Stage ETE 2015

Description du système

F-PHT
Un système d'index de filtres de Bloom pour la recherche d'information par mots clés

Réalisé par **DOAN** Cao Sang
Encadrant : M. **MAKPANGOU** Mesaac, Regal

1 Juillet 2015

Table des matières

Chapitre 1

Vue globale

1.1 Prefix Hash Tree (PHT)

Un arbre préfixe est un arbre numérique ordonné qui est utilisé pour stocker une table associative où les clés sont généralement des chaînes de caractères. Contrairement à un arbre binaire de recherche, aucun nœud dans le trie ne stocke la chaîne à laquelle il est associé. C'est la position du nœud dans l'arbre qui détermine la chaîne correspondante [1].

Pour tout nœud, ses descendants ont en commun le même préfixe. La racine est associée à la chaîne vide. Des valeurs ne sont pas attribuées à chaque nœud, mais uniquement aux feuilles et à certains nœuds internes se trouvant à une position qui désigne l'intégralité d'une chaîne correspondante à une clé.

Pour faire une recherche d'une valeur associée à une clé, au départ, on se situe à la racine de l'arbre, en prenant le premier élément de la clé de la requête, on trouve le chemin étiqueté par cet élément, s'il n'existe pas, on est sûr que cette clé n'est pas dans l'arbre. Dès que l'on trouve le chemin, on arrive sur le bon nœud et continue en prenant le deuxième élément de la clé de requête, on applique cette méthode jusqu'à quand on trouve cette clé et se termine sur une feuille.

1. Wikipédia

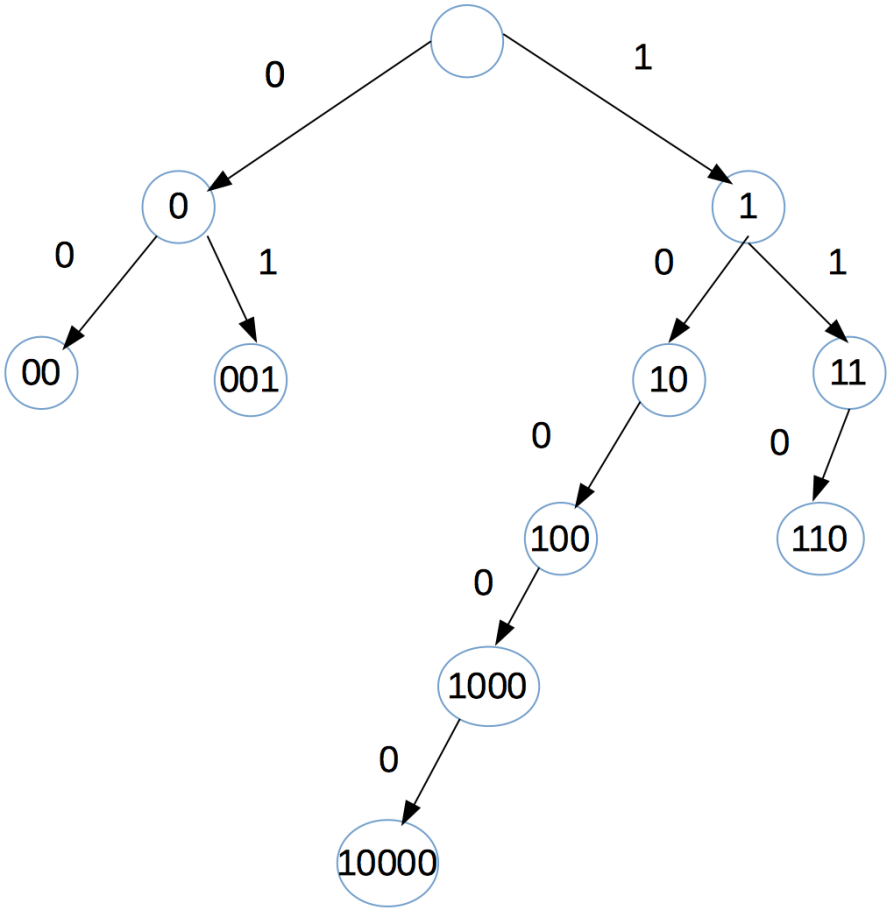

FIGURE 1.1 – Arbre de préfixe

1.2 F-PHT

F-PHT est un arbre préfixe de "multibit". Pour stocker une clé, on suit le chemin étiqueté par les fragments successifs de cette clé (dans l'ordre de leur rang) jusqu'à ce qu'on arrive sur une feuille pouvant stocker cette clé.

Un fragment est un morceau d'une clé (sous-ensemble de bits). Si on considère les clés de taille m, le système découpe chaque clé en f fragments de taille identique. Par convention, ces fragments sont numérotés de *0* à *f-1*.

15															0
1	0	0	0	1	1	0	1	0	0	0	0	1	0	1	0

TABLE 1.1 – Exemple le filtre de Bloom

La table 1.1 représente une clé de taille $m = 16$ bits, elle est ensuite découpée en $f = 4$ fragments. Le premier fragment est la suite de bits *"1000"* qui se trouve les plus à gauche de cette clé. Le dernier est *"1010"*.

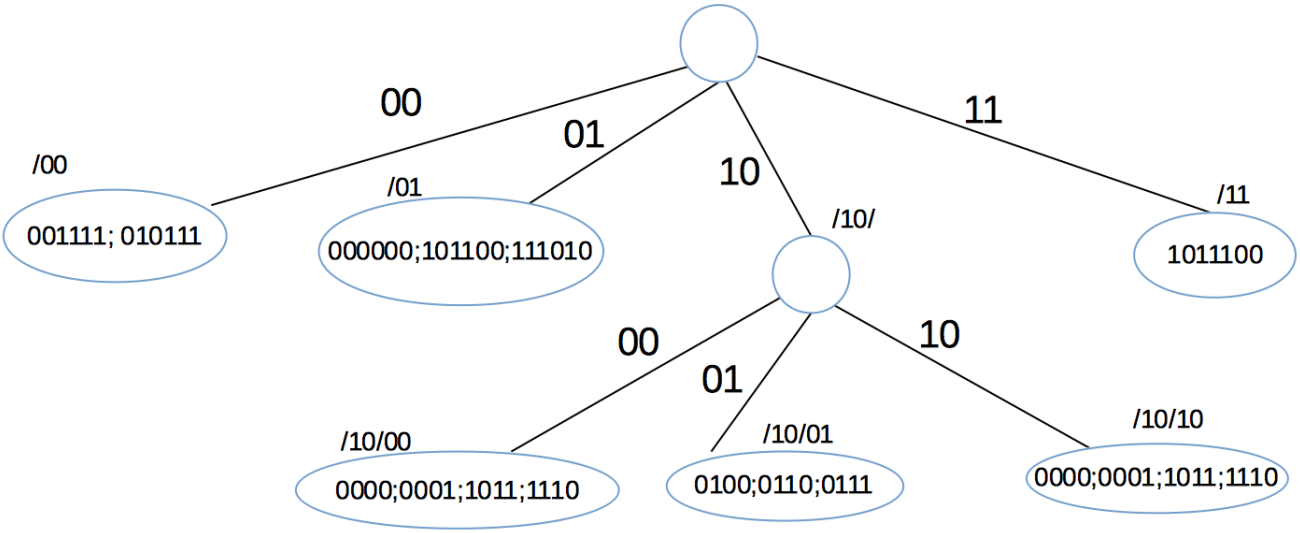

FIGURE 1.2 – Arbre F-PHT

La figure 1.2 représente l'arbre F-PHT qui stocke une clé de taille $m = 8$ bits, la taille d'un fragment est de 2 bits. Au dessus de chaque nœud (interne ou feuille) se trouve son identifiant unique et il caractérise également le chemin unique conduit vers ce nœud.

Chapitre 2

Système d'indexation

2.1 Objectif

Les clés indexées sont des filtres de Bloom de taille m. L'objectif du système est d'une part de stocker les filtres de Bloom donnés par les utilisateurs, d'autre part de permettre de rechercher tous les filtres de Bloom (stockés dans le système) qui sont sur-ensemble d'un filtre de Bloom caractérisant des mots clés d'une requête.

2.2 Architecture d'un système d'indexation

Le système d'indexation est sur un ensemble de serveurs d'index répartis sur plusieurs machines. Chaque serveurs gère un ou plusieurs nœuds de ce système.

2.3 API du système d'indexation

Chapitre 3

Fonctionnement du système

3.1 Données de chaque nœud dans l'arbre

3.2 Protocole

Ce protocole porte des actions entre le client et le système comme : add, search, remove.

Lors que le système reçoit un filtre de Bloom à indexer, il appelle simplement cette méthode **add**. Cette méthode envoie un message d'ajout à la racine. Ensuite, la racine route ce message jusqu'à la feuille qui s'occupe ce filtre grâce à la valeur de fragment de rang i.

3.2.1 Cas d'un nœud feuille

Lors qu'une feuille reçoit la demande **add** d'un filtre de Bloom, elle cherche l'indice où le met dans la table *LocalRoute*. Si l'entrée est pleine, pour chaque entrée de la table, elle demande le système de créer un nœud correspondant, et ajoute son identifiant dans la nouvelle table *LocalRoute*. Cette nouvelle table remplace l'autre après les transferts des données vers les fils convenables. Si l'entrée est libre, la feuille ajoute simplement. Elle devient un nœud.

3.2.2 Cas d'un nœud

Un nœud reçoit la demande d'ajout d'un filtre de Bloom est simple, en regardant dans sa table *LocalRoute*, trouve une entrée qui satisfait la requête et la route vers ce fils.

3.3 Recherche des filtres de Bloom

Cette commande **search** est aussi envoyée vers la racine, par contre, elle renvoie cette requête vers tous les fils qui contient.

3.3.1 Cas d'un nœud feuille

Une feuille reçoit la requête, elle compare la requête avec les filtres dans sa table *LocalRoute*, et collecte tous les filtres qui contiennent la requête. Une fois, tous les filtres satisfaisants sont renvoyés vers la racine.

3.3.2 Cas d'un nœud

Un nœud compare aussi la requête avec les entrées dans *LocalRoute*, toutes les entrées qui contiennent la requête sont sélectionnées pour récupérer tous les nœuds afin de leur renvoyer la requête.

3.4 Suppression d'un filtre de Bloom

Cette action **remove** fonctionne exactement comme l'ajout d'un filtre de Bloom, la racine route la requête de suppression vers le fils qui gère cette requête. Dès qu'elle arrive à la feuille, cette feuille cherche dans sa propre table *LocalRoute* une entrée correspondante et la supprime. Si cette table devient vide après la suppression, elle demande de la supprimer au système, en plus, le système se charge de notifier son père la disparition de son fils pour qu'il puisse supprimer l'entrée de son fils dans sa table de routage *LocalRoute*. Du coup, si sa propre table *LocalRoute* devient vide après la suppression de son fils, il aussi demande de se supprimer au système et notifie son père. Ces opérations continuent jusqu'à quand il n'y a plus de table *LocalRoute* vide.