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

NDOMBI TSHISUNGU Christian & **DOAN** Cao Sang
Encadrant: M. **MAKPANGOU** Mesaac, Regal

UPMC

2 Mai 2015

Table de contents

1. Présentation
   - Problème classique
   - Nos solutions
2. Filtre de Bloom
   - Insertion
   - Recherche
   - Exemple
3. Algorithme des fonctions
   - CREATE_FILTER
   - PUT
   - SEARCH
4. Résultat de tests
   - Contexte de test
   - Recherche aléatoire
   - Recherche selective
5. Question

Présentation

Problème classique

- volume de données de plus en plus importants
- accès à l'information $\rightarrow$ **efficacité**(Temps et coût)

FreeCore

Search : psar sujets

- Approche : Filtre de Bloom
- ~~Méthode d'accès FreeCore : Indexation(Listes inversées)~~
- Méthode d'accès : Indexation(Vecteur d'approximation)

Filtre de Bloom

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

TABLE: Filtre de Bloom

Filtre de Bloom

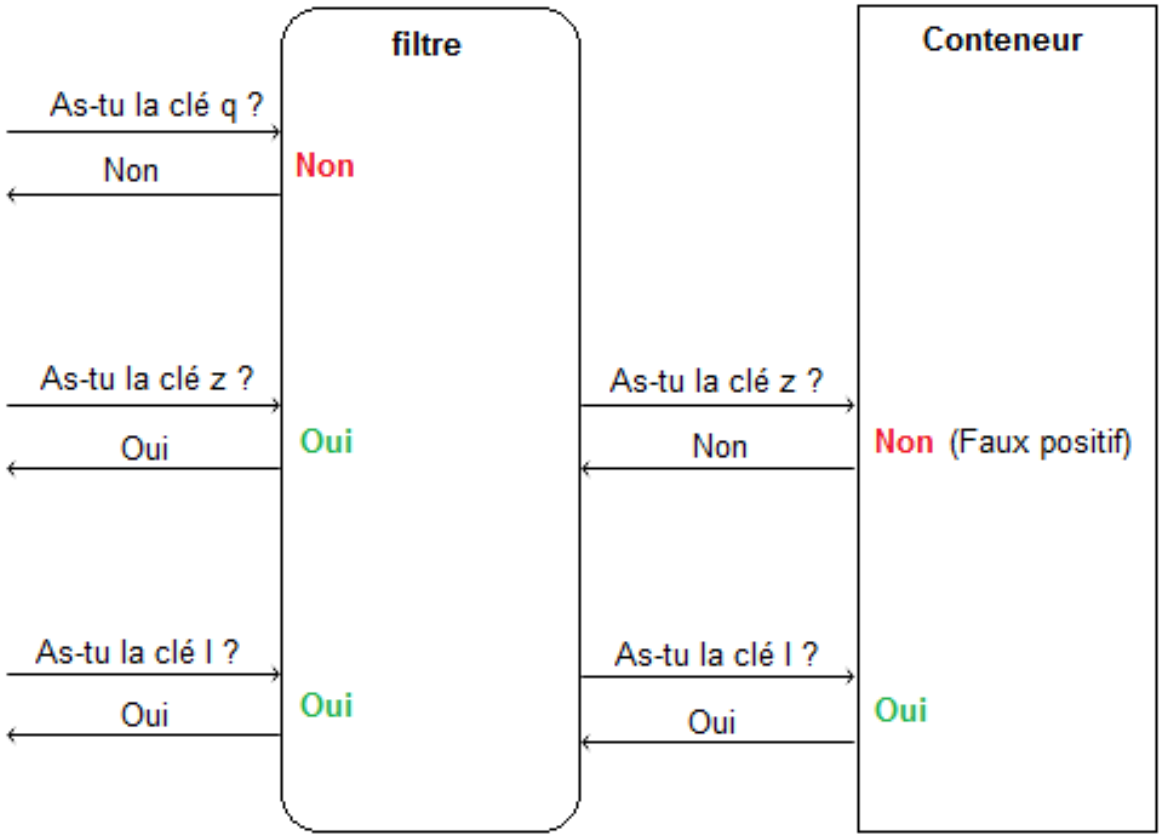


FIGURE: isMember ?

IN : x objet à insérer dans le filtre de Bloom B
FUNCTION : *insert(x)*
OUT : $\emptyset$

---

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

Filtre de Bloom

Recherche dans le filtre de Bloom

IN : x objet à tester dans le filtre de Bloom B
FUNCTION : *ismember(x)*
OUT : *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

Filtre de Bloom

Exemple

m=16(taille du filtre)
k=3 (nombre de tables d'hachage)
Filtre de Bloom : 0000000000000000

Ajout :
h1(a) | h2(a) | h3(a) = 0000000100001010
h1(b) | h2(b) | h3(b) = 1000001100000000
h1(y) | h2(y) | h3(y) = 0010010000100000
h1(l) | h2(l) | h3(l) = 0100000010001000

Filtre de Bloom : 1110011100101010 - *les positions 8 et 3 sont des collisions*

Requête:
h1(q) | h2(q) | h3(q) = 0010000010000001 - ***q*** *n'est pas présent*
h1(z) | h2(z) | h3(z) = 1000010010000000 - ***z*** *et* ***l*** *sont*
h1(l) | h2(l) | h3(l) = 0100000010001000 *probablement présent*

Filtre de Bloom

Approche d'indexation avec le filtre de Bloom

Exemple : Indexation avec le filtre de Bloom

Soient 4 documents à publier représentés par un triplet comprenant : un contenu, un ensemble de mots clés, et une estampille qui identifie de façon unique cette publication parmi toutes les autres publications

Doc[1]=(Contenu[1],mots_clé[1],estampille[1])
Doc[2]=(Contenu[2],mots_clé[2],estampille[2])
Doc[3]=(Contenu[3],mots_clé[3],estampille[3])
Doc[4]=(Contenu[4],mots_clé[4],estampille[4])

Calcule des clés de stockage des publications

Doc[1] ⇒ (D1,d1) = (D1, 1001001010011001) = 37 529
Doc[2] ⇒ (D2,d2) = (D2, 0001100100011010) = 6 426
Doc[3] ⇒ (D3,d3) = (D3, 0000010111110001) = 1 521
Doc[4] ⇒ (D4,d4) = (D4, 0001100010010011) = 6 291

k	*Clé de stockage*	*Contenus (Di)*
0	0000000000000000	...
1	0000000000000001	...
2	0000000000000010	...
...		
1521	0000010111110001	D3, ...
...		
6291	0001100010010011	D4, ...
...		
6426	0001100100011010	D2, ...
...		
37 529	1001001010011001	D1, ...
...		
2^{16}-1= 65 535	1111111111111111	...

Filtre de Bloom

Approche de vecteur approché

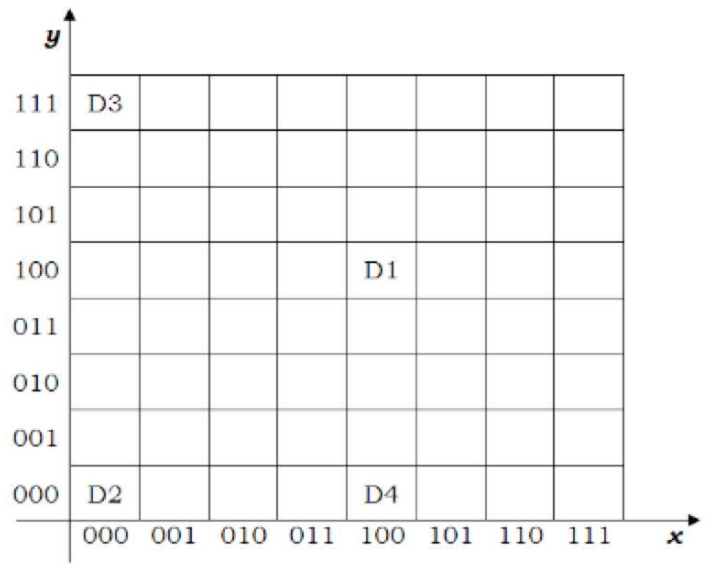
- Chaque filtre de Bloom peut être vu comme un point dans un espace à n dimensions.

- 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. (2e Partie)

n°	Vecteurs	fourchette
0	000	000 00000 ⇒ 000 11111
1	001	001 00000 ⇒ 001 11111
2	010	010 00000 ⇒ 010 11111
3	011	011 00000 ⇒ 011 11111
4	100	100 00000 ⇒ 100 11111
5	101	101 00000 ⇒ 101 11111
6	110	110 00000 ⇒ 110 11111
7	111	111 00000 ⇒ 111 11111

Estampilles	X	Y	Coordonnée
D1	100 10010	100 11001	(4,4)
D2	000 11001	000 11010	(0,0)
D3	000 00101	111 11001	(0,7)
D4	000 11000	100 10011	(0,4)

k	Clé de stockage	Contenus (Di)
0	0000000000000000	...
1	0000000000000001	...
2	0000000000000010	...
...		
1521	0000010111110001	D3, ...
...		
6291	0001100010010011	D4, ...
...		
6426	0001100100011010	D2, ...
...		
37 529	1001001010011001	D1, ...
...		
$2^{16}-1=65\,535$	1111111111111111	...



Algorithme des fonctions

CREATE_FILTER

IN : $\sum desc$
FUNCTION : $create\_filter(\sum desc)$
OUT : B^{512}

$init(B^{512})$
$x \leftarrow \text{FIRST}(\sum desc)$
while $x \neq \emptyset$ **do**
    $i \leftarrow \text{SHA\_256}(x)$
    $j \leftarrow i \ mod \ 512$
    $B^{512}[j] \leftarrow 1$
    $x \leftarrow \text{NEXT}(\sum desc)$
end
return B^{512}

Algorithme des fonctions

PUT

IN : filtre de Bloom de taille 512 bits B^{512}
FUNCTION : $put(B^{512})$
OUT : $\emptyset$

$i \leftarrow$ MAX_LEVEL
$vector_i \leftarrow$ CREATE_VECTOR(B^{512}, i)
$x \leftarrow$ FIRST(VA_file)
while $x \neq \emptyset$ **do**
    if $vector_i = x$ **then**
        BREAK
    end
    $x \leftarrow$ NEXT(VA_file)
end

Algorithme des fonctions

PUT(suite)

if $vector_i \neq x$ **then**
  $VA\_file \leftarrow$ ADD($vector_i$)
end
for $i = MAX\_LEVEL \ldots 1$ **do**
  if $i = 1$ **then**
    $vector_i \leftarrow$ CREATE_VECTOR(B^{512}, i)
    CREATE_FILE($vector_i$, B^{512})
  else
    $vector_i \leftarrow$ CREATE_VECTOR(B^{512}, i)
    CREATE_FILE($vector_i$, CREATE_VECTOR(B^{512}, i - 1))
  end
end
return $\emptyset$

Algorithme des fonctions

SEARCH

IN : filtre de Bloom de taille 512 bits B_{req}^{512}
FUNCTION : $search(B_{req}^{512})$
OUT : $\sum doc$

$i \leftarrow$ MAX_LEVEL
$vector_i \leftarrow$ CREATE_VECTOR(B_{req}^{512}, i)
$tmp \leftarrow$ CREATE_FILE(i)
$x \leftarrow$ FIRST(VA_file)
while $x \neq \emptyset$ **do**
    if $vector_i \subseteq x$ then
        tmp $\leftarrow$ ADD(x)
    end
    $x \leftarrow$ NEXT(VA_file)
end

Algorithme des fonctions

SEARCH(suite)

for $i = MAX\_LEVEL - 1 \ldots 1$ **do**
    $vector_i \leftarrow$ CREATE_VECTOR(B_{req}^{512}, i)
    $x \leftarrow$ FIRST(FILE($i + 1$))
    $tmp \leftarrow$ CREATE_FILE(i)
    while $x \neq \emptyset$ **do**
        $y \leftarrow$ FIRST(FILE(x))
        while $y \neq \emptyset$ **do**
            if $vector_i \subseteq y$ **then**
                tmp $\leftarrow$ ADD(y)
            end
            $y \leftarrow$ NEXT(FILE(x))
        end
        $x \leftarrow$ NEXT(FILE($i + 1$))
    end
end

Algorithme des fonctions

SEARCH(suite)

$x \leftarrow$ FIRST(FILE(1))
while $x \neq \emptyset$ **do**
    $y \leftarrow$ FIRST(FILE(x))
    while $y \neq \emptyset$ **do**
        if $B_{req}^{512} \subseteq y$ **then**
            $\sum doc \leftarrow$ ADD(FIRST(FILE(y)))
        end
        $y \leftarrow$ NEXT(FILE(x))
    end
    $x \leftarrow$ NEXT(FILE(1))
end
return $\sum doc$

Résultat de test

Contexte de test

- 107 459 documents
- 319 720 fichiers générés
- Filtre de Bloom de taille 512 bits
- SHA_256
- 6 niveaux index

Résultat de tests

Recherche aléatoire

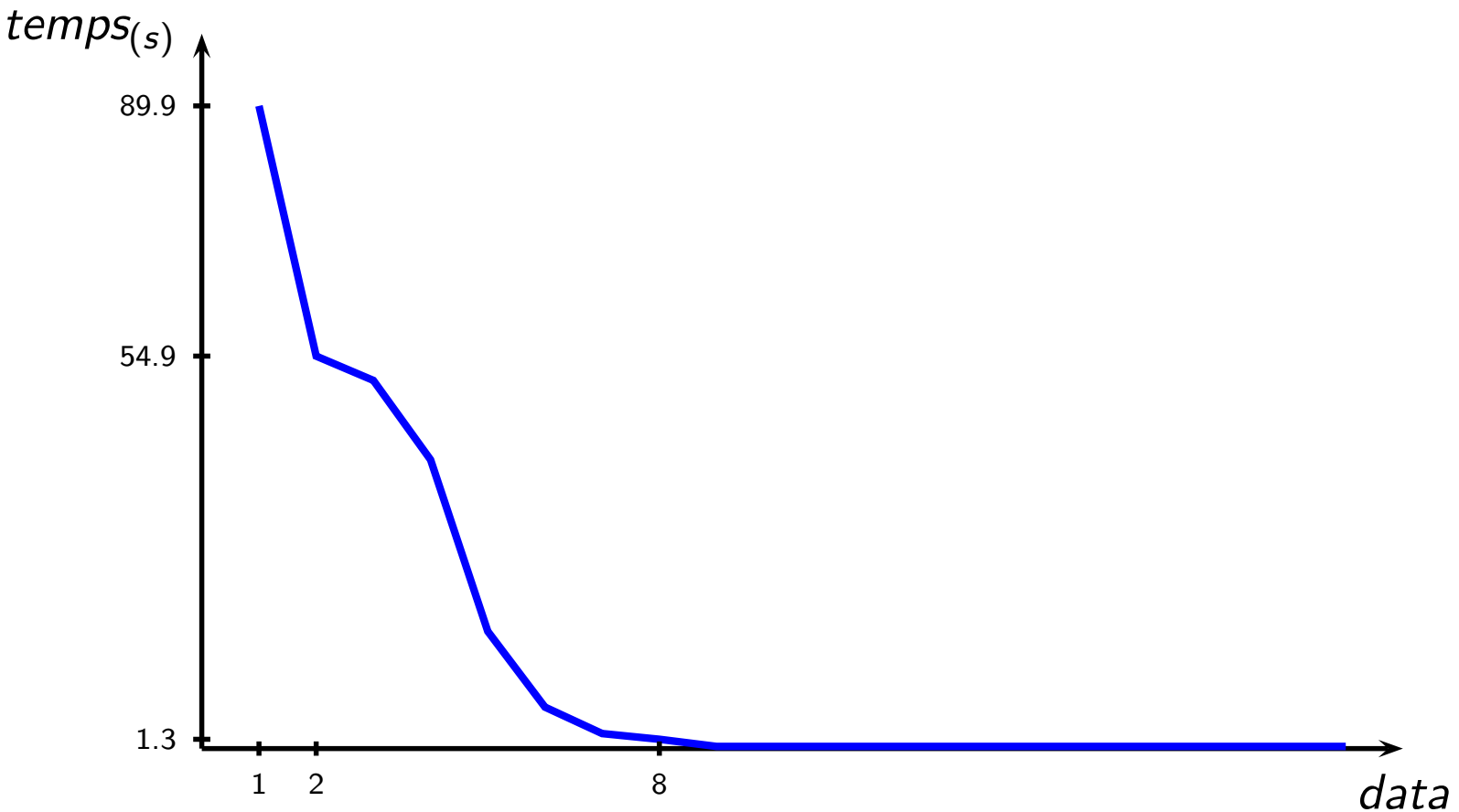


FIGURE: Recherche aléatoire

Résultat de tests

Recherche sélective

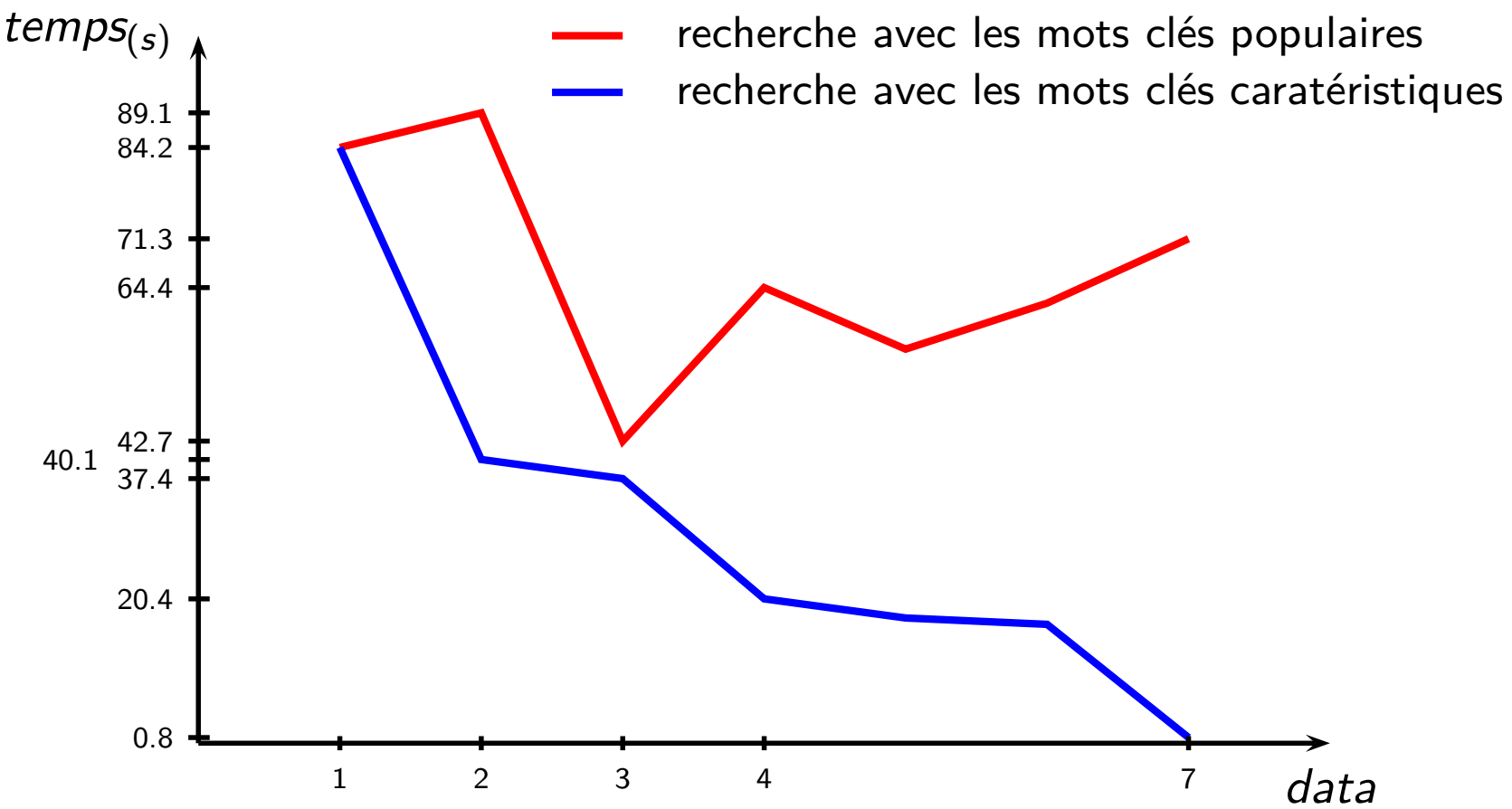


FIGURE: Recherche sélective

Question

Merci de votre attention & Question