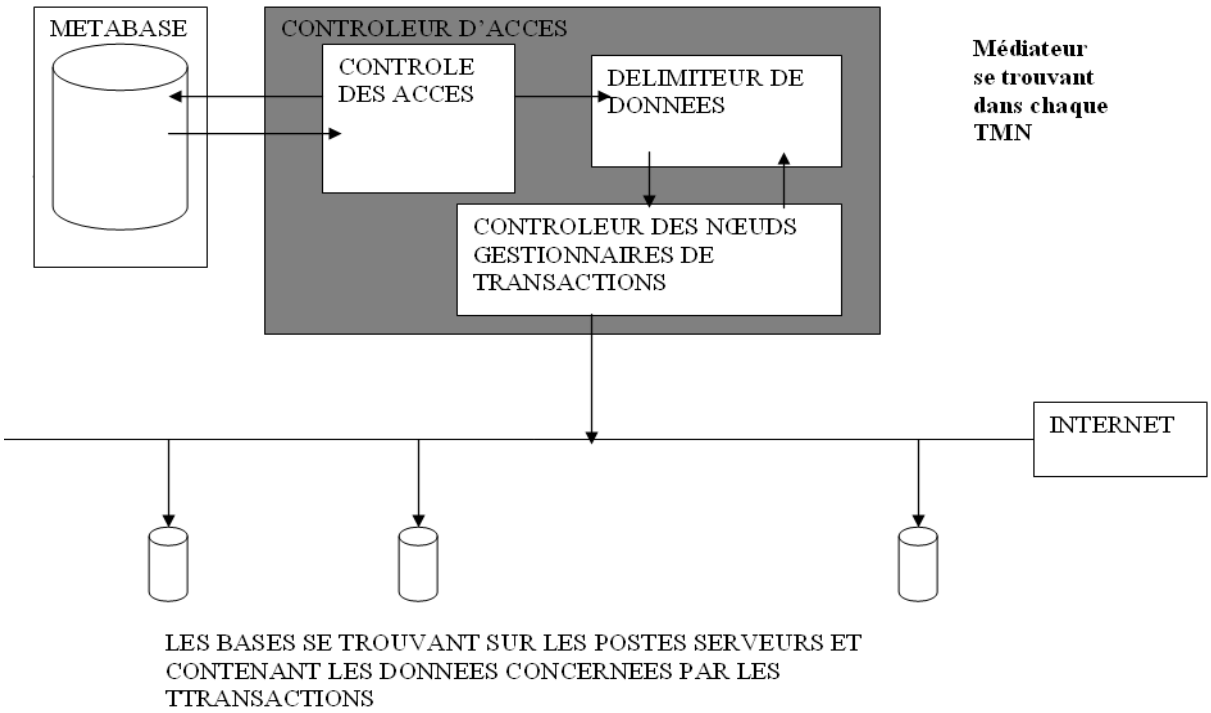


Figure 8 : le contrôleur des accès

Description du schéma : de la figure 8

Le module de contrôle des accès calcule la localité et la fréquence des accès en déterminant la valeur de la métrique Loc-acc que nous avons déjà défini en première partie de notre travail;

Le module délimiteur de données permet de déterminer si le nœud gestionnaire de transaction est un fournisseur (c'est-à-dire va transmettre la requête à l'un de ses nœuds de données) de données ou un nœud simple (c'est-à-dire va transmettre la requête à l'un de ses routeurs répliques) et donc permet de situer la localité des données.

Le module contrôleur de nœuds gestionnaires de transactions permet d'envoyer un message aux nœuds gestionnaires se trouvant dans les cellules répliques pour déterminer le nouveau nœud fournisseur de données vers lequel les transactions seront routées.

5. Algorithme pour créer des zones de concurrence en fonction des accès.

Chaque routeur garde les informations sur les accès qu'il reçoit à savoir (le type d'accès, les objets concernés (nœuds de données concernés), position objets ou position des nœuds de données, avatars ayant lancé la transaction, date transaction, avatars possédant les objets ...).
Nous avons un index [zone, routeur] stocké au niveau de la metabase des nœuds routeurs et qui permet de savoir le routeur qui gère chaque zone.
Au bout de chaque durée d, le dictionnaire de données est verrouillé pour permettre la redéfinition des données. Chaque routeur calcule la métrique Loc-acc de sa zone.
Une zone est saturée si le nombre d'accès lancés sur les objets qu'elle contienne est très grand (Loc-acc>>1).

- Si dans une zone il y'a une petite portion de nœuds de données qui est fortement accédée c'est-à-dire qui est à l'origine de la saturation de la zone, alors cette zone peut être compartimentée en deux zones.
    - Pour la nouvelle zone contenant la portion de nœuds de données fortement accédées issue du découpage, un dictionnaire de données est généré et va contenir les informations sur les données de cette zone. Un routeur (nœud gestionnaire de transactions) est ajouté pour cette nouvelle zone.
    - L'autre nouvelle zone contenant des données non fortement accédées reste sous le contrôle du routeur qui gérait la zone avant le découpage. Et les données qu'elle contient restent sous le contrôle du SGBD d'avant le test.

A chaque fois qu'une zone contient des nœuds de données fortement accédées et des nœuds de données délaissées on peut essayer de le diviser en deux zones et affecter un nouveau routeur à la nouvelle zone contenant les nœuds de données fortement accédées. La nouvelle zone contenant la portion de données fortement accédées sera une zone de concurrence.

- Si tous les nœuds de données de la zone sont fortement accédés alors on ne compartimente pas les données de la zone, on considère la zone comme une zone de concurrence et un serveur va gérer les données de cette zone. Le dictionnaire de données est modifié.
- Si une zone de concurrence (ayant tous ses nœuds de données fortement accédés) devient délaissée alors on peut le remettre sous le contrôle du SGBD qui le gérait avant qu'elle ne devienne une zone de concurrence. Pour cela il suffit de consulter

l'index [zone, routeur] pour remettre la zone sous le contrôle du routeur d'origine qui le gérait avant l'élection de la zone en zone de concurrence.

En procédant ainsi nous pouvons créer à chaque fois des zones de concurrence en fonction des accès des utilisateurs en compartimentant les données et en leur allouant un nouveau routeur.

Nous avons ci-dessous le schéma d'un SGBD qui gère k zones avant que le test de découpage dynamique ne se fasse et le schéma montrant la création d'une nouvelle zone de concurrence par découpage des données d'une même zone.

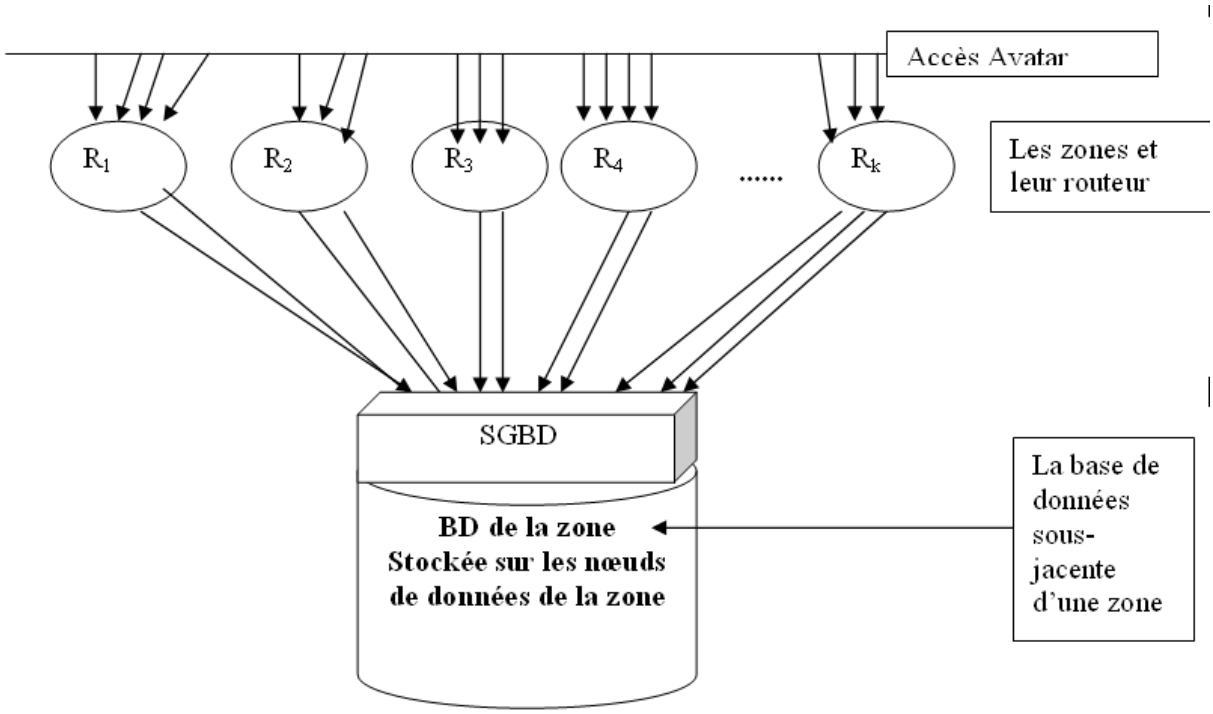


Figure 9-a : schéma d'un SGBD gérant plusieurs zones

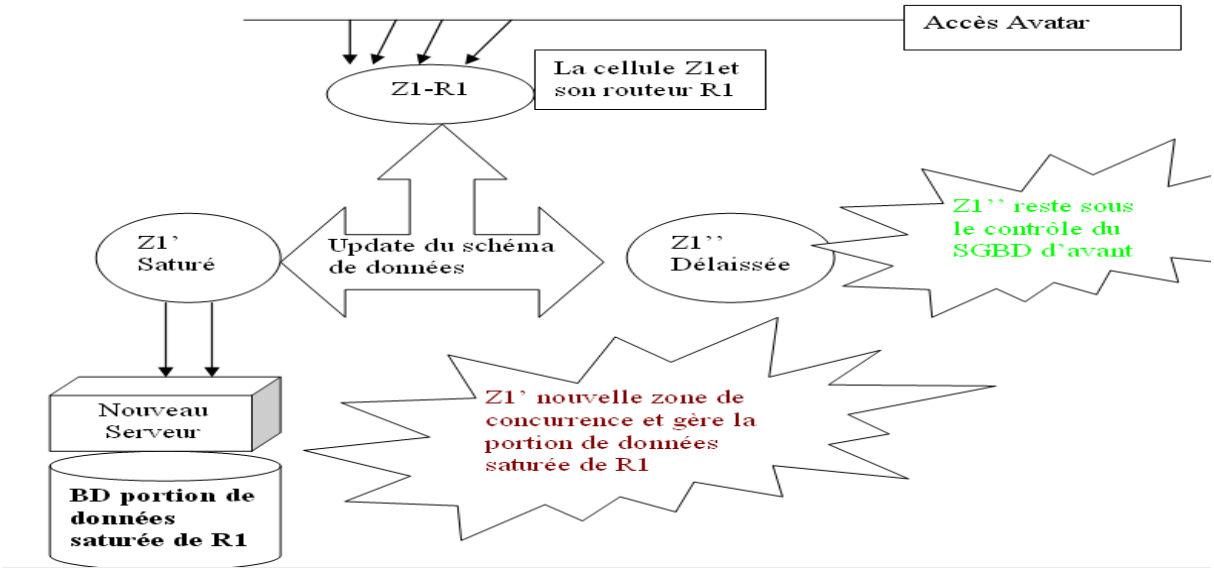


Figure 9-b) : Création d'une nouvelle zone de concurrence par découpage d'une cellule (zone)

6. Le protocole de routage des transactions lancées par les utilisateurs

Nos nœuds gestionnaires de transactions fonctionnent ainsi :

Quand ils reçoivent des transactions ,ils gardent les métadonnées sur ces transactions(type transaction ,objets concernés ,position objets ,avatars ayant lancé la transaction ,date transaction ,avatars possédant les objets ...) dans leur métabase . N'oublions pas que pour exécuter un accès sur un objet, l'avatar doit être dans la même zone que l'objet. Les gestionnaires de transactions s'échangent leur liste de fichier descripteur pour connaître les nœuds objets voisins. Notre protocole fonctionne ainsi :

Les avatars lancent des accès vers des nœuds objets .Ces accès aux données dans une zone sont routés vers le gestionnaire de transaction de la zone. Le gestionnaire de transactions peut être dans l'état fournisseur c'est-à-dire traite les accès ou dans l'état simple où il transmet les accès à l'une de ses répliques.

- Si ce dernier est un fournisseur alors il route les accès vers ses nœuds objets adéquats dans sa zone.
- Sinon, si ce gestionnaire de transaction n'est pas un fournisseur, alors il route les accès vers le gestionnaire de transaction de sa zone réplique qui est le moins chargé. Pour déterminer si un nœud gestionnaire de transaction est un fournisseur, ce dernier

calcule la métrique Loc-acc. Cette métrique permet de quantifier la localité et la fréquence des accès aux objets d'une zone.

- Si la Loc-acc d'une zone est supérieure au nombre N de pairs se trouvant sur la zone, alors la zone est une zone saturée et le gestionnaire de transaction de la zone saturée devient un pair simple. La réplication des zones est asynchrone et chaque zone élue (routeur de la zone est fournisseur) doit recevoir les transactions qu'il lui faut pour être capable de traiter les prochaines transactions. Si un nœud fournisseur lors du test change d'état pour devenir un nœud simple, alors il choisit l'une de ses répliques la plus à jour pour lui envoyer le nombre de mises à jour qu'il lui faut pour pouvoir traiter les prochaines transactions à venir. Cette réplique devient fournisseur à son tour.

Chaque nœud gestionnaire de transaction peut soit tirer un délai aléatoire pour déterminer s'il continue d'être un fournisseur ou au bout de chaque T secondes tous les nœuds gestionnaires de transactions peuvent déterminer s'ils continuent d'être des fournisseurs.

Cependant nous optons pour la première solution tout en fixant une contrainte sur la durée de ce délai aléatoire .En effet ce délai ne doit pas être trop grand

En cas de panne d'un nœud gestionnaire de transactions un autre nœud parmi ses deux répliques, qui est le moins chargé, est désigné comme nouveau fournisseur. Et va traiter les Transactions vers les données de ce nœud. Ils existent plusieurs mécanismes pour déterminer le nouveau nœud [cf. Réplication des données de Pascal Moli]

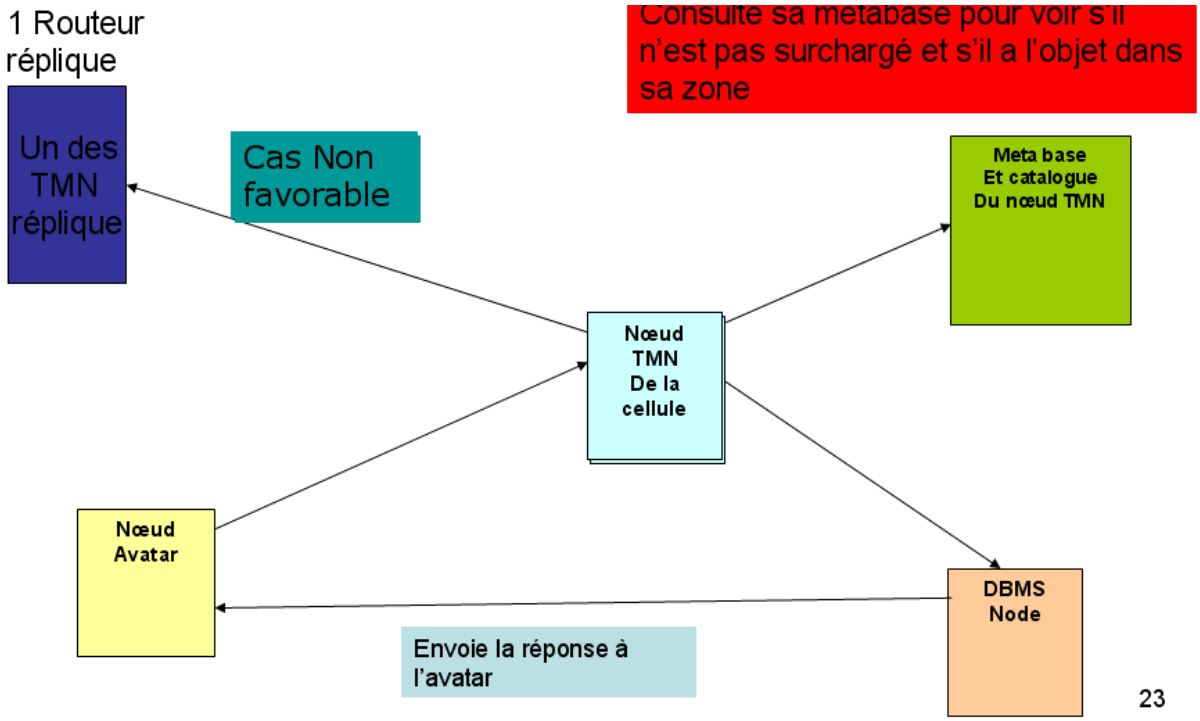


Figure 10 : le protocole de routage des accès

Dans cette section, nous avons proposé une architecture pair-à-pair basée sur l'overlay (réseau logique) Voronet pour décentraliser l'application Second Life.

Nous avons proposé un a lgorithme pour compartimenter (c'est-à-dire délimiter) dynamiquement les données en fonction des accès réalisés par les utilisateurs de Second Life.

Nous avons aussi fourni un protocole pour éviter la centralisation du contrôle des accès aux données du réseau.

Cependant pour être validé, tout protocole doit être testé dans l'environnement adéquat pour lequel il a été fourni. Pour tester notre algorithme, nous avons réalisé différentes expériences en utilisant le simulateur PeerSim .Dans la section suivante nous montrons pourquoi notre choix a porté sur ce simulateur

Chapitre 5

Validation de Notre Algorithme

1 PeerSim

C'est un logiciel de simulation d'environnements pair-à-pair. Il est *open source* et son code écrit en Java lui assure une grande modularité mais aussi une forte extensibilité. Peersim permet de simuler le mode de fonctionnement d'un réseau à grande échelle avec des milliers de noeuds, voir même des centaines de milliers à quelques millions.

L'utilisation de PeerSim consiste essentiellement à définir le comportement des noeuds du réseau. En effet, il offre un ensemble d'éléments, classes et interfaces, assurant la génération du réseau. On peut en citer selon leur utilité :

- Les noeuds (*Node*) qui forment le réseau. Ce sont les pairs. Chacun d'eux peut exécuter un ensemble de protocoles et peut aussi joindre ces voisins grâce à l'interface *Linkable*.
- Le réseau (*Network*) représente l'ensemble de tous les pairs. C'est une vision globale de l'environnement P2P.
- Les planificateurs (*Scheduler*) comme leur nom l'indique, ils planifient le fonctionnement du réseau : l'état d'initialisation, l'ordre d'exécutions des différents composants…

Parmi les principaux composants de PeerSim, on note :

- Les protocoles (*Protocol*) qui sont exécutés par les noeuds. Ils constituent le noyau du simulateur du fait qu'ils déterminent son comportement. Un noeud peut exécuter plusieurs protocoles. Un protocole n'utilise que les informations locales du noeud qui l'exécute.
- Les *Observer*s permettent grâce à leur vision globale du système d'afficher directement à l'écran les résultats ou bien de les rediriger vers un fichier pour d'autres utilités.
- Les *Dynamic*s assurent la dynamicité du système en créant des arrivés et départs de noeuds. Ils peuvent être utilisés au début de la simulation pour initialiser le système.

1.1 Les modes de simulation avec PeerSim

PeerSim supporte deux types de simulation à savoir :
La simulation par cycle : La simulation s'effectue dans un ordre séquentiel et dans chaque cycle, chaque protocole peut exécuter son comportement.
La simulation par évènement : elle supporte la concurrence; un ensemble d'évènements (messages) est planifié et les protocoles d'un noeud sont exécutés en fonction des évènements survenus.

Un fichier texte de configuration permet de paramétrer la simulation en fonction des besoins. Ce fichier permet de définir l'évolution du réseau, sa taille, la durée de la simulation, le nombre d'expériences voulues…

Sa modularité et sa licence libre font de PeerSim un simulateur privilégié pouvant être adapté à plusieurs contextes. Le code source du simulateur est téléchargeable sur [30]].

2 Notre Implémentation

2.1) Scénario principale

Nous avons un réseau pair à pair formé de plusieurs groupes de nœuds de données, chaque groupe a ses accès gérés par un nœud gestionnaire de transaction ou nœud routeur. Chaque nœud de données a une base de données. Et la base de données de chaque groupe est constituée des bases de ses nœuds de données.
Un ensemble de C clients envoie en début de simulation un nombre X de messages à chaque routeur. Les routeurs transmettent ces messages à leurs nœuds de données. En effet chaque nœud contient un numéro aléatoire compris. Si le numéro du message est égal au numéro du nœud de données alors le routeur transmet à ce dernier le message. Les nœuds de données traitent les messages et notifient le client ayant lancé le message. Si le nombre de messages adressés à un nœud de données atteint une limite fixée, ou si le nombre de messages adressés à un ensemble de nœuds de données d'un même groupe atteint une limite, nous découpons le groupe en deux groupes. L'un des groupes aura les nœuds de données les moins accédés et

l'autre groupe le ou les nœuds de données les plus accédées. Chaque routeur a deux routeurs répliques qui lui permettent de contrôler sa charge pour qu'il ne se retrouve pas saturé.

2.2) Détails techniques

Nous avons quatre classes à savoir :

- la classe SearchProtocol
- la classe SearchDataInitializer
- la classe ParamObserver
- la classe CompartProtocol

- La classe SearchDataInitializer permet d'initialiser les paramètres de simulations comme le nombre de clients, de routeurs, de nœuds de données.
- La classe ParamObserver permet de récupérer les résultats de la simulation comme l'évolution de la charge des nœuds de données, des nœuds routeurs. En observant l'évolution de la charge des nœuds, nous pourrons obtenir le temps de réponse, le nombre de zones de concurrence créées …
- La classe CompartProtocol étend la classe SearchProtocol et nous y a vons codé plusieurs méthodes permettant de simuler notre algorithme pour compartimenter les zones quand la charge soumise sur une zone devient très importante.

Voici les méthodes les plus importantes parmi ces dernières.

- ✓ La méthode fragmenter_zone () : Elle permet de fragmenter un groupe de nœuds représentant une zone en affectant les nœuds les plus accédés à u n nouveau groupe (nouveau routeur) pour les isoler afin d'éviter la saturation des autres nœuds du groupe par des messages qui ne leurs sont pas destinés.
- ✓ La méthode doTransfer () : Elle permet d'équilibrer la charge entre les nœuds routeurs et leurs répliques.
- ✓ La methode tester_load () permet de vérifier l'état d'un nœud du réseau pour voir s'il est surchargé ou pas.
- ✓ La methode send_message_routeur () permet aux nœuds clients d'envoyer des messages aux routeurs.
- ✓ La methode forward (SMessage mes) permet aux routeurs de transmettre les messages à leurs data_node.

3 Expériences

3-1) Expérience 1 : Le taux de routeurs ayant un surplus de requêtes

Nous commençons par montrer dans cette première expérience que notre algorithme parvient à éviter l'accroissement du nombre de goulots d'étranglements dans le réseau.

Nous avons une base de données pair-à-pair constituée de plusieurs nœuds de données regroupés dans des zones et les accès sont gérés au niveau de ces zones par des nœuds routeurs. N'oublions pas que chaque routeur a deux nœuds routeurs répliques dans notre architecture. Le fait de pouvoir équilibrer la charge entre les routeurs permet d'éviter qu'un nœud routeur ne devienne un goulot d'étranglement. Nous fixons le nombre de clients à 40. A chaque cycle, chaque client envoie 100 messages à tous les routeurs et ces derniers vont transmettre ces messages à leurs nœuds de données en suivant notre protocole de routage des accès. Quand la charge d'un nœud routeur atteint une valeur très supérieure à la valeur maximale de la métrique loc-acc de la zone du routeur, nous considérons que le routeur est saturé.

Nous étudions l'évolution de la fonction donnant le rapport du nombre de routeurs ayant un surplus de requêtes sur le nombre total de routeurs en faisant croitre la valeur de métrique Loc-acc de 30 à 90.

La figure ci-dessous nous donne cette variation :

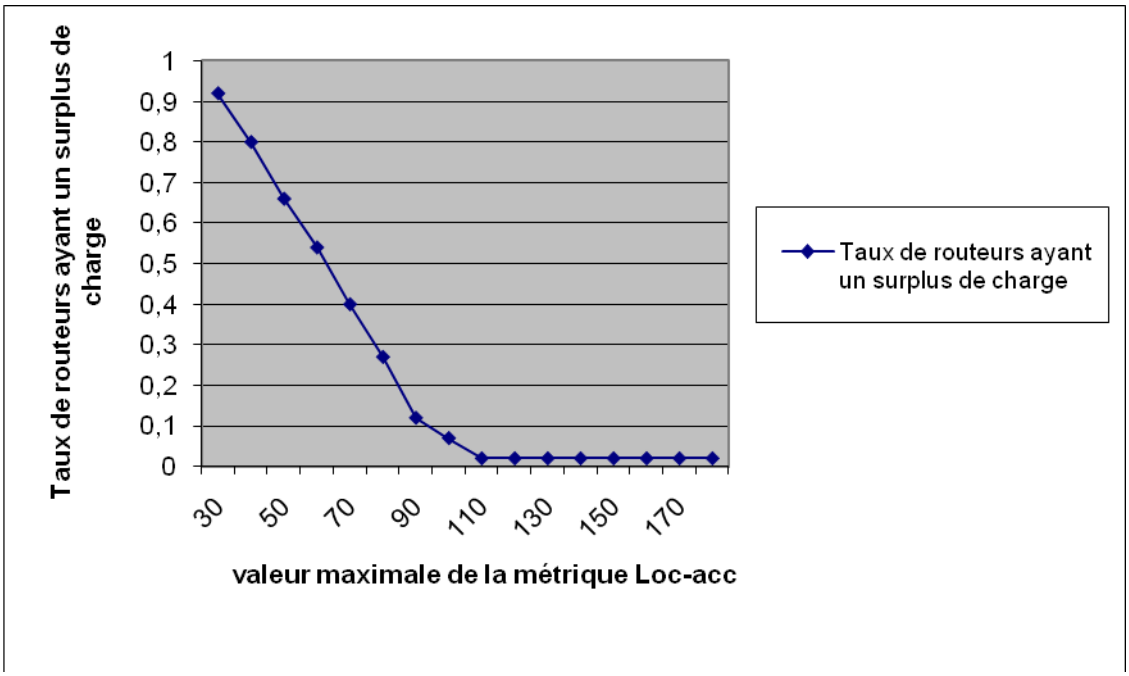


Figure 11 : le taux de routeurs ayant un surplus de requêtes

Chaque routeur qui voit le nombre de messages de son buffer atteindre la valeur maximale de Loc-acc fixée applique notre protocole d'équilibrage de charge en désignant un de ses répliques comme nouveau fournisseur.

Quand la charge maximale à partir de laquelle les routeurs sont considérés comme saturés est très petite, le pourcentage de routeurs saturés devient très grand. En effet le nombre de test d'équilibrage devient élevé et au bout d'un nombre très petit de cycles tous les routeurs fournisseurs ou simples deviennent vite saturés.

Par contre quand la valeur maximale de Loc-acc devient importante, les tests sont espacés dans le temps et les routeurs simples ont le temps de traiter les requêtes qu'ils avaient dans leur buffer avant de devenir à nouveau fournisseurs.

Notre système pair à pair tend vers un état ou le nombre de routeurs ayant un surplus de charge devient négligeable et on voit le rapport du nombre de routeurs saturés sur le nombre total de routeurs tendre vers une valeur négligeable.

Nous pouvons tirer comme conclusion de cette première expérience que notre algorithme permet en choisissant une valeur adéquate de la métrique Loc-acc de diminuer considérablement le nombre de routeurs ayant un surplus de charge. Cependant lorsque la valeur maximale de la métrique Loc-acc à partir de laquelle un routeur est considéré comme

saturé est trop grande on risque d'avoir un temps de réponse considérable du à la durée d'attente des requêtes dans les buffers des routeurs. Nous allons dans la suite voire l'évolution de ce temps de réponse en fonction de la charge injectée par les clients.

3-2) Expérience 2 : Le Temps de réponse obtenu avec notre protocole

Le but de notre protocole pour compartimenter les données était d'éviter le contrôle centralisé des accès aux données qui devient un goul ot d'étranglement lorsque la charge devient exponentielle. Nous allons montrer que notre protocole permet d'éviter les goulots d'étranglement. Et pour cela nous étudions les variations de l'une des critères de performance de notre système pair-à-pair qu'est le temps de réponse. Dans PeerSim en mode par cycle, le temps s'exprime en termes de nombre de cycles.
Et pour déterminer le temps qu'il faut pour traiter les accès, nous devrons donc évaluer le nombre de cycles qu'il faut pour que tous les accès soient traités.
Pour déterminer si ce temps de réponse est acceptable, nous le comparerons au temps de réponse obtenu dans un graphe petit monde où nous n'avons pas appliqué notre protocole pour compartimenter les données. Nous allons d'abord expliquer le scénario ensuite nous montrons l'évolution du temps de réponse en fonction de la charge puis nous montrons le temps de réponse obtenu co mparé au temps de réponse dans un gr aphe petit monde sans notre protocole.

- Scénario

Nous avons un réseau formé de plusieurs groupes de nœuds pairs. Dans chaque groupe, un routeur reçoit les requêtes et les transmet à ces nœuds de données de destination. Un client envoie un nombre X de messages dans le réseau et un message n'est donné à un routeur que s'il contient le data_node de destination. Les numéros des messages sont aléatoires et chaque nœud de données a un numéro aléatoire. Si le numéro du data_node est le numéro du message alors le nœud de données est le nœud de destination du message.
Pour obtenir le temps de réponse, nous étudions l'évolution de la charge maximale dans les SGBD ou plus précisément dans les nœuds de données. Quand cette charge maximale tombe à zéro, nous considérons que tous les SGBD ont traités leurs requêtes.

- Comparaison du temps obtenu avec le temps obtenu dans un graphe sans notre protocole

Dans cette partie nous faisons varier le nombre de messages lancés par les clients dans le système pair-à-pair nous comparons le temps de réponse obtenu avec le temps de réponse obtenu dans un graphe petit monde où notre protocole n'est pas appliqué.
La figure suivante nous montre l'évolution du temps de réponse obtenu avec notre protocole en fonction du nombre de requêtes lancées dans le réseau en début de simulation.

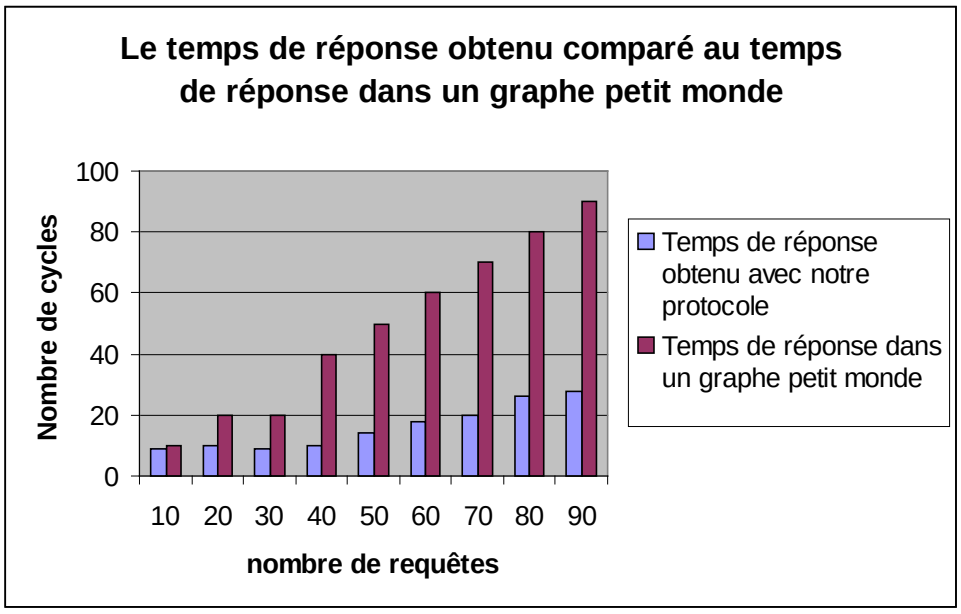


Figure 12 : le temps de réponse obtenu

En move nous avons le temps de réponse obtenu avec notre protocole qui est linéaire et en rouge le temps obtenu sans notre protocole qui croit de manière exponentielle.
Le fait de compartimenter les données et d'élire de nouveaux routeurs gérant chacun une zone permet d'éviter que des requêtes portant sur des nœuds de données non saturées se retrouvent sur le même routeur que des requêtes portant sur des nœuds de données saturés. Et cela permet de diminuer le temps de traitement des requêtes

En compartimentant les données de manière dynamique notre protocole fournit un temps de réponse qui est presque égale au tiers du temps de réponse d'un graphe petit monde.

Cependant la fragmentation des données induit une augmentation du temps de traitement des données du fait qu'il faille verrouiller les métadonnées lors de la redéfinition du schéma des données. Le nombre de fragmentations varie en fonction de la charge injectée et peut avoir un effet dégradant sur le temps de réponse et dans l'expérience qui suit nous allons soumettre notre système à une forte charge pour voir l'évolution de ce temps de réponse

3-3) Expérience 3 : L'évolution du temps de réponse quand la charge croit

Dans cette expérience le scénario est le même que dans la précédente. Nous avons un système pair-à-pair où des clients envoient des messages. Les messages sont transmis par les nœuds routeurs à leurs nœuds de destination. Quand un nœud de données commence à saturer un groupe, la fragmentation est appliquée.

Nous faisons varier le nombre de message et nous mesurons à chaque fois le nombre de cycles pour que tous les nœuds soient traités.

La courbe ci-dessous nous donne l'évolution de ce temps de réponse.

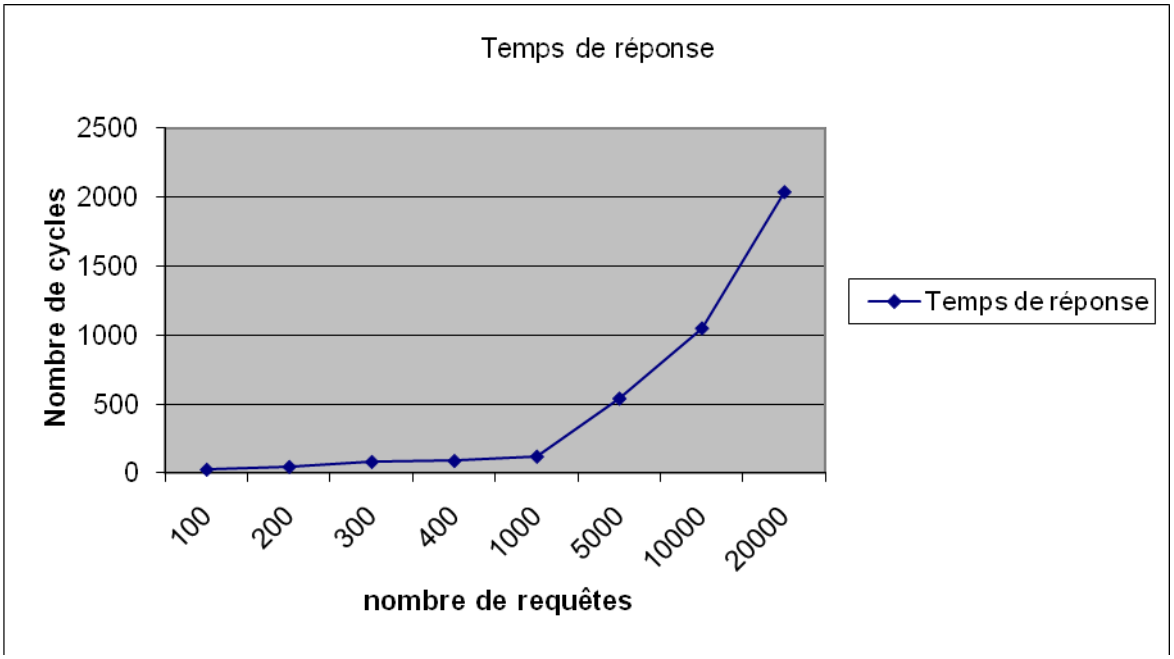


Figure 13 : le passage à l'echelle

En bleu nous avons la courbe du nombre de cycles qu'il faut pour que toutes les requêtes soient traitées qui croit de manière logarithmique au début pour les valeurs de 100 à 400 pour devenir linéaire quand la charge dépasse 1000 requêtes

Notre algorithme évite le contrôle centralisé des données en fragmentant les zones (groupes) de notre système pair-à-pair. Cette fragmentation dépend de la charge contenue dans les nœuds de données des zones. Lorsque la charge injectée dans le système croit, le nombre de fragmentations réalisées devient important et cela peut provoquer une augmentation du nombre de messages de mise à jour des routeurs répliques. Cela alourdit la charge dans la base de données pair-à-pair et donc provoque une augmentation du temps de réponse.
Cependant quand le nombre de fragments devient très petit c'est-à-dire quand le nombre de routeurs devient négligeable, la quantité de données ou le nombre de nœuds de données devient important par routeur ou par zone étant donné que chaque routeur gère une zone.
Dans ce cas le nombre de requêtes que devront gérer chaque routeur devient importante et la encore le temps de réponse peut croitre du au temps d'attente dans les buffers des routeurs.
Donc dans de futurs travaux on pourrait essayer de voire quelle serait le nombre adéquat de nœuds de données à allouer à chaque routeur pour éviter une dégradation du temps de réponse.

4) Conclusion

En résumé, nous avons mis en place un algorithme qui compartimente les zones d'un système de base de données pair-à-pair et crée ainsi des zones de concurrence et des zones délaissées.
Les données sont des données des applications web 2.0 plus précisément de Second life et avec ces applications étiquetées web 2.0, les requêtes d'update sont négligeables comparées aux requêtes de lecture. Et c'est la raison pour laquelle nous n'avons pas cherché à différencier dans nos simulations, les requêtes de lecture et les requêtes d'écriture.
Notre algorithme permet de traiter les requêtes dans un délai acceptable, mais la fragmentation induit un surcoût qui devient non négligeable lors du passage à l'échelle et peut dégrader les performances de notre système dans certaines conditions.

Conclusions et Perspectives

Le but de notre stage de mémoire de DEA dans le domaine des bases de données réparties était de proposer un algorithme pour compartimenter les données dans un système de base de données pair-à-pair afin de pouvoir éviter la centralisation du contrôle des accès. En effet du fait de la croissance exponentielle du nombre des utilisateurs de ces applications communautaires web 2.0, cette centralisation devient un goulot d'étranglement.

Lors de notre étude de ces applications, nous nous sommes limités à une de ces applications communautaires qu'est Second Life et nous avons justifié ce choix par le fait que les applications communautaires web 2.0 présentent des ressemblances au niveau des objets manipulés par leurs utilisateurs. Et aussi par le fait que les jeux de mondes virtuels web 2.0 peuvent incorporer les autres applications web 2.0 telles que les applications de partage de » photos, les blogs …..

Nous avons démontré par simulation que l'algorithme que nous avons proposé permet d'obtenir un temps de réponse acceptable car négligeable comparé aux temps de réponse obtenu dans les graphes petit monde où notre protocole n'est pas appliqué mais induit un surcout sur le temps de traitement des accès web 2.0 du à la fragmentation des données qu'il réalise. Ce qui lors du passage à l'échelle peut poser problème étant donné que ce surcout augmente avec l'augmentation du nombre de zones du fait de l'augmentation de la charge des messages de mise à jour. En diminuant le nombre de zones, la quantité de données ou le nombre de nœuds de données par zone peut augmenter considérablement et provoquer une augmentation du temps de réponse du au temps d'attente dans les buffers des routeurs.

Dans une prochaine étude, nous pourrions étendre notre travail à toutes les applications web 2.0 et essayer de voir le nombre de nœuds de données adéquat par zone pour obtenir un temps de réponse proche de l'optimum en temps de performance.

BIBLIOGRAPHIE

[1] S. Gancarski, H. Naacke, E. Pacitti and P. Valduriez. *The Leganet System: Freshness-Aware Transaction Routing in a Database Cluster*. Information Systems Journal 32(2), pp. 320-343, 2006

[2] G. et O. Gardarin. *Le Client-Serveur*. Editions Eyrolles, 1997

[3] Mathieu Jan. *JuxMem: un service de partage transparent de données pour grilles de calculs fondé sur une approche pair-à-pair*. Thèse de doctorat, Université de Rennes 1, IRISA, Rennes, France, November 2006.

[4] Cécile Le Pape. *Contrôle de Qualité des Données Répliquées dans un Cluster*. Thèse de doctorat, Université Pierre et Marie Curie, LIP6, Paris, France, December 2005.

[5] http://juxmem.gforge.inria.fr/

[6] Samuel Dalens, Vincent Lepage, En Lin. *Routage dans les réseaux Peer-to-Peer.* Article écrit sous la direction de Messieurs Kofman et Rougier

[7] Reza Basseda. *Fragment Allocation in Distributed Database Systems*

[8] M. Etienne Rivière *Collaborative overlay networks for decentralized search in large-scale distributed systems*. ASAP research group (and well, PARIS team for some time too) Advised by Anne-Marie Kermarrec IRISA/Université de Rennes *1*

[9] Solipsis*: A Decentralized Architecture for Virtual Environments.* Davide Frey IRISA Jerôme Royant Orange Labs

[10] Olivier Beaumont, Anne-Marie Kermarrec, Loris Marchal, Étienne Rivière. *VoroNet: A scalable object network based on Voronoi tessellations*

[11] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma and Steven Lim. *A Survey and Comparison of Peer-to-Peer Overlay Network Schemes*

[12] Wikipédia http://fr.wikipedia.org/wiki/pair-à-pair

[13] Second Life http://www.secondlife.com

[14] Flickr http://www.flickr.com

[15] Picasa Album photo http:// www.picasa.com

[16] Peersim http://sourceforge.net/projects/peersim

[17] Talel Abdessalem Pierre Senellart. *Concepts et modèles des Webs communautaires*

[18] Olivier Beaumont, Anne-Marie Kermarrec, Étienne Rivière : *RayNet : approximation de structures complexes pour la recherche de données multidimensionnelles à grande échelle*

[19] Vincent Gramoli *Mémoire atomique auto-reconfigurable pour systèmes P2P*
IRISA-INRIA, Campus de Beaulieu 35042, Rennes, France.
Vincent.Gramoli@irisa.fr

[20] J. Kleinberg. *The Small-World Phenomenon: An Algorithmic Perspective*.
In Proceedings of the 32nd ACM Symposium on Theory of
Computing, 2000.

[21] D. P. *Luebke. A developer's survey of polygonal simplification algorithms.*
IEEE Computer Graphics and Applications, 21(3):24–35,
2001.

[22] S.-Y. Hu and G.-M. Liao. *Scalable peer-to-peer networked virtual environment.*
In NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, pages 129–133. ACM, 2004.

[23] Abdelkader Lahmadi. *Recherche de données dans les réseaux pair-à-pair*
DHT : tables de hachage distribués CA , CHORD

[24] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. *Chord: A scalable peer-to-peer lookup service for internet applications*. In *Proc. of SIGCOMM'01*, 2001.