Projet SAR

RAPPORT

Recherche de filtres de Bloom similaires
Application à la recherche par mots clés basée sur une DHT

Réalisé par **NDOMBI TSHISUNGU** Christian & **DOAN** Cao Sang
Encadre : M. **MAKPANGOU** Mesaac, Regal

26 Mars 2015

Table des matières

1 Introduction **2**

2 Projet **4**
2.1 Marché 4
2.2 Contexte et objectifs 4
2.3 Enoncé du besoin 4
2.3.1 Stockage et indexation des publications 5
2.3.2 Recherche de contenus 5

3 Filtre de Bloom **6**

Chapitre 1

Introduction

Dans ce sujet, on abordera sur les recherches par mots clés. Une fois, on tape un mot ou une chaine de mots, l'application cherche dans la base de données pour trouver la liste des documents qui contient les descriptions correspondantes avec ces mots clés.

Dans la base de données qui contient des grosses masses de donées, les index sont basés sur les descriptions des documents. Chaque mot est une clé, chaque clé contient une liste des documents ayant ce mot. Au contraire à la liste normale, le mot clé est le nom du document et il contient la liste des mots qui se trouvent dans la description de ce document. Par exemple, le mot *PSAR : le projet pour les étudiants du M1*, normalement, le mot clé est *PSAR*, donc il contient sa description *le projet pour les étudiants du M1*, mais dans cette liste, les mots clés sont *le, projet, pour, les, étudiants, du, M1* et chaque mot contient le mot *PSAR*. On s'appelle la *liste inversée*.

Du coup, on utilise les tables de hachage distribuées (DHT) pour stocker les listes inversées. L'avantage d'utilisation de ces tables est que la complexité est égale à $\theta(1)$ car on trouve directement l'élément que l'on recherche. La statégie de distribution des listes inversées la plus utilisée (appelée partionnement vertical) consiste à associer à chaque terme une clé qui désigne le pair qu stocke la liste inversée associée à ce terme. Une fois que l'on lance la recherche sur les documents qui sont satisfaits à l'ensemble des mots que l'on a saisi, le système doit chercher pour chaque mot dans cet ensemble la liste des documents qui le contiennent, et puis fait l'intersection des listes inversées retrouvées.

Dans les systèmes pair-à-pair (P2P), chaque nœud contient un ou plusieurs mots clés, une fois qu'on cherche un mot, le système va demander le nœud où se trouve ces mots clés. Le problème est que si l'on cherche un ensemble de mots, il doit obligatoirement envoyer plusieurs requêtes à plusieurs nœuds pour récupérer les listes des documents et les faire l'intersection. Si on a un

grand nombre de machines qui cherchent une quantité importante de mots clés, le système sera saturé. Et aussi, le problème se pose sur un ou quelques nœuds précises, et les autres sont libres dans la plupart de temps. Donc, le coût de la recherche est d'autant plus élevé que la requête est précise et le déséquilibrage de charges des noeuds d'indexation du fait de la non-uniformité de la popularité des termes, ce sont des problèmes qu'on va rechercher la solution dans ce projet.

Chapitre 2

Projet

2.1 Marché

Il existe actuellement plusieurs moteurs de recherche par mots clés sur le marché. Il est toujours nécessaire d'améliorer les recherches en minimisant le temps de réponse et le coût de recherche, d'autant plus que les besoins de recherche deviennent très importants lorsqu'on exploite des bases de données de plus en plus grande et complexe. L'application que nous développerons doit répondre explicitement à ces exigences.

2.2 Contexte et objectifs

Comme l'application FreeCore[1], qui utilise le filtre de Bloom pour faciliter la recherche par mot clé, le but de notre projet est d'explorer d'autres solutions basées sur une recherche de filtre de Bloom similaires. Chaque filtre de Bloom sera vu comme un point dans un espace à n dimensions plutôt qu'une concaténation de n mots binaires. On détermine ensuite une relation de proximité et on exploite les algorithmes de recherche des filtres similaires pour réduire l'ensemble de filtres à examiner.

2.3 Enoncé du besoin

Comme FreeCore, l'application exploitera les propriétés des filtres de Bloom. Il permettra d'une part, de stocker les publications dans un fichier VA_file (put). Et d'autre part, d'effectuer des recherches de contenus par mots clés (search).

2.3.1 Stockage et indexation des publications

L'application permet de rechercher des publications qui contiennent tous les mots clés de la requête de façon optimale en utilisant le filtre de Bloom. Elle doit :

- créer un filtre de Bloom correspondant à la description de la publication,
- créer un vecteur de n dimensions.
- créer les lieux où on stocke les différents documents,
- à partir d'un fichier de test, classer et indexer les documents dans les lieux correspondants,
- ajouter un nouveau document à partir d'un filtre de Bloom,

2.3.2 Recherche de contenus

Pour la recherche l'application créera un Filtre de Bloom des mots clés, représentants notre critère de recherche, c'est-à-dire les mots clés et devra :

- recherche un document à partir du filtre de Bloom,
- utilise le vecteur approximatif pour indexer et rechercher.

En outre, l'application doit assurer les services suivants :

- afficher les messages d'erreurs, s'il en existe,
- afficher les états de l'application,
- interagir avec l'utilisateur,
- faciliter les tests en utilisant les fichiers de test ou en utilisant les entrées saisit par l'utilisateur.

Note : Nous ne traiterons pas les erreurs possibles obtenues en cas de faute de frappe ou de faute d'orthographe, ni la différence de genre et de nombre des mots clés. Qui pourront faire objet d'une suite de ce travail.

Chapitre 3

Filtre de Bloom

Un filtre de Bloom est une structure de données probabiliste compacte inventée par Burton Howard Bloom en 1970. L'avantage d'utilisation de filtre de Bloom est que cette technique nous permet de savoir avec certitude que l'élément n'est pas présenté dans l'ensemble d'élément, c'est-à-dire il ne faut pas y avoir de faux négatif mais il peut y avoir des faux positifs. On ne peut savoir qu'avec une certaine probabilité, l'élément peut être présent dans l'ensemble. Ce qui réduit d'une manière considérable les entrées lorsqu'on fait une recherche dans une masse de données. En plus, la taille de filtre est fixe et indépendante du nombre d'éléments contenus, par contre, plus déléments plus de faux positifs.

En réalité, le filtre de Bloom a un structure très simple, un tableau B de m bits associé à i fonctions de hachage h_i, $0 \leq i \leq m-1$ permettant de mapper tout élément de l'ensemble à une des m cases du tableau. Ces fonctions de hachage ont une répartition uniforme des éléments de l'ensemble sur le tableau, et évidemment, doivent avoir une répartition différente. Au départ, le filtre représente un ensemble vide, et toutes les cases sont à 0.

Pour chaque clé k à ajouter à B, au lieu de se contenter de mettre à vrai la case B. $h(k)$ avec une seule fonction de hachage comme on le fait classiquement, on va mettre à vrai les m cases B. $h_i(k)$. Le principe étant que la probabilité que deux clés différentes aient les mêmes m valeurs pour leurs fonctions de hachage est faible.

Pour savoir si une clé est présente, on s'assura que les m cases de la table B correspondant aux valeurs des m fonctions de hachage sont posisitionnées à 1. Ce filtre est utile pour déterminer si un élément ne fait pas partie d'un ensemble, afin par exemple pour définir rapidement d'un traitement lourd lors de vérifier qu'une personne ne fasse pas partie d'une liste noire : d'abord, une vérification rapide avec le filtre de Bloom, puis en cas de potentiel posistif, un vérification plus certaine avec la comparaison dans la base de données.

Le filtre de Bloom a quelques désavantages comme par exemple pour supprimer un élément dans l'ensemble de données, il nous faut reconstruire le filtre, en plus les faux posisitifs augmentent avec le nombre d'éléments présents dans l'ensemble. Nombreuses solutions utilisent des techniques probabilistes pour réduire le traitement d'information et leur coût.

Le filtre de Bloom et leurs variantes sont largement utilisés dans divers systèmes distribués. Plusiseurs recherches récentes et beaucoup de nouveaux algorithmes ont été proposés pour les systèmes distribués qui sont directement ou indirectement basés sur Bloom filtres[2].

Algorithme 1 *Insertion dans le filtre de Bloom*

Entrée : x objet à insérer dans le filtre de Bloom B
Fonction : $insert(x)$
Retourne : $\emptyset$

---

for $i = 0...k-1$ **do**
    $j \leftarrow h_i(x)$
    if $B_j == 0$ **then**
        $B_j \leftarrow 1$
    end
end

Par exemple, supposons que nous souhaitions ajouter la clé "computer" dans la table B de taille 16 bits, que nous ayons 4 fonctions de hachage $h_i, 0 \leq i < 4$ et que $h_0(\text{"}computer\text{"}) = 3$, $h_1(\text{"}computer\text{"}) = 8$, $h_2(\text{"}computer\text{"}) = 15$, $h_3(\text{"}computer\text{"}) = 10$, $h_4(\text{"}computer\text{"}) = 11$. Donc, l'état de la table B après l'insertion sera :

1	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0

TABLE 3.1 – Exemple filtre de Bloom

Algorithme 2 *Test d'appartenance d'un élément dans le filtre*

Entrée : x objet à tester dans le filtre de Bloom B
Fonction : $ismember(x)$
Retourne : *bool*

$m \leftarrow true$
$i \leftarrow 0$
while m && $i \leq k - 1$ **do**
    $j \leftarrow h_i(x)$
    if $B_j == 0$ **then**
        $m \leftarrow false$
    end
    $i \leftarrow i + 1$
end return m

Bibliographie

[1] Mesaac Makpangou, Bassirou Ngom, Samba Ndiaye : ***Freecore : Un substrat d'indexation des filtres de Bloom fragmentés pour la recherche par mots clés***. ComPAS'2014, Apr 2014, Neuchâtel, Switzerland

[2] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz : ***Theory and Practice of Bloom Filters for Distributed Systems***. Communications Surveys & Tutorials, IEEE. pp. 131 – 155. Fevrier. 2012