Évaluation de la fiabilité d'une table de hachage distribuée construite dans un plan hyperbolique

Telesphore Tiendrebeogo, Daouda Ahmat, Damien Magoni

Évaluation de la fiabilité d'une table de hachage distribuée construite dans un plan hyperbolique

Telesphore Tiendrebeogo, Daouda Ahmat, Damien Magoni

Université de Bordeaux - LaBRI
351, Cours de la Libération
F-33405 Talence Cedex
{tiendreb,adaouda,magoni}@labri.fr

RÉSUMÉ. *Une table de hachage distribuée doit pouvoir acheminer les messages de requête en supportant le passage à l'échelle. Bien que plusieurs solutions existent déjà, elles nécessitent souvent une topologie prédéfinie entre les nœuds ainsi que des tables de routage. Nous proposons d'utiliser un algorithme de routage glouton basé sur des coordonnées virtuelles provenant du plan hyperbolique afin de construire une table de hachage distribuée ayant une topologie quelconque et ne requiérant pas de table de routage. Nous définissons à l'aide de cet algorithme un nouveau système de table de hachage distribuée fiable et supportant le passage à l'échelle. Nous fournissons une analyse des coûts de complexité et nous évaluons ses performances par des simulations en les comparant à des solutions existantes. Les résultats montrent que notre système apporte de la flexibilité aux nœuds tout en restant fiable et extensible en présence de remous.*

ABSTRACT. *A distributed hash table needs to route requests in a scalable way. Although several solutions do exist, they often require nodes to connect to each others by following a specific topology and to maintain routing tables. We propose a greedy routing algorithm based on virtual coordinates taken from the hyperbolic plane for building a distributed hash table while letting nodes connect to any others as they wish and without maintaining any routing table. In this paper, we use this algorithm to define a new scalable and reliable distributed hash table system. We provide a scalability analysis and we evaluate its performance and efficiency by carrying out simulations including other existing solutions. Results show that our system brings more flexibility to the nodes while still being scalable and reliable in presence of churn.*

MOTS-CLÉS : *plan hyperbolique, routage glouton, table de hachage distribuée.*

KEYWORDS: *distributed hash table, greedy routing, hyperbolic plane.*

DOI:10.3166/TSI..

1. Introduction

Les tables de hachage distribuées (THD) ont été beaucoup étudiées pour la capacité extensible de stockage et de récupération de données qu'elles fournissent à de nombreuses applications pair-à-pair (P2P). Les solutions existantes ont toutes leurs avantages et leurs inconvénients comme cela est montré dans (Lua *et al.*, 2005) mais elles nécessitent en général des topologies contraintes et des tables de routage dont la taille est en général de l'ordre de $O(log(n))$. Nous proposons un système nommé CLOAK permettant de construire un réseau recouvrant (*overlay* en anglais) sans imposer une topologie particulière aux nœuds participants (tel qu'un anneau, un arbre ou un hypercube). Chaque nœud gère sa portion d'espace d'adressage et les nœuds peuvent se connecter arbitrairement entre eux. Ils peuvent joindre ou quitter le réseau P2P de façon efficace sans avoir besoin d'une connaissance globale du réseau. Nous proposons aussi une fonction de correspondance pour stocker les paires (clé, valeur) sur les nœuds afin que la récupération soit efficace et évite la surcharge d'une région particulière du réseau. Nous avons effectué une analyse et des simulations pour montrer le potentiel de notre solution en termes d'extensibilité (i.e., passage à l'échelle) et de fiabilité pour construire une THD. Cet article est une version étendue de notre précédent article publié dans (Tiendrebeogo *et al.*, 2012). Nous y avons ajouté la définition et l'évaluation des mécanismes de réplication (cf. section 4.2) pour assurer la fiabilité de la THD en présence de remous (*churn* en anglais) ainsi que des nouveaux résultats de simulation sur des grands réseaux de 10 000 nœuds (cf. section 6). Ce travail se base sur nos précédents travaux de recherche (Cassagnes *et al.*, 2011) décrivant l'utilisation et les limitations du plan hyperbolique comme espace d'adressage virtuel. Nos contributions présentées dans cet article sont les suivantes :

– nous définissons les algorithmes d'adressage distribué et de routage glouton utilisés par le réseau recouvrant (cf. section 3) ;

– nous définissons les algorithmes de correspondance (clé vers adresse) et de réplication utilisés par notre THD (cf. section 4) ;

– nous comparons les coûts de complexité de notre solution à ceux d'autres THD existantes (cf. section 5) ;

– nous fournissons des résultats obtenus par simulation concernant l'évaluation des performances de notre THD et d'autres THD existantes (cf. section 6).

2. État de l'art

Les tables de hachage distribuées ont fait l'objet de nombreuses études du fait de leurs propriétés attractives qui peuvent être résumées comme suit (Balakrishnan *et al.*, 2003) :

L'auto-organisation : l'organisation et la maintenance du système sont distribuées entre les nœuds. Il n'y a pas besoin d'un serveur central pour gérer le stockage et la distribution des données. Ainsi, le problème lié à la disparition d'un nœud est supprimé et la tolérance aux pannes augmente.

Le passage à l'échelle : du fait de leur structure décentralisée, les THD peuvent être facilement étendues pour gérer un grand nombre de nœuds.

La robustesse : les départs, les arrivées ou les pannes de nœuds causent un minimum de perturbations dans le système et affectent seulement une partie de l'ensemble du réseau.

De nombreuses THD ont été étudiées ces dix dernières années et nous renvoyons le lecteur à (Lua *et al.*, 2005) pour un état de l'art détaillé de ces solutions. Notre proposition emprunte certains éléments à ces THD préexistantes. Notre fonction de correspondance pour placer les clés sur le cercle unité est similaire à celui défini dans Chord (Stoica *et al.*, 2001). Cependant, contrairement à Chord, nous ne plaçons pas les nœuds eux-mêmes sur ce cercle mais à l'intérieur du disque unité en utilisant des coordonnées complexes. Similairement à CAN (Ratnasamy *et al.*, 2001), nous utilisons un espace multidimensionnel de coordonnées, mais au lieu d'utiliser un multitore cartesien de dimension d, nous utilisons le plan hyperbolique de dimension 2 de courbure -1 nommé $\mathbb{H}^2$. Notre algorithme de routage glouton est basé sur une distance comme dans Kademlia (Maymounkov, Mazières, 2002). Mais contrairement à Kademlia qui utilise une métrique XOR, nous utilisons la distance hyperbolique définie dans le modèle du disque de Poincaré. Un autre gros avantage de notre algorithme de routage glouton est qu'il ne nécessite aucune table de routage contrairement aux algorithmes de routage basés sur les préfixes tels que ceux utilisés dans Pastry (Rowstron, Druschel, 2001) et Tapestry (Zhao *et al.*, 2004). Seules les coordonnées des voisins d'un nœud sont requises pour acheminer un message. L'idée d'utiliser le plan hyperbolique comme espace d'adressage virtuel provient des travaux de Kleinberg (Kleinberg, 2007). Nous avons étendu ses travaux afin de permettre au réseau d'être dynamique et extensible. En effet, comme nous créons un réseau recouvrant, nous sommes en mesure de fixer le degré de l'arbre d'adressage à une valeur determinée au préalable et nous n'avons pas besoin de découvrir le nœud de plus haut degré. De plus, nous avons défini une fonction de correspondance originale, tandis que Kleinberg a suggéré l'utilisation de CAN pour implémenter une THD. Notre approche par rayon de stockage présentée à la section 4.1 permet de gérer automatiquement l'extension et la diminution du réseau recouvrant ce qui optimise le stockage des clés dans un réseau très dynamique. Plus de détails concernant l'extensibilité et l'efficacité de notre solution de création d'un réseau recouvrant basé sur un addressage et un routage hyperboliques peuvent être trouvés dans nos précédents travaux (Cassagnes *et al.*, 2011). L'évaluation des THD, définie dans (Naor, Wieder, 2003), peut se baser sur plusieurs paramètres :

Les coûts de connexion/déconnexion : les mouvements des participants devraient causer un minimum de perturbations et être gérés facilement par le système. Quand un nœud rejoint ou quitte la THD, seul un petit nombre de nœuds devrait être impacté.

La congestion : la performance du service ne devrait pas subir de goulot d'étranglement ce qui implique que le coût de la recherche devrait être réparti entre les participants.

Les distances : le chemin d'une requête de recherche devrait impliquer aussi peu de participants que possible.

La tolérance aux pannes : les pannes des nœuds et des connexions ne doivent pas impacter le fonctionnement du système.

Nous utilisons certains de ces indicateurs ainsi que d'autres dans la section 5 concernant l'analyse de la complexité des coûts et dans la section 6 concernant l'évaluation des performances.

3. Adressage et routage glouton dans le plan hyperbolique

Nous présentons ici l'algorithme permettant d'adresser les nœuds du réseau recouvrant afin de pouvoir utiliser un algorithme de routage glouton qui garantit un routage à 100 % (donc exempt de tout minimum local) lorsque le réseau est statique. Nous présentons aussi des méthodes pour réparer la cohérence du réseau lorsqu'il est dynamique. Nous rappelons au préalable quelques éléments de géométrie hyperbolique qui servent de fondation à nos algorithmes.

3.1. Plan hyperbolique

Nous rappelons ici quelques propriétés de la géométrie hyperbolique. Chaque nœud du réseau se verra attribuer une adresse virtuelle qui sera définie par les coordonnées d'un point du plan hyperbolique. Nous utilisons le plan hyperbolique noté $\mathbb{H}^2$ de rayon de courbure -1. Le modèle que nous utilisons pour représenter ce plan hyperbolique s'appelle modèle du disque de Poincaré. Dans ce modèle, nous nous référons aux points du plan en utilisant des coordonnées complexes. Une propriété importante du plan hyperbolique est que nous pouvons le paver avec des polygones de taille quelconque, appelés *p-gons*. Chaque pavage est représenté par une notation de la forme $\{p, q\}$ où chaque polygone possède p faces et où q polygones se touchent à chaque sommet. Cette forme de notation est appelée un symbole de *schläfli*. Il existe un pavage hyperbolique $\{p, q\}$ pour chaque couple (p, q) obéissant à l'inégalité $(p-2)*(q-2) > 4$. Dans un pavage, p est le nombre de faces des polygones du *primal* et q est le nombre de faces des polygones du *dual*. Nous faisons tendre p vers l'infini, transformant ainsi le primal en un arbre régulier infini de degré q. Le *dual* est alors découpé en un nombre infini de *q-gons*. Ce pavage particulier découpe le plan hyperbolique en espaces distincts et construit un arbre ayant des sommets aux coordonnées uniques, comme cela a été montré par Coxeter *et al.* dans (Coxeter, Whitrow, 1950 ; Coxeter, 1954). Nous appelons cet arbre un **arbre d'adressage**. Comme il s'agit d'un arbre régulier de degré q, le nœud racine peut donner une adresse unique à chacun de ses q voisins et tout autre nœud que la racine peut donner une adresse unique à $q - 1$ voisins.

Dans le modèle du disque de Poincaré, la distance entre deux points quelconques z et w est donnée par une courbe qui minimise la longueur entre ces deux points et qui est appelée une *géodésique* du plan hyperbolique. Calculer la longueur d'une *géodésique*

entre deux points z et w et ainsi obtenir leur distance hyperbolique $d_{\mathbb{H}}$ est possible en utilisant la métrique de Poincaré (Beardon, Minda, 2006) :

$$d_{\mathbb{H}}(z, w) = argcosh(1 + 2\delta) \tag{1}$$

dans laquelle on définit δ comme étant égal à :

$$\delta = \frac{|z - w|^2}{(1 - |z|^2)(1 - |w|^2)} \tag{2}$$

La distance hyperbolique $d_{\mathbb{H}}(z, w)$ est additive et est une métrique Riemannienne. Dans l'arbre régulier de degré q défini ci-dessus, la distance entre deux nœuds voisins quelconques est toujours la même. Autrement dit, toutes les arêtes de l'arbre possèdent la même longueur.

3.2. Arbre d'adressage

Nous expliquons maintenant comment nous créons l'arbre d'adressage dans le réseau recouvrant et nous verrons ensuite comment les messages sont routés dans ce même réseau recouvrant. La première étape dans la création d'un réseau recouvrant est de démarrer le premier nœud et de fixer le degré q de l'arbre d'adressage. Nous rappelons que les coordonnées hyperboliques (i.e., un nombre complexe) d'un nœud de l'arbre d'adressage sont utilisées comme l'adresse correspondant au nœud dans le réseau recouvrant. Le premier nœud, situé au centre du disque de Poincaré, possède l'adresse (0, 0). Tout nœud du réseau peut donc distribuer les adresses correspondant à ses fils dans l'arbre d'adressage. Le degré q détermine combien d'adresses chaque nœud sera capable de donner. Le degré q de l'arbre est défini au commencement pour toute la durée de vie du réseau recouvrant. Le réseau recouvrant est alors construit incrémentalement, chaque nœud joignant un ou plusieurs nœuds existants. Au cours du temps, les nœuds quitteront le réseau recouvrant jusqu'à ce qu'il n'y ait plus de nœud à la fin dans le réseau recouvrant. Dans notre réseau recouvrant, un nœud peut se connecter à un ou plusieurs autres à chaque instant dans le but d'obtenir une adresse. Si un nœud a distribué toutes ses adresses, il doit rediriger toute nouvelle demande vers d'autres nœuds du réseau recouvrant, cependant il peut quand même accepter des liens supplémentaires dits liens redondants.

L'algorithme 3 montre comment calculer les adresses qui peuvent être données aux fils d'un nœud donné. Comme indiqué ci-dessus, le premier nœud prend l'adresse (0, 0), est la racine de l'arbre, et peut assigner q adresses, ensuite chaque nœud suivant peut assigner $q - 1$ adresses. Cet algorithme exécuté de façon distribuée sur chaque nœud assure que les nœuds ont des coordonnées uniques. La connaissance globale du réseau recouvrant n'est pas nécessaire, un nouveau nœud peut obtenir des coordonnées simplement en demandant à un nœud existant d'être son père et de lui donner une adresse. Si le nœud demandé a déjà donné toutes ses adresses, le nouveau nœud doit demander une adresse à un autre nœud du réseau. Au pire, un nœud arrivera toujours a obtenir une adresse en se connectant à un nœud qui est une feuille de l'arbre

et qui par conséquent n'a pas encore donné d'adresse. Quand un nouveau nœud obtient une adresse, il calcule les adresses (i.e., les coordonnées hyperboliques) de ses futurs fils dans l'arbre d'adressage afin de les proposer à des nouveaux nœuds. L'arbre d'adressage est ainsi incrémentalement construit en même temps que le réseau recouvrant.

Afin de calculer les coordonnées des points du plan hyperbolique, nous utilisons les isométries. Dans notre contexte, une isométrie est une transformation géométrique définie dans le plan complexe et composée d'une rotation et d'une translation. Chaque isométrie est définie par deux nombres complexes. Le premier nombre complexe r définit la rotation et le second nombre complexe t définit la translation. Etant donné n'importe quel nombre complexe z et une isométrie $I = \{r, t\}$, alors :

$$I(z) = \frac{r \times z + t}{1 + \overline{t} \times r \times z} \tag{3}$$

Etant donné deux isométries $I_1 = \{r_1, t_1\}$ et $I_2 = \{r_2, t_2\}$, alors :

$$I_1 \times I_2 = \left\{ \frac{r_1 \times r_2 + r_2 \times t_1 \times \overline{t_2}}{r_1 \times t_2 \times \overline{t_1} + 1}, \frac{r_1 \times t_2 + t_1}{r_1 \times t_2 \times \overline{t_1} + 1} \right\} \tag{4}$$

Etant donné une isométrie $I = \{r, t\}$, alors son inverse est définie par :

$$inv(I) = \{\overline{r}, 0\} \times \{1, -t\} \tag{5}$$

Les générateurs sont les isométries utilisées pour créer les points de l'arbre d'adressage. Il y a q générateurs. Chaque générateur est construit à partir de deux isométries, une isométrie de rotation notée R :

$$R = \left\{ e^{i\frac{2\pi}{q}}, 0 \right\} \tag{6}$$

et une isométrie de translation notée T construite ainsi :

$$T = \left\{ 1, \tanh\left(\operatorname{argcosh}\left(\frac{1}{\sin\left(\frac{\pi}{q}\right)} \right) \right) \right\} \times \{-1, 0\} \tag{7}$$

On peut ainsi définir les générateurs pour être utilisés dans le calcul des adresses du réseau, comme indiqué dans l'algorithme 1.

Le nœud racine de l'arbre d'adressage, qui est aussi le premier nœud du réseau recouvrant est initialisé avec les paramètres donnés dans l'algorithme 2.

Notons que Margenstern propose aussi une méthode pour créer un arbre hyperbolique dans (Margenstern, 2002), cependant son approche très géométrique ne s'attache pas à rendre son algorithme distribué, ce qui est pour nous une condition sine qua non. Notre approche plus analytique ne fait qu'appel à des calculs élémentaires dans le plan complexe et qui peuvent être distribués sur tous les nœuds du réseau recouvrant.

Algorithme 1. Définition des générateurs

`DéfinirGénérateurs` (*Générateurs[], q*)
begin
  $R = \left\{ e^{i\frac{2\pi}{q}}, 0 \right\}$
  $T = \left\{ 1, \tanh\left(\operatorname{argcosh}\left(\frac{1}{\sin\left(\frac{\pi}{q}\right)} \right) \right) \right\} \times \{-1, 0\}$
  for $i \leftarrow 0$ **to** $q - 1$ **do**
    $Générateurs[i] = R^i \times T \times inv(R^i)$

Algorithme 2. Initialisation des paramètres du nœud racine

`InitialiserParamètresRacine` (*Racine*)
begin
  $Racine.Coordonnées \leftarrow (0, 0)$
  $Racine.Index \leftarrow 0$
  $Racine.Isométrie \leftarrow \{1, 0\}$

Algorithme 3. Calcul des coordonnées des fils d'un nœud

`CalculerCoordonnéesFils` (*Père, Générateur[], q*)
begin
  for $i \leftarrow 0$ **to** $q - 1$ **do**
    if $i = 0$ **and** $Père \neq Racine$ **then**
      continue
    $Fils.Index \leftarrow Père.Index + i \bmod q$
    $Fils.Isométrie \leftarrow Père.Isométrie \times Générateurs[Fils.Index]$
    $Fils.Coordonnées \leftarrow Fils.Isométrie(0)$

Comme dans tout réseau recouvrant nous avons besoin d'une méthode de démarrage (*bootstrap* en anglais). Un nouveau nœud non encore inséré dans le réseau recouvrant doit connaître l'adresse IP d'au moins un nœud se trouvant déjà dans le réseau recouvrant afin de pouvoir ensuite obtenir des propositions d'adresses provenant de nœuds pouvant distribuer des adresses. Cette adresse IP peut être découverte par des moyens extérieurs tels qu'une page *web*, un message instantané ou un courrier électronique. Un nœud du réseau recouvrant dont l'adresse IP est diffusée à l'extérieur du réseau recouvrant est appelé un portail (*gate* en anglais).

3.3. Routage glouton

Nous proposons ici un algorithme de routage glouton basé sur une métrique hyperbolique pour des réseaux recouvrants à large échelle. Quand un nouveau nœud est connecté à un ou plusieurs nœuds situés à l'intérieur du réseau recouvrant et qu'il a obtenu une adresse de l'un de ces nœuds, il peut commencer à envoyer des messages. Nous postulons que le nœud connaît l'adresse du nœud de destination. Dans le cas contraire, nous supposons que les nœuds possèdent des noms permanents et que les paires (nom, adresse) sont stockées dans la THD définie dans la section 4. Nous supposons alors que le nœud source connaît le nom du nœud destinataire et retrouve son adresse à partir de son nom.

Le processus de routage est réalisé dans chaque nœud situé sur le chemin de la source à la destination par l'utilisation de l'algorithme glouton basé sur la distance hyperbolique entre les nœuds. Quand un message est reçu par un nœud, celui-ci calcule la distance hyperbolique de chacun de ses voisins à la destination et transmet le message à son voisin qui est le plus proche de la destination. Si aucun voisin n'est plus proche que le nœud lui-même, le message a atteint un minimum local et d'autres techniques expliquées ci-après doivent être utilisées pour permettre au message d'atteindre sa destination. Sinon si aucune autre technique ne réussit, le message est détruit. L'algorithme 4 permet de transmettre un message au prochain nœud se rapprochant le plus de sa destination.

Algorithme 4. Acheminement d'un message dans le réseau

```
ProchainSaut (Nœud, Message) return Nœud
begin
    N_min ← Noeud
    u ← Noeud.Coordonnées
    w ← Message.NoeudDestinationNode.Coordonnées
    d_min ← argcosh(1 + 2|u−w|² / ((1−|u|²)×(1−|w|²)))
    foreach Voisin ∈ Noeud.Voisins do
        v ← Voisin.Coordonnées
        d ← argcosh(1 + 2|v−w|² / ((1−|v|²)×(1−|w|²)))
        if d < d_min then
            d_min ← d
            N_min ← Voisin
    return N_min
```

Dans un réseau recouvrant fortement dynamique, plusieurs problèmes peuvent apparaître. Les auteurs de (Cvetkovski, Crovella, 2009) précisent que le routage dans le plan hyperbolique est robuste à condition que l'intégrité de l'arbre d'adressage soit maintenu. Dans un réseau recouvrant créé sur un environnement réel, les défaillances

des nœuds et des liens sont fréquents. Dans un réseau recouvrant, nous pouvons définir deux niveaux de pannes :

– le premier niveau de panne impacte l'arbre d'adressage ;
– le second niveau de panne impacte le graphe du réseau recouvrant.

Au premier niveau, si un lien inclus dans l'arbre échoue, alors le routage glouton hyperbolique échouera pour tous les chemin passant par ce lien. De plus, si un nœud autre qu'un nœud feuille échoue, ceci pourrait partitionner l'arbre en une forêt à q sous-arbres et ainsi perturberait la connectivité de l'arbre (Cvetkovski, Crovella, 2009). Il faut donc utiliser une technique de réparation et de maintenance de l'arbre d'adressage. Si l'arbre d'adressage est brisé, deux principales approches peuvent être utilisées pour restaurer la connectivité :

1. supprimer les adresses attribuées aux nœuds situés au-delà du nœud ou du lien tombé en panne et réassigner des adresses à ces nœuds (renumérotation) ;

2. restaurer l'arbre en remplaçant le nœud en panne par un nouveau nœud ayant les mêmes connexions que son prédécesseur (restauration).

La première solution que nous appelons la méthode de renumérotation est très simple à implémenter mais elle peut-être coûteuse si la taille du réseau recouvrant situé au-delà du nœud/lien en panne est importante, car elle peut conduire à la renumérotation d'une grande partie du réseau. La seconde solution que nous appelons la méthode de restauration est moins coûteuse parce que les adresses sont gardées, mais elle est beaucoup plus difficile à implémenter, car il peut être délicat pour un nouveau nœud d'établir les même connexions que celles qu'avait le nœud tombé en panne. Par exemple, il faut que le nouveau nœud récupère les adresses IP des voisins de l'ancien nœud. Dans la partie concernant les simulations, nous n'avons implémenté que la renumérotation pour l'instant. Au second niveau, si un lien du réseau recouvrant qui tombe en panne n'appartient pas à l'arbre d'adressage ou si un nœud feuille tombe en panne alors le routage glouton hyperbolique s'effectuera toujours sans erreur bien que les chemins du réseau recouvrant soient susceptibles d'être plus longs.

Comme les méthodes de réparation du réseau recouvrant ne sont pas instantanées, le trafic peut être perturbé durant tout le temps où la cohérence de l'adressage est brisée. Afin de contourner d'éventuels minima locaux dues aux incohérences, diverses heuristiques peuvent être utilisées. Un exemple de ces heuristiques est une technique appelée *Gravity-Pressure GP* qui est présentée dans (Cvetkovski, Crovella, 2009) et est aussi utilisée dans (Papadopoulos *et al.*, 2010). En arrivant dans un nœud étant minimum local, le message entre en mode pression. Dans ce mode, le message maintient une liste des nœuds qu'il a visité depuis qu'il est entré dans ce mode et le nombre de visites de chaque nœud. Ce processus continue jusqu'à ce que le message trouve un nœud dont la distance à la destination est plus petite que la distance du minimum local courant. La solution présentée nécessite de stocker des informations dans le message ce qui peut poser des problèmes d'implémentation.

4. THD dans le plan hyperbolique

Nous présentons ici les algorithmes permettant de définir une THD fiable sur les nœuds du réseau recouvrant. Nous détaillons une fonction de correspondance (*mapping* en anglais) permettant de définir le nœud ayant à stocker une paire (clé, valeur) donnée ainsi que des mécanismes de réplication permettant de fiabiliser la THD en répliquant les données.

4.1. Fonction de correspondance

Notre solution est un système de THD structurée qui utilise l'adressage local et le routage glouton présentés dans la section 3. Dans une THD, les informations sont stockées sous la forme de paires (clé, valeur) de manière équilibrée sur tous les nœuds du réseau recouvrant. Cependant dans notre solution, tous les nœuds du réseau recouvrant ne stockeront pas forcément des paires. Nous appelons *stockeur*, un nœud du réseau recouvrant qui stocke des paires. Dans une THD, les requêtes de stockage sont nommées PUT et les requêtes de recherche sont nommées GET. La figure 1 montre comment et où une paire donnée est stockée dans le réseau recouvrant. La profondeur d'un nœud dans l'arbre d'adressage est définie comme étant le nombre de nœuds ascendants à traverser avant d'atteindre la racine de l'arbre (incluant la racine elle-même). Quand le réseau recouvrant est créé, une profondeur maximale p_{max} est choisie. Cette valeur est définie comme étant la profondeur maximale que peut avoir n'importe quel stockeur de l'arbre. Tous les nœuds qui ont une profondeur inférieure ou égale à p_{max} peuvent stocker des paires (clé, valeur) et être ainsi des stockeurs. Tous les nœuds qui ont une profondeur supérieure à p_{max} ne stockent pas de paires.

Lorsqu'un nœud souhaite stocker une information définie par une paire (clé, valeur) dans la THD, il doit déterminer l'adresse du nœud qui va stocker cette paire à partir de la clé. Il créé la clé hachée en appliquant l'algorithme SHA-1 sur la clé (qui est un identifiant quelconque supposé unique). Il divise ensuite la clé hachée de 160-bits en $r_c = 5$ parties égales de 32-bits chacune (pour la redondance du stockage, cf. ci-dessous). Le nœud sélectionne une première sous-clé et la fait correspondre à un angle par une transformation linéaire. L'angle est donné par :

$$\alpha = 2\pi \times \frac{\texttt{Sous-clé de 32-bits}}{\texttt{0xFFFFFFFF}} \tag{8}$$

Le nœud calcule alors un point virtuel sur le cercle unité :

$$v(x,y) \text{ avec} \begin{cases} x = cos(\alpha) \\ y = sin(\alpha) \end{cases} \tag{9}$$

Ensuite, le nœud détermine les coordonnées hyperboliques (i.e., donc l'adresse) du stockeur le plus proche du point virtuel calculé ci-dessus. Ce stockeur possède la profondeur maximale p_{max} et est considéré comme le premier stockeur potentiel. Dans la figure 1, nous fixons la profondeur maximale p_{max} à trois pour des raisons de

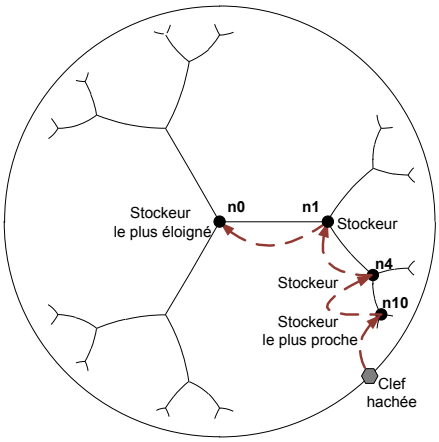

Figure 1. Stockage d'une paire dans le réseau recouvrant

lisibilité. Il est important de noter que ce plus proche stockeur du point virtuel ($n10$ sur la figure) peut ne pas exister en réalité si aucun nœud actuel du réseau recouvrant ne possède cette adresse. La requête est routée à l'intérieur du réseau recouvrant par l'utilisation de l'algorithme glouton de la section 3. Si la requête échoue parce que le stockeur n'existe pas ou à cause de l'échec d'un nœud ou d'un lien, elle est redirigée vers le père du stockeur initialement calculé ($n4$ sur la figure). Ce processus continue jusqu'à ce que la requête atteigne un stockeur qui existe. En atteignant un stockeur existant, la paire est stockée dans ce stockeur. La requête peut ainsi remonter l'arbre d'adressage jusqu'à sa racine ayant l'adresse (0, 0) qui est le stockeur le plus éloigné du point virtuel. Le chemin pris par la requête du plus proche stockeur au plus éloigné est défini comme étant le **rayon de stockage**.

Ce processus assure que les paires sont toujours stockées prioritairement dans le stockeur le plus proche du cercle unité et en dernier dans le stockeur le plus proche de la racine. On remarque que la paire est stockée sur le(s) même stockeur(s) peu importe l'adresse du nœud origine de la requête (condition nécessaire pour que l'algorithme soit correct). Si l'arbre d'adressage est déséquilibré, beaucoup de paires peuvent être stockées dans des nœuds proches du centre et ainsi les surcharger. Dans le but de résoudre ce problème, chaque stockeur n'est en mesure de stocker qu'un nombre maximum de paires. Au delà, toute nouvelle paire est refusée et la requête est redirigée vers les ascendants.

Enfin, précisons que notre solution possède la propriété du hachage **consistant** : si un nœud apparaît ou disparaît, seules les paires stockées par les nœuds adjacents qui changent d'adresse sont affectées. Les clés des autres stockeurs ne sont pas im-

pactées et l'ensemble du système reste cohérent. Comme dans beaucoup de systèmes existants, les paires sont stockées suivant une stratégie de type *soft state*. Une paire doit être stockée par son créateur à intervalle de temps réguliers Δ_t. Ainsi, lorsque la topologie du réseau change et que les adresses de certains nœuds changent, les clés rafraichies se retrouvent éventuellement stockées dans de nouveaux stockeurs. De plus, tout stockeur purge les clés les plus anciennes, ayant dépassé une durée de stockage de Δ_t, lorsqu'il doit stocker de nouvelles clés mais que sa capacité de stockage est saturée. Un message de suppression peut être envoyé par le créateur pour supprimer la paire avant la fin de l'intervalle de temps.

4.2. Mécanismes de réplication

Dans le but de rendre notre système de THD robuste et efficace, nous avons introduit deux mécanismes de réplication. Pour fournir de la redondance, le nœud effectue le processus de stockage décrit ci-dessus pour chacune des quatre autres sous-clés de 32 bits. Ainsi, cinq rayons de stockage différents sont utilisés et cela améliore les chances de succès des requêtes de recherche ainsi que l'homogénéité de la distribution des paires sur les stockeurs. La division de la clé en $r_c = 5$ sous-clés est arbitraire et pourrait être augmentée ou réduite en fonction du besoin de redondance. Nous désignons ce mécanisme de redondance par le terme de réplication **circulaire**. De façon générale, a partir d'une paire nommée A et de l'empreinte de hachage de sa clé avec l'algorithme SHA-1, nous obtenons une clé de 160-bits que nous répartissons en un certain nombre arbitraire r_c de sous-clés. Chaque sous-clé permet ainsi de calculer un point P_i situé sur la bordure du disque du modèle de Poincaré. Le nœud N_i susceptible de contenir une réplique de la paire A, est le nœud le plus proche géométriquement du point P_i. L'ensemble des r_c répliques circulaires est constitué des N_0, N_1, ..., N_{r_c} nœuds les plus proches des points P_0, P_1, ..., P_{r_c} situés sur le cercle unité.

De plus, toujours pour des raisons de redondance, une paire peut être stockée dans plus d'un stockeur faisant partie du rayon de stockage. Un stockeur peut en effet stocker une paire et ensuite rediriger sa requête de stockage vers son ascendant pour qu'il la stocke à son tour. Le nombre de copies d'une paire suivant le rayon de stockage peut être une valeur arbitraire r_r définie à la création du réseau recouvrant. Nous utilisons le terme de réplication **radiale** pour désigner la réplication de la paire A suivant le rayon de stockage. En effet, à partir du premier stockeur existant situé le plus proche du cercle, nous répliquons la paire r_r fois sur les nœuds ascendants en direction de la racine de l'arbre d'adressage. On s'arrête, soit lorsque l'on atteint un nombre de réplications radiales égal à $r_r = \left\lceil \frac{log(n)}{log(q)} \right\rceil$ (où q correspond au degré de l'arbre d'adressage hyperbolique et n à la taille du réseau recouvrant), soit lorsque l'on atteint la racine de l'arbre d'adressage.

Pour conclure, nous avons donc défini deux mécanismes de réplication pour le stockage des copies d'une paire donnée :

1. nous pouvons utiliser r_c rayons de stockage grâce à la création de plusieurs sous-clés uniformément distribuées ;

2. nous pouvons stocker jusqu'à r_r paires dans les stockeurs situés sur le même rayon de stockage.

Ces mécanismes permettent à notre système de THD de faire face à une croissance non uniforme du réseau recouvrant et d'assurer qu'une paire sera stockée de façon redondante pour maximiser le taux de succès des recherches. Le nombre r_c de sous-clés et le nombre r_r de copies dans un rayon sont des paramètres qui peuvent être fixés à la création du réseau recouvrant. Le choix de ces valeurs nécessite un compromis à faire entre l'amélioration de la fiabilité de la THD et le besoin d'espace de stockage dans les stockeurs.

Nous présentons maintenant les algorithmes de réplication permettant à la THD de résister au phénomène de remous en stockant plusieurs répliques lors de requêtes de stockage et en cherchant l'une de ces répliques lors de requêtes de recherche. Nous détectons les défaillances grâce au stockage périodique des paires dans les stockeurs. Après avoir rejoint la THD, chaque nœud doit procéder au stockage de ses paires à chaque intervalle de $\Delta_t = m$ unités de temps. Si après cet intervalle, un nœud ayant déjà fait une requête de stockage, ne réitère pas l'opération, la paire est supprimée au niveau de la table de stockage de ses stockeurs et cela indique que le nœud émetteur de ces paires est en panne ou injoignable. Ceci est illustré par la figure 2.

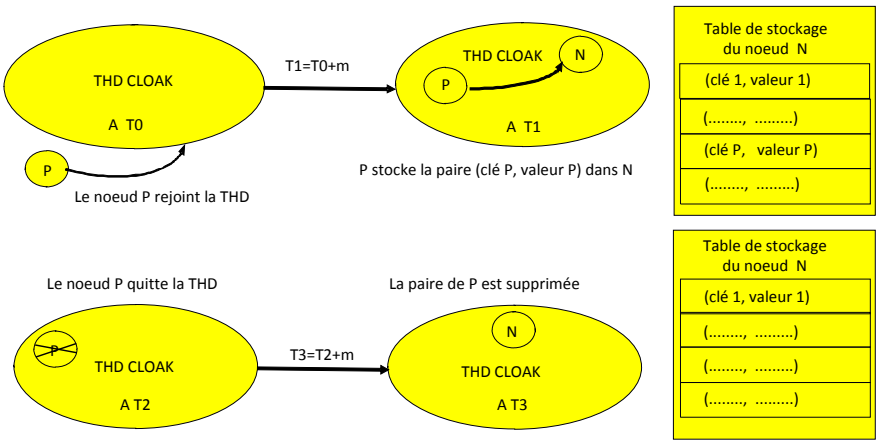

Figure 2. Mécanisme de gestion de la table de stockage

L'algorithme 5 explique comment notre système de THD stocke une paire donnée (clé, valeur) sur ses différents stockeurs. Nous supposons que la réplication radiale est effectuée systématiquement alors que la réplication circulaire est effectuée arbitrairement suivant le choix de la valeur r_c. Comme expliqué ci-dessus, une paire peut

stocker au maximum $r_r = \left\lceil \frac{log(n)}{log(q)} \right\rceil$ copies (ou répliques) le long du rayon de stockage avec n étant égal à la taille du réseau recouvrant et q étant égal au degré de l'arbre d'adressage.

Algorithme 5. Stockage avec réplication

```
StockerReplique(Source, Paire)
begin
    PUTRequest.Payload ← Paire;
    Cle ← Paire.Cle();
    CleHachee ← FonctionHachage(Cle);
    for red ← 1, r_c do
        SousCle[red] ← CalculerSousCle(CleHachee, red);
        prof ← p_max;
        i ← 0;
        while i ≤ ⌈log(N)/log(q)⌉ AND prof ≥ 0 do
            AdresseCible[red][prof] ← CalculerAdresse(SousCle[red], prof);
            Cible ← PUTRequest.Routage(Source, AdresseCible[red][prof]);
            if Cible != null then
                Cible ← PUTRequest.Payload;
                i++;
            prof − −;
```

L'algorithme 6 explique comment notre système recherche et récupère une paire stockée dans la THD. Grâce à la réplication circulaire, les requêtes peuvent être effectuées en parallèle pour réduire le temps de recherche au détriment d'un trafic supplémentaire cependant.

5. Analyse de la complexité des indicateurs de performance

Nous fournissons dans cette section une brève analyse de la complexité des coûts de notre système de THD. Nous présentons premièrement quatre indicateurs de performance que nous utilisons dans notre analyse. Ces indicateurs ont été définis et utilisés dans l'étude de (Lua *et al.*, 2005).

Distance : cet indicateur compte le nombre de nœuds à traverser jusqu'à atteindre la destination.

Congestion : cet indicateur compte le nombre de chemins qui traversent un nœud donné.

Etats : cet indicateur compte le nombre d'états qui doivent être stockés dans un nœud pour que le routage fonctionne. Il est donc égal au nombre d'entrées trouvées dans la table de routage du nœud.

Messages : cet indicateur compte le nombre de messages qui sont échangés quand un nœud rejoint ou quitte le réseau recouvrant.

Algorithme 6. Recherche avec réplication

```
ChercherReplique(Noeud, Cle)
begin
    GETRequest.Payload ← Cle;
    CleHachee ← FonctionHachage(Cle);
    for red ← 1, r_c do
        SousCle[red] ← CalculerSousCle(CleHachee, red);
        prof ← p_max;
        while prof ≥ 0 do
            AdresseCible[red][prof] ← CalculerAdresse(SousCle[red], prof);
            Cible ← GETRequest.Routage(Noeud, AdresseCible[red][prof]);
            if Cible != null then
                GETReply.Payload ← (Cle, Valeur);
                GETReply.Routage(Cible, Noeud);
                return ;
            prof − −;
```

Dans notre système, les nœuds dans le réseau recouvrant sont connectés les uns aux autres à leur guise ainsi aucune topologie n'est imposée. Chaque nœud peut avoir autant de liens qu'il le veut avec les autres nœuds et un lien est au moins un minimum pour établir une connexion au réseau recouvrant. La seule exigence est que l'arbre d'adressage du réseau recouvrant reste intègre pour supporter le routage glouton. Comme notre solution utilise un routage glouton, les nœuds n'ont pas besoin de construire et de maintenir des tables de routages. Par conséquent, le nombre d'**états** à maintenir dans chacun des nœuds est seulement égal au nombre de ses nœuds voisins. Ce nombre ne croît pas avec n, donnant ainsi une complexité constante $O(1)$.

Comme tout réseau recouvrant est au minimum égal à son arbre d'adressage (lorsqu'aucun lien redondant n'existe), les **distances** entre deux nœuds quelconques sont de l'ordre de $O(log(n))$ sauts. Si les nœuds ont un grand nombre de liens redondants (i.e., n'appartenant pas à l'arbre d'adressage), la distance moyenne pourra être beaucoup plus petite. Si la topologie du réseau recouvrant prend la forme d'un réseau à échelle libre (Gyarmati, Trinh, s. d.), les distances pourront être de l'ordre de $O(log(log(n)))$ comme cela est montré dans (Cohen, Havlin, 2003). Quelle que soit la topologie, le nombre de chemins traversant chaque nœud (i.e., son niveau de **congestion**) aura une probabilité d'au plus $O(log(n)/n)$.

Quand un nœud rejoint le réseau recouvrant, seuls ses voisins (ceux auxquels le nouveau nœud s'est connecté) ont besoin de mettre à jour leur table de voisinage avec une complexité du coût des **messages** indépendante de n. De façon similaire, quand un nœud quitte le réseau recouvrant, seuls ses voisins ont besoin de mettre à jour leur table de voisinage ce qui donne aussi une complexité du coût des messages de l'ordre de $O(1)$. Si l'arbre d'adressage est brisé et ne peut être restauré dans un temps raisonnable tel qu'expliqué dans nos travaux précédents (Cassagnes *et al.*, 2011), un

ré-adressage partiel du réseau peut intervenir pour des nœuds ayant des adresses dérivées des adresses du nœud ayant échoué ou hors de portée. Dans ce dernier cas, supposé être peu fréquent, la complexité du coût des messages est de l'ordre de $O(n)$. Le ré-adressage permet de fournir aux nœuds du réseau la capacité de se connecter aux autres nœuds qu'ils souhaitent. Si nous forçons les nœuds à se connecter à des nœuds spécifiques pour la restauration de l'arbre d'adressage (comme le fait Chord, où l'adresse IP du nœud détermine à quels autres nœuds il doit se connecter) alors la complexité attendue est de l'ordre de $O(1)$ pour un nœud quittant le réseau recouvrant. Ainsi, le ré-adressage doit être vu comme une option qui peut être écartée si les performances priment sur la flexibilité.

Le tableau 1 compare les complexités des quatre indicateurs définis ci-dessus pour des systèmes de THD variés incluant notre solution. Pour CAN, d est un entier supérieur ou égal à 2 et ainsi $0 < 1/d < 1$. Les résultats présentés dans ce tableau ont été rassemblés en utilisant les données publiées dans (Lua *et al.*, 2005) ainsi que les analyses ci-dessus. Précisons que les fonctions log avec des bases différentes sont considérées comme ayant des complexités équivalentes.

Tableau 1. Ordres de grandeur des indicateurs de différentes THD

THD	**Distance**	**Congestion**	**Etats**	**Messages**
CAN	O($n^{(1/d)}$)	O($n^{(1/d)}/n$)	O(1)	O(1)
Chord	O(log(n))	O(log(n)/n)	O(log(n))	O($log^2(n)$)
CLOAK	O(log(n))	O(log(n)/n)	O(1)	O(1) ou O(n)
Kademlia	O(log(n))	O(log(n)/n)	O(log(n))	O(log(n))
Pastry/Tapestry	O(log(n))	O(log(n)/n)	O(log(n))	O(log(n))
Viceroy	O(log(n))	O(log(n)/n)	O(log(n))	O(log(n))

6. Évaluation des performances par simulation

Contrairement à nos résultats préliminaires présentés dans (Tiendrebeogo *et al.*, 2012), les résultats montrés dans cette section proviennent de nouvelles simulations réalisées avec le simulateur *PeerSim* (Montresor, Jelasity, 2009). Dans le but d'évaluer notre solution, nous avons implémenté l'adressage, le routage et les requêtes de notre THD CLOAK. Nous avons de plus utilisé les implémentations existantes sur *PeerSim* de Chord (Stoica *et al.*, 2001), Kademlia (Maymounkov, Mazières, 2002) et MSPastry (Castro *et al.*, 2003) et avons effectué les simulations avec les mêmes paramètres (e.g., durée de simulation, topologie des nœuds, durées des sessions des nœuds, etc.) afin de pouvoir comparer les résultats à ceux de CLOAK. Le code source de notre THD CLOAK ainsi que tous les résultats de simulation, sont disponible ici [1].

1. http://www.labri.fr/perso/magoni/cape

6.1. Paramètres de simulation

Nous créons des réseaux recouvrants de taille initiale égale à 10 000 nœuds. Nous avons fixé le degré q de l'arbre d'adressage à 4 pour les simulations, conformément aux résultats de nos travaux précédents (Cassagnes *et al.*, 2011) montrant que la longueur moyenne des chemins est plus courte pour $q = 4$ dans un contexte de simulation dynamique.

Nous faisons varier le taux d'attrition t_a entre 10 % et 60 % par intervalles de 10 % et nous cherchons à voir comment le système réagit face à ces perturbations. Chaque simulation dure deux heures et les mesures sont effectuées sur des périodes de 10 minutes. Cela signifie que toutes les 10 minutes t_a % de nœuds quittent le réseau recouvrant et sont remplacés par des nouveaux nœuds suivant une loi de probabilité exponentielle. Les messages de stockage ainsi que ceux de recherche sont transmis à la fréquence de 1 000 par seconde. Chaque point de chaque graphique est la moyenne de 15 exécutions et l'écart type est donné.

6.2. Résultats de simulation

Les résultats sont placés en deux catégories. La première présente des résultats comparatifs entre Chord, CLOAK, Kademlia et MSPastry concernant le taux de succès des requêtes, la distance moyenne parcourue et la latence moyenne. La deuxième présente des résultats spécifiques à CLOAK concernant le nombre total de voisins (i.e., la valence), la quantité de trafic de messages de contrôle et la taille des tables de stockage.

6.2.1. Comparaison des THD

6.2.1.1. Taux de succès

La figure 3 montre la variation du taux de succès en fonction du nombre de réplications appliqué à notre système réseau recouvrant. On peut noter qu'en abscence de réplication, le taux de succès des requêtes de recherche est en moyenne de 83 %. Ce taux varie très peu (de l'ordre de 0,2 % à 4 %) lorsque l'on effectue 1 à 4 réplications sur le système. L'impact de la réplication sur le taux de succès commence à être significatif lorsque le taux d'attrition augmente. Ainsi, à 30 % de remous, 1 réplication induit une hausse du taux de succès de 2.1 %, et 4 réplications une hausse de 21 %. De même à 60 % de remous, le bénéfice de notre mécanisme de réplication s'accroît. Avec 1 réplication, le taux de succès connaît une hausse de 12 % et avec 4 réplications une hausse de 49 % par rapport au taux d'attrition de 30 %. Nous voyons ainsi l'avantage que procure notre stratégie de réplication sur le taux de succès. Dans la suite, nous étudierons le coût induit par une stratégie en terme de congestion du réseau. Comme les résultats obtenus dans le cas des requêtes de stockage sont très proche de ceux-ci, nous nous limitons au cas des requêtes de recherche pour éviter les redondances.

La figure 4 montre les taux de succès comparés des requêtes de recherche en fonction du taux d'attrition. Nous pouvons voir que tous les systèmes de THD s'exécutent

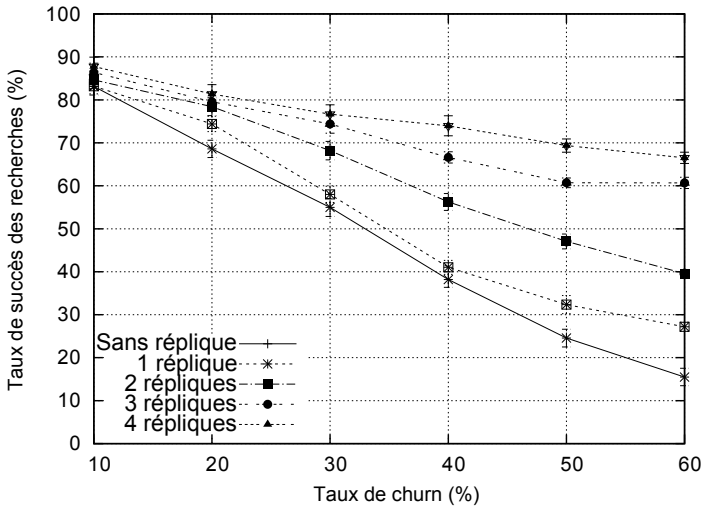

Figure 3. Taux de succès vs nombre de réplications

de façon similaire avec un ratio de succès linéairement décroissant avec le taux d'attrition. CLOAK a le deuxième meilleur taux de succès, après MSPastry et suivi de près par Chord avec presque les mêmes valeurs pour des taux d'attrition compris entre 10 % et 30 %. Kademlia a le plus bas taux de succès. Comme les courbes pour les requêtes de stockage sont très similaire à celles de la résolution, nous ne les montrons pas pour éviter la redondance.

6.2.1.2. Distance

La figure 5 montre les distances moyennes comparées des chemins mesurés en nombre de sauts des requêtes de recherche en fonction du remous. Ici encore, les systèmes de THD ont le même comportement avec une longueur de chemin (en sauts) faiblement décroissante lorsque le taux d'attrition augmente. MSPastry présente la plus courte longueur de chemin, suivi de CLOAK. Kademlia a en moyenne 0,6 saut de plus que MSPastry quel que soit le taux d'attrition, tandis que Chord a en moyenne une longueur du chemin supérieur de 1,2 pour MSPastry et de 1 pour CLOAK, bien que cette différence tende à décroître quand le taux remous est supérieur ou égal à 40 %. Les résultats pour les distances des requêtes de stockage montrés à la figure 6 sont très similaires à ceux des requêtes de recherche.

6.2.1.3. Latence

La figure 7 montre les latences moyennes comparées des requêtes de recherche en fonction du taux d'attrition. Etant donné qu'un chemin mesuré en nombre de sauts ne

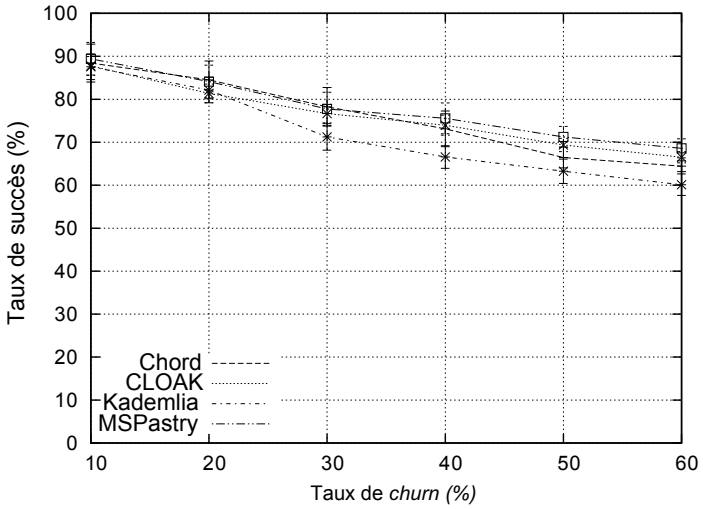

Figure 4. Taux de succès vs THD utilisée

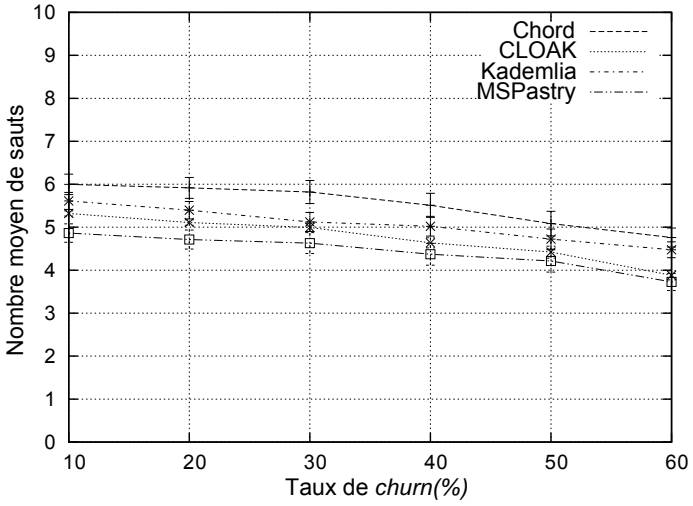

Figure 5. Distance parcourue par les GET

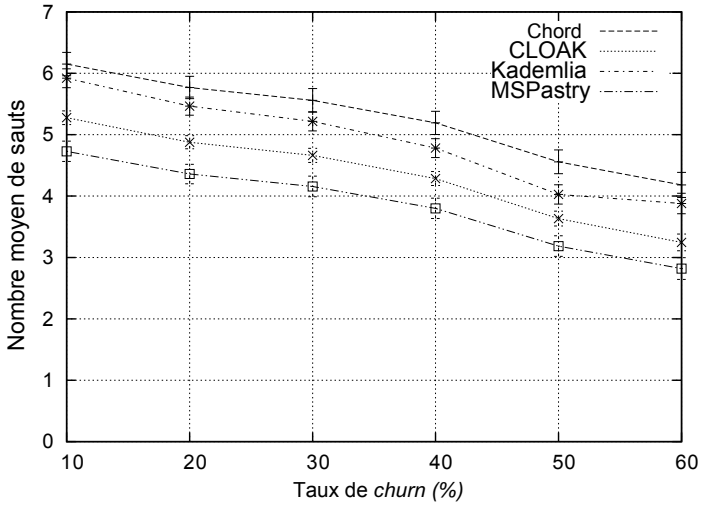

Figure 6. Distance parcourue par les PUT

ce traduit pas nécessairement par une latence élevée, nous avons mesuré cette dernière pour évaluer le temps pris par une requête pour arriver à destination. Tous les systèmes de THD ont presque la même latence à chaque taux d'attrition, excepté Kademlia qui a entre 200 ms et 280 ms de plus que les autres. Les résultats montrent en effet que malgré des distances supérieures, Chord s'exécute aussi rapidement que MSPastry et CLOAK lorsqu'on observe la latence. La figure 8 montre les latences moyennes comparées des requêtes de stockage en fonction du taux d'attrition. Les latences sont plus importantes que pour les requêtes de recherche car celles-ci doivent couvrir le temps nécessaire à la réplication. Là où une requête de recherche s'arrête dès le premier résultat trouvé, une requête doit se poursuivre pour être stockée sur tous les nœuds redondants nécessaires.

6.2.2. Coût de la réplication

6.2.2.1. Valence

Pour distinguer le degré d'un nœud dans l'arbre d'adressage par rapport à son degré total incluant les liens redondants, nous définissons ce degré total comme étant la *valence* de ce nœud. Ainsi la valence est le nombre total de voisins que ce nœud possède, c'est-à-dire le nombre de nœuds avec lesquels il est directement connecté. Nous cherchons à analyser l'évolution du nombre de chemins alternatifs dans le réseau recouvrant. Nous mesurons la capacité du réseau recouvrant à utiliser des liens redondants n'appartenant pas à l'arbre d'adressage (*shortcuts* en anglais), afin de pallier au problème du remous.

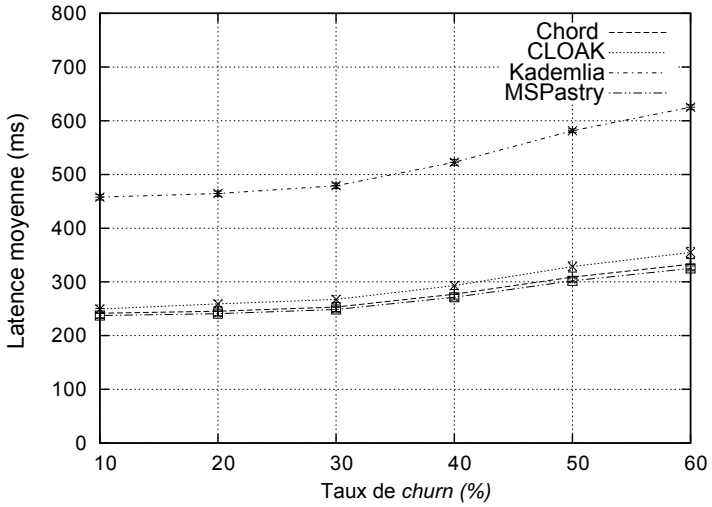

Figure 7. Latence des GET

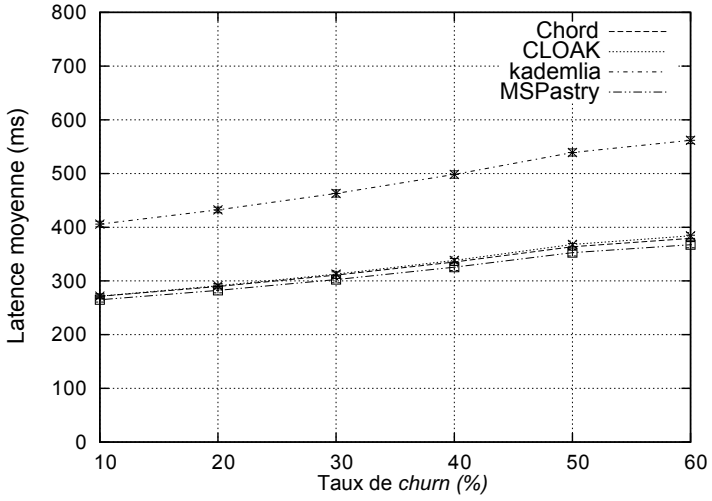

Figure 8. Latence des PUT

La figure 9 présente la répartition de la valence des nœuds en fonction du taux d'attrition lorsque aucune réplication n'est utilisée et au bout de 10 mn de simulation. Elle indique une proportion similaire de nœuds ayant une valence de 1 et 2 quel que soit le taux d'attrition. Ces proportions se différencient pour une valence comprise entre 2 et 8 et se stabilise pour des valences variant de 8 à 256. Ainsi, on note qu'après 10 minutes de simulation, 5 % des nœuds ont une valence supérieure à 8 nœuds, contre respectivement 25 % pour 60 % de taux d'attrition. On notera aussi qu'aucun nœud n'a une valence supérieure à 512. Ces résultats s'expliquent par le fait que notre architecture afin d'être tolérante aux fautes permet aux nœuds de créer plusieurs liens redondants (i.e., liens vers des nœuds qui ne sont ni l'ascendant, ni l'un des descendants du nœud) avec d'autres nœuds du système. La figure 10 indique qu'après 2 heures de simulation, les courbes se différencient nettement selon le taux d'attrition. Ainsi, la proportion des nœuds ayant une valence comprise entre 8 et 64 croît avec le taux d'attrition. Dans cette représentation, on note que 26 % des nœuds ont une valence comprise entre 8 et 64 pour un taux d'attrition de 10 % alors que 84 % des nœuds ayant une valence comprise dans ce même intervalle pour un taux d'attrition de 60 %. Les lignes horizontales entre 8 et 256 (voire moins selon le remous) indiquent que les valences sont soit très petites (< 8), soit très grandes (> 256). Il n'y a pas de nœuds qui ont des valences moyennes étant donné que le pourcentage de répartition ne baisse pas. Cependant plus le remous augmente, plus on trouve des nœuds ayant une valence assez grande (> 64) mais inférieure aux extrêmes. Cela montre que le nombre de nœuds ayant un nombre significatif de liens redondants augmente avec le remous ce qui est un comportement attendu.

La figure 11 présente la répartition de la valence des nœuds en fonction du taux d'attrition lorsque la réplication est utilisée (5 copies en tout par paire) et au bout de 10 mn de simulations. Cette figure est très similaire à la figure 9, la réplication n'ayant donc aucun effet en début de simulation. La figure 12 représente la répartition de la valence dans le réseau recouvrant lorsque la réplication est utilisée après 2 heures de simulation. Comme dans le cas sans réplication associée à chaque donnée stockée, on observe une hausse de la proportion des nœuds ayant une valence entre 8 et 256 lorsque le taux d'attrition augmente. Cependant ici la hausse est beaucoup plus importante. Avec 10 % de remous, 44 % des nœuds ont une valence supérieure ou égale à 8 alors que ce chiffre est de 26 % sans réplication. Avec 60 % de remous, 97 % des nœuds ont une valence supérieure ou égale à 8 alors que ce chiffre est de 84 % sans réplication. Ici le phénomène décrit pour la figure 10 est encore plus accentué. Avec 60 % de remous, 55 % des nœuds ont une valence supérieure à 64. Le réseau se trouve donc fortement maillé afin d'assurer une fiabilité suffisante du routage (66 % de requêtes réussies à un tel taux de remous).

6.2.2.2. Quantité de trafic

Nous évaluons ici le nombre de messages de requêtes de recherche qui traversent chaque nœud du réseau recouvrant. Nous avons tracé la fonction de répartition complémentaire (communément appelée CCDF en anglais) pour chaque taux d'attrition afin d'observer l'impact du remous sur la répartition du trafic. Le nombre de répliques

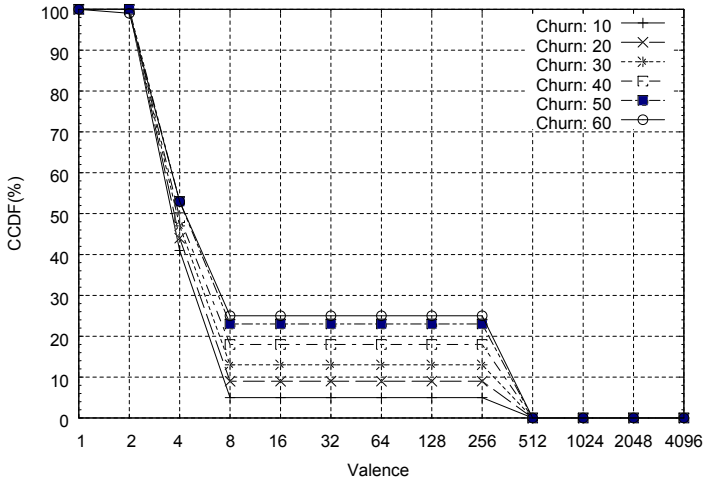

Figure 9. Valence vs taux d'attrition sans réplication après 10 min

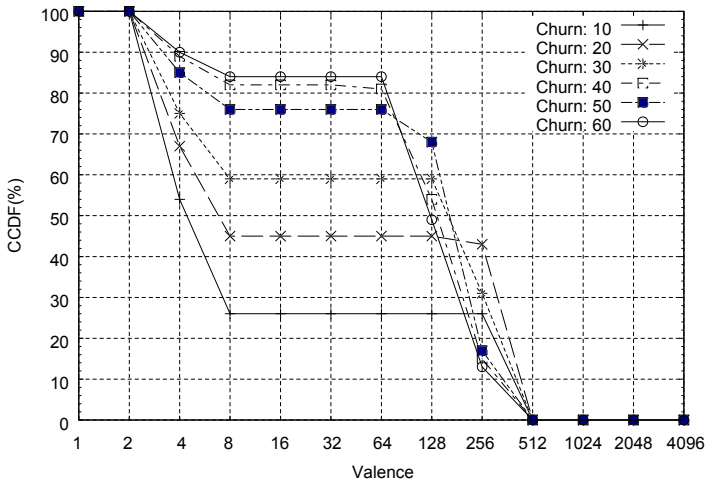

Figure 10. Valence vs taux d'attrition sans réplication après 2h

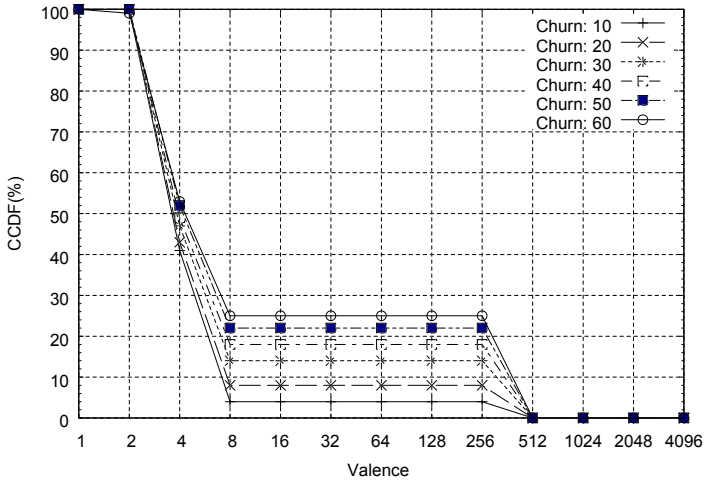

Figure 11. Valence vs taux d'attrition avec réplication après 10 min

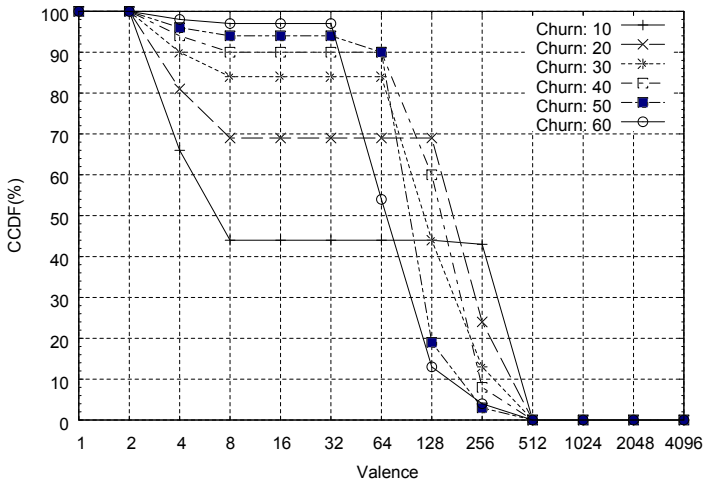

Figure 12. Valence vs taux d'attrition avec réplication après 2h

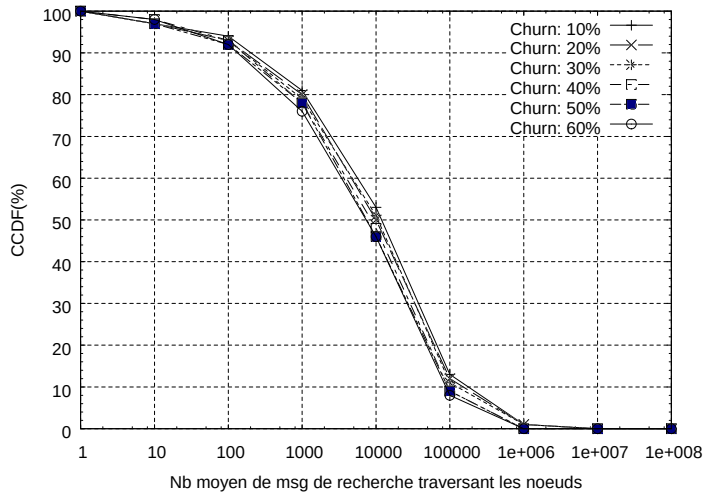

Figure 13. Quantité de trafic vs taux d'attrition après 10 min

est fixé à 5. La figure 13 montre une légère différenciation selon le taux d'attrition mais globalement le remous a peu d'impact en début de simulation. On note par exemple que 45 % des nœuds sont traversés par moins de 10 000 messages à 60 % de remous contre 55 % à 10 % de remous. Ce résultat est normal car plus il y a de remous, plus les chemins sont raccourcis (dûs aux pertes de messages) et plus des chemins alternatifs sont empruntés dans le réseau recouvrant entrainant ainsi une faible congestion dans le système. Nous montrons à travers ces résultats la flexibilité de notre mécanisme de recherche qui malgré un fort taux d'attrition enrégistre une certaine fluidité des messages dans le système. La figure 14 montre la répartition du trafic après 2 heures de simulation. Par rapport à la figure 13 on peut observer deux phénomènes. D'abord, les courbes se situent toutes vers la droite du graphique ce qui illustre une augmentation du trafic vu par les nœuds. De plus, les courbes se différencient notablement selon le taux d'attrition entre 100 et 10^7 messages traversés et on observe dans cette zone une réduction notable du nombre de nœuds subissant un trafic donné lorsque le taux d'attrition augmente.

6.2.2.3. Taille des tables de stockage

Notre objectif ici est d'étudier l'évolution de la taille des tables de stockage dans les nœuds du réseau recouvrant soumis à des taux d'attrition variant de 10 % à 60 % avec un nombre de répliques fixé à 5. Nous étudions aussi l'incidence de notre stratégie de réplication sur la taille de ces tables. La figure 15 présente la CCDF de la taille des tables de stockage après 10 minutes de simulation selon le taux d'attrition.

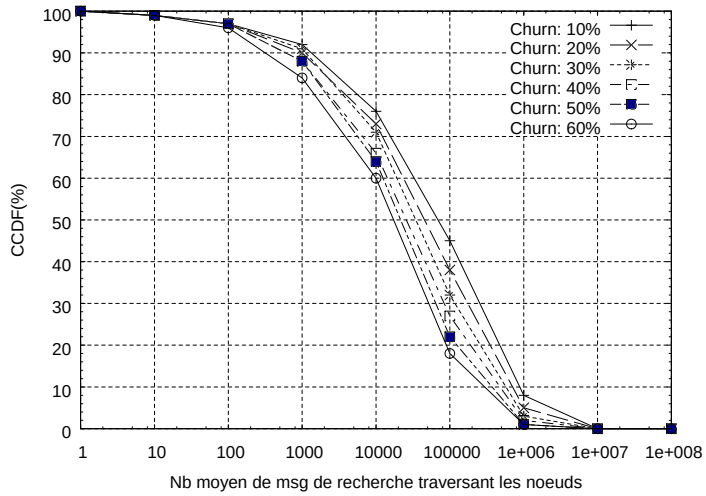

Figure 14. Quantité de trafic vs taux d'attrition après 2h

La distribution des tailles des tables de stockage est peu différenciée pour les taux d'attrition de 10 % à 60 %. La figure 16 montre que cette tendance se modifie avec la durée de simulation. En effet, après 2 heures de simulation les courbes montrent une inflation très significative, avec des écarts de 15 % environ pour les tailles comprises entre 16 et 64 par rapport au début de la simulation. De plus, elles se différencient selon le taux d'attrition surtout pour les nœuds ayant des tailles moyennes entre 4 et 64. La quantité des stockeurs ayant une table de taille supérieure à 8 est de 84 % pour un taux d'attrition de 10 % alors qu'elle est de 60 % pour un taux d'attrition de 60 %. Cette différence devient moins perceptible à partir d'une taille supérieure à 256. Ces résultats indiquent que plus il y a de remous dans le système, plus il y a de perte des requêtes de stockage, et moins les stockeurs sont accessibles stockant donc moins de données. Ainsi, pour 80 % des nœuds, une variation du taux d'attrition de 10 % à 60 % induit une diminution de moitié de la taille des tables de stockage. Cette situation montre donc que lorsque le taux d'attrition augmente, la taille moyenne des tables de stockage diminue.

Nous observons maintenant l'impact du nombre de répliques sur la répartition des tailles des tables de stockage pour les cas de taux d'attrition de 10 % et de 60 % après 2 heures de simulation. La courbe 17 à 10 % montre que les courbes se différencient pour des tailles de 16 à 256, une augmentation des répliques entrainant logiquement une augmentation du nombre de nœuds ayant une taille donnée. La courbe 18 à 60 % montre que les courbes se différencient pour des tailles beaucoup plus faibles que précédemment de 4 à 64. De plus les courbes de répartition sont plus basses pour

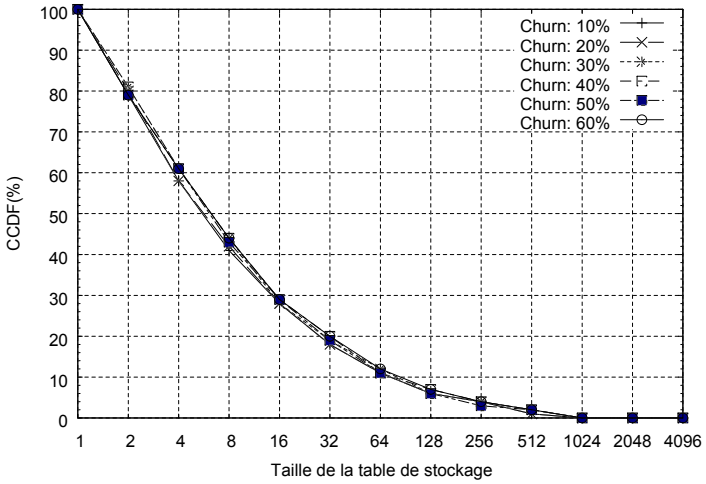

Figure 15. Taille des tables vs taux d'attrition après 10 min

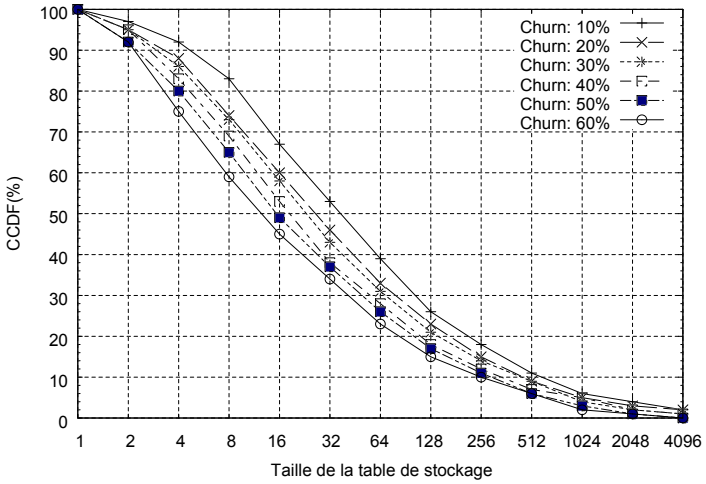

Figure 16. Taille des tables vs taux d'attrition après 2h

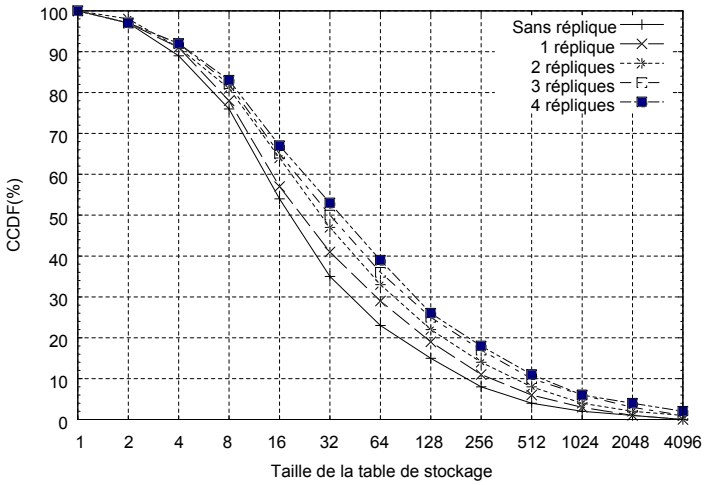

Figure 17. Taille des tables vs nb de répliques à 10 % de remous

une taille donnée car comme nous l'avons vu ci-dessus, une augmentation du remous entraîne une réduction importante de la taille des tables.

Tous ces résultats montrent que d'une part notre THD affiche des performances similaires aux THD existantes en termes de taux de succès du routage, de distances et de temps de latence. D'autre part, on constate globalement que les répliques se répartissent assez bien dans la THD et qu'elles n'impactent pas trop négativement le nombre de messages et les tailles des tables de stockage.

7. Conclusion

Fournir un service d'annuaire par THD aux applications distribuées est une tâche difficile. Grâce à ses propriétés de tessellation, le plan hyperbolique au travers du modèle du disque de Poincaré est adéquat pour attribuer des coordonnées virtuelles aux nœuds participant à un réseau recouvrant. Nous avons montré dans cet article qu'en définissant une fonction de correspondance appropriée, il est facile de construire et de maintenir une THD sur un tel réseau. Nous avons proposé un système de THD nommé CLOAK qui est capable de passer à l'échelle tout en restant fiable face aux remous. En effet, notre mécanisme de réplication nous permet d'augmenter considérablement le taux de succès des requêtes de recherche tout en distribuant les répliques de façon homogène.

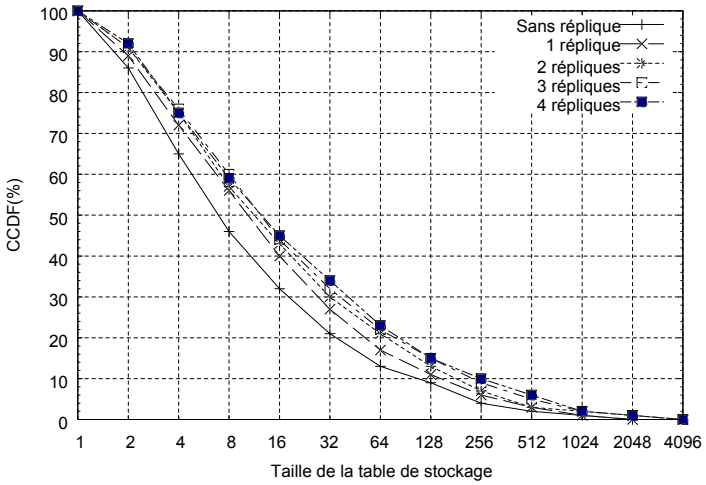

Figure 18. Taille des tables vs nb de répliques à 60 % de remous

Notre analyse a montré que notre solution possède des coûts ayant des ordres de grandeur similaires à ceux des THD existantes. Nos résultats de simulation ont confirmé notre analyse et ont montré que le taux de succès, le nombre de sauts et la latence des requêtes de recherche et de stockage sont similaires à ceux de Chord, MSPastry et Kademlia qui sont trois systèmes de THD très populaires. L'observation de la valence indique que notre système ne nécessite pas beaucoup de liens redondants pour garantir un taux de succès élevé des requêtes de recherche et de stockage en contexte de fort remous.

L'avantage clé de notre solution est qu'elle requiert un coût mémoire indépendant de la taille du réseau recouvrant car elle ne nécessite pas dans chaque nœud le maintien d'une table de routage mais seulement le maintien d'une liste des voisins. Cette liste est en général de très petite taille, en particulier dans les réseaux de type *petit-monde*. Un autre avantage de notre solution est que les nœuds peuvent se connecter librement les uns aux autres, tandis que dans les autres systèmes de THD tels que Chord, les nœuds doivent s'insérer dans la THD en se connectant à d'autres nœuds prédéterminés en fonction de leurs adresses IP. Etant donné ces deux avantages, et ayant des performances identiques par ailleurs, nous pensons que notre THD peut être une alternative supérieure aux THD existantes.

Nos travaux futurs vont consister à implémenter et évaluer d'autres mécanismes permettant de maintenir l'arbre d'adressage en évitant les ré-adressages ou bien en réduisant le coût de leur complexité. Nous souhaitons aussi définir et implémenter des niveaux de THD supplémentaires sur notre THD existante afin de pouvoir utiliser

des indirections. Par exemple un nom de groupe se traduirait par un ensemble de noms d'individus, qui se traduirait par un ensemble de noms de terminaux qui se traduirait enfin par un ensemble d'adresses du réseau recouvrant. Il faudra donc faire des requêtes successives à différents niveaux afin de récuperer l'information souhaitée (i.e., les adresses dans notre exemple). Cette faculté permettra à notre système de fournir des services innovants de type routage au plus proche (*anycast* en anglais), routage par contenu, etc.

Références

Balakrishnan H., Kaashoek M. F., Karger D., Morris R., Stoica I. (2003). Looking up data in p2p systems. *Communications of the ACM*, vol. 46, n° 2, p. 43–48.

Beardon A. F., Minda D. (2006). The hyperbolic metric and geometric function theory. In *International workshop on quasiconformal mappings and their applications.*

Cassagnes C., Tiendrebeogo T., Bromberg D., Magoni D. (2011). Overlay addressing and routing system based on hyperbolic geometry. In *Proceedings of the IEEE Symposium on Computers and Communications.*

Castro M., Costa M., Rowstron A. (2003). Performance and dependability of structured peer-to-peer overlays. In *Proceedings of the International Conference on Dependable Systems and Networks.*

Cohen R., Havlin S. (2003). Scale-free networks are ultrasmall. *Phys. Rev. Lett.*, vol. 90, n° 5.

Coxeter H. (1954). Regular honeycombs in hyperbolic space. In *International congress of mathematicians*, vol. 3, p. 155-169.

Coxeter H., Whitrow G. (1950). World-structure and non-euclidean honeycombs. *Proceedings A of the Royal Mathematical Society of London*, vol. 201, p. 417-437.

Cvetkovski A., Crovella M. (2009). Hyperbolic embedding and routing for dynamic graphs. In *Proceedings of infocom.*

Gyarmati L., Trinh T. A. (s. d.). Scafida: a scale-free network inspired data center architecture. *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, n° 5, p. 4–12.

Kleinberg R. (2007). Geographic routing using hyperbolic space. In *Proc. of the 26th infocom conference*, p. 1902-1909.

Lua E. K., Crowcroft J., Pias M., Sharma R., Lim S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, vol. 7, p. 72-93.

Margenstern M. (2002). On the infinigons of the hyperbolic plane, a combinatorial approach. *Fundam. Inf.*, vol. 56, n° 3, p. 255–272. http://dl.acm.org/citation.cfm?id=964787.964790

Maymounkov P., Mazières D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, p. 53–65.

Montresor A., Jelasity M. (2009). Peersim: A scalable p2p simulator. In *Proc. of the 9th International Conference on Peer-to-Peer*, p. 99-100.

Naor M., Wieder U. (2003). Novel architectures for p2p applications: the continuous-discrete approach. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, p. 50–59.

Papadopoulos F., Krioukov D., Boguñá M., Vahdat A. (2010, March). Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *Proceedings of the ieee infocom conference.* San Diego, CA.

Ratnasamy S., Francis P., Handley M., Karp R., Shenker S. (2001). A scalable content-addressable network. In *Proceedings of the acm sigcomm conference*, p. 161–172.

Rowstron A., Druschel P. (2001). Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Ifip/acm international conference on distributed systems platforms (middleware)*, p. 329-350.

Stoica I., Morris R., Karger D., Kaashoek F., Balakrishnan H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM Conference*, p. 149–160.

Tiendrebeogo T., Ahmat D., Magoni D. (2012). Reliable and scalable distributed hash tables harnessing hyperbolic coordinates. In *Proceedings of the 5th IFIP International Conference on New Technologies, Mobility and Security.*

Zhao B. Y., Huang L., Stribling J., Rhea S. C., Joseph A. D., Kubiatowicz J. D. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, vol. 22, p. 41-53.