KADEMLIA

INTRODUCTION :

Kademlia(alias Kad) est un réseau de recouvrement créé pour décentraliser les autres réseaux d'échange de fichiers Peer-to-Peer. Le protocole précise la structure du réseau Kademlia, les communications entre les nœuds et l'échange d'information. Les nœuds communiquent grâce au protocole UDP.

FONCTIONNEMENT :

A l'intérieur d'un réseau existant, Kademlia crée un nouveau réseau, à l'intérieur duquel chaque nœud est identifié par un numéro d'identification, un ID (nombre binaire à 160 bits.)
Passée une phase d'amorçage consistant à contacter un nœud du réseau puis à obtenir un ID, un algorithme calcule la « distance » entre deux nœuds, et interroge plusieurs nœuds suivant cet algorithme afin de trouver l'information recherchée. Cet algorithme utilise une notion de distance entre deux nœuds, calculée grâce à une opération mathématique (OU exclusif, aussi appelée XOR) délivrant un résultat sous forme de nombre entier : la « distance ». Cette dernière n'a rien à voir avec la situation géographique des participants, mais modélise la distance à l'intérieur de la chaîne des ID. Il peut donc arriver qu'un nœud en Allemagne et un nœud en Australie soient « voisins ».
Une information dans Kademlia est conservée dans des « valeurs », chaque valeur étant jointe à une « clé ». On dit de Kademlia qu'il est un réseau <valeur,clé>.
Lors de la recherche d'une certaine clé, l'algorithme explore récursivement le réseau au cours de différentes étapes, chaque étape s'approchant plus près de la clé recherchée, jusqu'à ce que le nœud contacté retourne la valeur, ou que plus aucun nœud ne soit trouvé. La taille du réseau n'influe pas énormément sur le nombre de nœuds contactés durant la recherche ; si le nombre de participants du réseau double, alors le nœud de l'utilisateur doit demander l'information à un seul nœud de plus.
D'autres avantages sont inhérents à une structure décentralisée, augmentant par exemple la résistance à une attaque de déni de service. Même si toute une rangée de nœuds est submergée, cela n'aura que des effets limités sur la disponibilité du réseau, qui « recoudra » le réseau autour de ces trous.

tables de hachage :

Une fonction de hachage est une fonction qui est « statistiquement injective » : si x != y alors probablement h(x) != h(y) Une table de hachage est une structure où les données sont stockées selon la valeur d'une fonction de hachage:

0	1	h(x)	3	h(y)=h(z)	h(t)	6	7
		x		y,z	t		

DHT:

Une table de hachage distribuée (DHT) est une table de hachage implémentée par un ensemble de pairs communiquant à travers un réseau. Dans une DHT, la valeur de la fonction de hachage détermine le pair où la donnée est stockée.

Identificateur:

Un identificateur est un entier de 160 bits.
A = 0 111 011 100 010· · ·

Les identificateurs :

– identifient les nœuds;
– servent de clés.

Typiquement,

– les identificateurs des nœuds sont tirés au hasard ;
– les clés sont les hashes SHA-1 des « vraies » clés.

La métrique XOR

distance entre id1 et id2: d(id1,id2)= id1 XOR id2
exemple: si ID codé sur 3 bits:

$$d(1,4) == d(001_2, 100_2)$$
$$= 001_2 \text{ XOR } 100_2$$
$$= 001_2$$
$$= 5$$

Routage en XOR:

Connexion entre les nœuds:

Chaque nœud A dans Kademlia maintient une table de routage composée d'ensembles nommés buckets. Les buckets sont classés par ordre d'éloignement selon la métrique XOR.
Un bucket regroupe **k** nœuds dont les distances sont comprises entre 2^i et 2^{i+1}.
Le degré moyen d'un nœud est donc O ($k*\log_{2^b} N$).

Tous les nœuds du même bucket sont à la même distance du nœud A.
Chaque bucket correspond donc à un sous arbre et contient **k** voisins classés selon leur ancienneté dans le système.
A l'origine, Kademlia ne maintient pas des voisins séquentiels.
Cependant, cette information peut être tirée directement de sa table de routage .
En effet, les premiers buckets contiennent les identifiants des nœuds les plus proches selon la métrique XOR.
Ces nœuds nommés sibling constituent l'ensemble des voisins séquentiels dans S/Kademlia

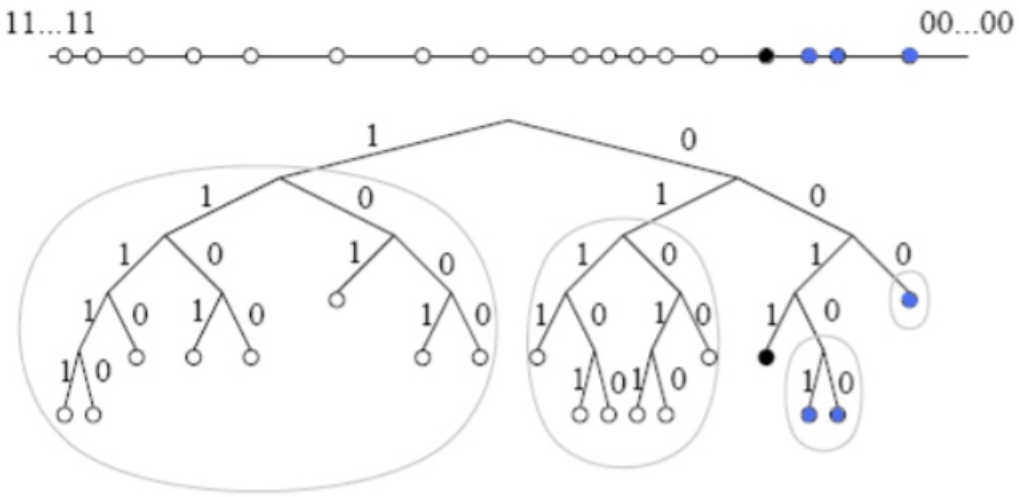

figure 1: Voisinage du nœud 0011. Les cercles en bleu représentent voisins sibling du nœud0011.

Routage :

Lorsqu'un nœud S cherche une clé K, il recherche itérativement les K nœuds les plus proches de K, S commence par identifier et envoyer sa requête à ses K voisins les plus proches parmi ses buckets. Ces voisins vont répondre à leur tour en renvoyant leurs K nœuds les plus proches, présents dans leurs tables de routage. Le nœud S met à jour sa table de routage puis renvoie sa requête aux K nouveaux nœuds les plus proches. La même procédure se répète jusqu'à ce que le nœud S ait contacté tous les autres nœuds renvoyés dans les réponses, et ainsi ait récupéré les K nœuds dont l'ID est le plus proche de la clé K. Cet algorithme a pour but d'augmenter le préfixe commun entre l'identifiant de la clé et celui des nœuds contactés à chaque saut. Le nœud qui effectue une recherche contacte donc un à un tous les nœuds intermédiaires de la route jusqu'à atteindre la destination. Ceux ci lui fournissent en retour une liste de contacts possibles pour le saut suivant (Figure 2). Le nombre de sauts moyen nécessaires pour une recherche de clé sur un alphabet de 2b chiffres est O $\log_{(2^b)}$N .

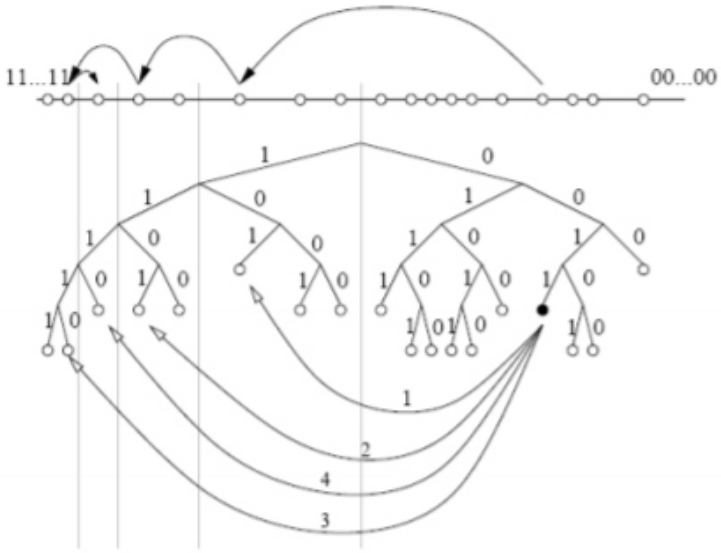

figure 2:Exemple de route Kademlia : le nœud 0011 localise le nœud 1110

Maintenance :

la maintenance dans Kademlia est implicite. Lorsqu'un nœud reçoit un message find_node, il met à jour le bucket approprié. Si le nœud expéditeur se trouve déjà dans ce bucket, il est déplacé en bas de la liste. Sinon, si le bucket n'est pas plein, la nouvelle entrée est insérée. Si le bucket est plein, le nœud A émet un PING vers le voisin le plus ancien dans ce bucket. Si ce dernier répond au PING, il est alors déplacé en fin de liste et la nouvelle entrée est ignorée. Kademlia conserve donc dans sa table les nœuds les plus anciens dans le réseau.

For more information:
http://people.rennes.inria.fr/Julien.Stainer/docs-akermarrec/maymounkov-kademlia-lncs.pdf