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

30 June 2015

Table des matières

1 Demande du travail **2**

2 Spécification du système **3**
2.1 Structure de données spécifiques 3
2.1.1 Filter 3
2.1.2 Couple 3
2.1.3 VA 4
2.2 Class Node 4
2.2.1 Variables locales 4
2.2.2 Méthodes 5

Chapitre 1

Demande du travail

Le travail comprend principalement 2 tâches :

— La programmation des primitives de base de F-PHT : add(), split(), remove() et search().

— L'évaluation des performances du système F-PHT pour une configuration donnée ($m = 512$;$f = 64$; $\gamma = 1000$)

1. coûts des insertions ;
2. coût d'une recherche ;
3. overhead mémoire ;
4. qualité de l'arbre(notamment équilibrage ou déséquilibrage de l'arbre)

Chapitre 2

Spécification du système

2.1 Structure de données spécifiques

2.1.1 Filter

Filter est soit un bitmap, soit un filtre de Bloom. La taille de ce filtre ou ce bitmap est configurable selon le besoin du système.

Variables locales

Char[] filter : filter est un tableau de caractères {0,1} de taille fixe qui est fixée et configurable par le système.

Méthodes

Filter Filter(String description, int size) utilise 3 fonctions de hachage pour générer les bits à 1 et les remplir dans le tableau de caractères *filter* à la position correspondante. Ici, la taille de filtre doit être précisé grâce au argument *size*.

Filter Filter(String s) : crée *filter* à partir d'une chaine de caractères {0,1}.

Filter Filter(char[] c) : crée *filter* à partir d'un tableau de caractères {0,1}.

boolean in(Object o) : teste si ce *filter* est dans l'objet *o*.

boolean isFilter() : retourne **true** si *filter* est un filtre de Bloom, sinon **false**.

Filter createIDAtRang(int rang) : crée un identifiant à partir d'un filtre au rang **i**.

2.1.2 Couple

Couple est un couple de valeur **<bitmap, Next_hop>**

Variables locales

Filter reference : un filtre soit le filtre de Bloom, soit l'identifiant d'un noeud dans le système.

int IP : l'adresse IP du noeud qui gère cette référence ou 0 si cette référence est un filtre de Bloom.

Méthodes

Couple Couple(Filter reference) : crée un couple **<reference, null>**.

Couple Couple(Filter reference, int IP) : crée un couple **<reference, IP>**.

2.1.3 VA

Cette structure est une liste des couples *<bitmap, Next_hop>*. Chaque noeud de l'arbre contient cette structure avec bitmap qui est une chaîne de bits, c'est une "instance de **Filter**" et Next_hop qui est soit NULL soit égal à un identifiant d'un noeud, ce ID est aussi une "instance de **Filter**". Cette structure contient également le nombre maximum de couples qu'elle peut stocker.

Variables locales

ArrayList<Couple> list : la liste des couples *<reference, IP>*.

int limit : le nombre maximum des éléments que la liste peut contenir.

Méthodes

void VA(int max) : initialise le nombre maximum des éléments dans *list*.

boolean add(Couple c) : ajout le couple *c* dans la liste des couples *list*, renvoie **true** si réussi, sinon **false**.

2.2 Class Node

Cette classe joue le rôle à la fois d'un noeud et à la fois d'une feuille dans l'arbre.

2.2.1 Variables locales

Filter ID identifiant de ce noeud, cet id est unique.

int IP l'adresse IP de ce noeud.

int rang le rang de ce noeud dans l'arbre.

boolean leaf indique si ce noeud est une feuille, c-à-d si ce noeud a déjà fait ***split()***. Initialisé à **true**.

int father le père de ce noeud dans l'arbre.

VA va cette variable est une structure VA qui contient des couples *<bitmap, Next_hop>*.

int root l'adresse du serveur central.

2.2.2 Méthodes

void split()

Cette méthode est appellée par la méthode ***add(Filter)*** dès que la limit de nombre de couples dans la structure VA *va* a été atteinte. Cette méthode va créer une nouvelle structure VA *va_new*, elle parcourt toute la structure *va*, pour chaque élement, elle calcule l'identifiant de rang *i+1* ID_{i+1} et demande au serveur central pour créer un nouveau noeud de ID_{i+1}, le serveur répond en envoyant l'adresse IP du nouveau noeud créé $IP_{ID_{i+1}}$. Elle envoie cet élement au nouveau noeud. Ensuite, elle ajoute le couple $<ID_{i+1} ; IP_{ID_{i+1}}>$ dans *va_new*. Après, elle supprime *va* et renomme *va_new* en *va*. Elle change la valeur de la variable locale **leaf** en **false**.

Algorithme 1 *split()*

INPUT : ∅
OUTPUT : ∅

---

1. VA *va_new = new VA()*
2. Couple *couple = va.getFirstElement()*
3. Filter *f = couple.getFilter()*
4. Filter *id*
5. **while** (*couple != null*)
   f = couple.getReference()
   id = f.createIDAtRang(i + 1)
   send(root, CREATE, id)
   int *IP = receive(root, CREATE, id)*
   va_new.add(<id, IP>)
   send(IP, ADD, f)
   couple = va.getNextElement()

end

add(Filter)

Dès que ce noeud reçoit la commande d'ajout d'un filtre dans sa base de données, ce noeud appelle cette méthode. Elle va créer un fragment de ce filtre qui correspond avec le rang **i** de ce noeud, si ce fragment n'est pas identique avec l'identifiant de ce noeud, il jette la commande. Si ce noeud est un noed, elle calcule l'identifiant du noeud au rang **i+1** ID_{i+1}, elle cherche dans *va*, s'il existe un fils qui a le même identifiant, elle renvoie ce filtre au fils. Sinon, elle demande au serveur central pour créer un nouveau noeud de ID_{i+1}, le serveur répond en envoyant l'adresse IP du nouveau noeud crée $IP_{ID_{i+1}}$. Elle envoie ce filtre au noeud reçu. Ensuite, elle ajoute le couple $<ID_{i+1} ; IP_{ID_{i+1}}>$ dans *va*. Si c'est une feuille, elle cherche dans la structure VA, s'il y a des places libres, si oui, elle ajoute dans cette structure le filtre, sinon, elle appelle la méthode ***split()***. Après le lancement de la méthode **split()**, elle calcule l'identifiant du noeud au rang **i+1** ID_{i+1} et recherche dans la structure VA *va*, s'il existe, elle envoie à l'adresse indiquée, sinon, elle demande au serveur central pour créer un nouveau noeud de ID_{i+1}, le serveur répond en envoyant l'adresse IP du nouveau noeud crée $IP_{ID_{i+1}}$. Elle envoie ce filtre au noeud reçu. Ensuite, elle ajoute le couple $<ID_{i+1} ; IP_{ID_{i+1}}>$ dans *va*.

Algorithme 2 *add(Filter)*

INPUT : Filter f
OUTPUT : $\emptyset$

---

1. Filter *id = f.createIDAtRang(i)*
2. **if**(*!id.equals(ID)*)
   return
   end
3. *id = f.createIDAtRang(i+1)*
4. **if**(*!leaf*)
   if *(va.exist(id))*
   send(va.getIP(id), ADD, f)
   else
   send(root, CREATE, id)
   int *IP = receive(root, CREATE, id)*
   va.add(<id, IP>)
   send(IP, ADD, f)
   end
   else
   if *(!va.add(<f, null>))*

```
        split()
        if (va.existe(id))
          send(va.getIP(id), ADD, f)
        else
          send(root, CREATE, id)
          int IP = receive(root, CREATE, id)
          va.add(<id, IP>)
          send(IP, ADD, f)
        end
      end
    end
```

search(Filter)

D'abord, cette méthode calcule si ce filtre correspond avec l'identifiant de ce noeud au rang **i**, si ce filtre ne l'appartient pas, il renvoies *null* au père. Si oui, elle calcule l'identifiant de ce filter au rang **i+1** ID_{i+1}, elle parcourt tous les élements dans la structure VA *va*. Si ce noeud est un noeud, elle va chercher tous les élements qui contient ID_{i+1} et envoye la commande **search(Filter)** vers le fils qui gère. Elle attend la réponse de tous les fils auxquels elle a envoyé. Dès qu'elle reçoit toutes les réponses soit elle affiche les réponse, le cas du root, soit elle les renvoie au père. Si ce noeud est une feuille, elle va collecter tous les filtres qui contiennent la requête et les renvoyer au père.

Algorithme 3 *search(Filter)*

INPUT : Filter f
OUTPUT : ∅

---

1. Filter *id = f.createIDAtRang(i)*
2. SI *(!id.in(ID))*
   send(father, SEARCH, null)
   FIN SI
3. **if** *(leaf)*
   Couple *couple = va.getFirstElement()*
   Filter *tmp*
   ArrayList<Filter> *listeFiltre = new ArrayList<Filter>()*
   while *(couple != null)*
   tmp = couple.getFilter()

```
            if (f.in(tmp))
               listeFiltre.add(tmp)
            end
            couple = va.getNextElement()
      end
      send(father, SEARCH, listeFiltre)
   else
      id = f.createIDAtRang(i+1)
      Couple couple = va.getFirstElement()
      Filter tmp
      int fils = 0
      ArrayList<Filter> listeFiltre = new ArrayList<Filter>()
      while (couple != null)
            tmp = couple.getReference()
            if (id.in(tmp))
               send(couple.getIP(), SEARCH, f)
               fils++
            end
            couple = va.getNextElement()
      end
      while (fils != 0)
            listeFiltre.add(receive(*, SEARCH, f))
            fils—
      end
      if (father != null)
         send(father, SEARCH, listeFiltre)
      else
         send(root, SEARCH, listeFiltre)
      end
   end
```