Structures de données avancées :

Distributed Compact Trie Hashing : CTH*

Pr ZEGOUR DJAMEL EDDINE
Ecole Supérieure d'Informatique (ESI)
www.zegour.uuuq.com
email: d_zegour@esi.dz

TH

Rappel Hachage Digital (TH)

- **Le hachage digital (81) est l'une des méthodes les plus rapides pour l'accès au fichiers monoclé, ordonnés et dynamiques.**
- **La technique utilise une fonction de hachage variable représentée par un arbre digital qui pousse et se rétracte en fonction des insertions et suppressions.**
- **Caractéristiques principales :**
  - ✓ **l'arbre réside en mémoire pendant l'exploitation du fichier.**
  - ✓ **6 Octets / case**
  - ✓ **Un accès au plus pour retrouver un article**

CTH

Rappel Hachage digital compact (CTH)

- Plusieurs manières de représenter la fonction d'accès en mémoire. ➔ Objectifs : doubler les fichiers adressés pour le même espace mémoire utilisé par la représentation standard.
- L'idée : représenter les liens de manière implicite au détriment d'algorithmes de maintenance légèrement plus long.
- Consommation : 3 octets par case du fichier.
  ➔ Ce qui permet d'adresser des millions d'articles avec très peu d'espace mémoire.
- Intéressante pour un environnement distribué.

CTH*

Mécanisme de construction

Distribution de CTH relativement aux propriétés des Sdds :

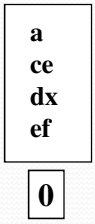
- Distribution des cases du fichier sur les serveurs(à raison d'un serveur par case)
- Pas de site maître
- Aucun dialogue entre les clients.
- Scalabilité

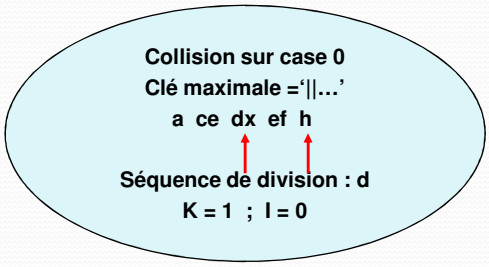
CTH

Mécanisme de construction

1. *a ce dx ef sont insérées dans la case 0.*

L'arbre : | 0





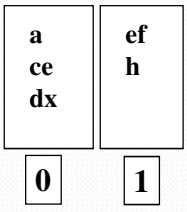
Capacité de la case = 4

CTH

Mécanisme de construction

2. *Après insertion de h*

L'arbre : d 0 | 1

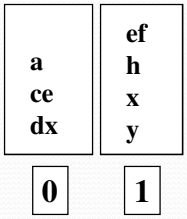


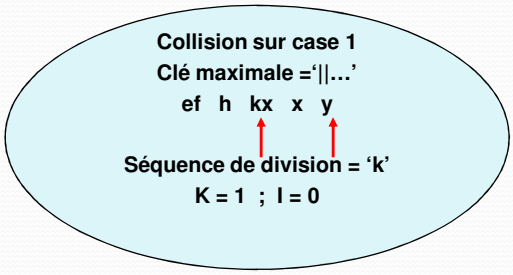
CTH

Mécanisme de construction

3. Insertion de x y

L'arbre : d 0 | 1



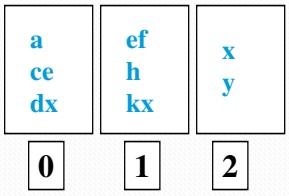


CTH

Mécanisme de construction

4. Après insertion de kx

L'arbre : d 0 k 1 | 2



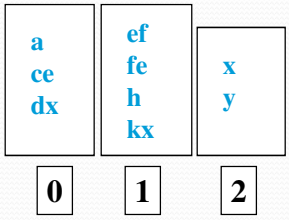
CTH

Mécanisme de construction

5. Insertion de fe

hx

L'arbre : d 0 k 1 | 2



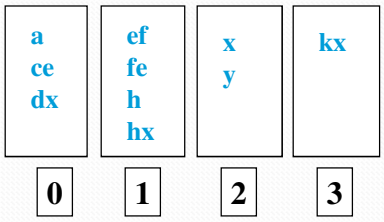
Collision sur case 1
Clé maximale = 'k|…'
ef fe h hx k
Séquence de division = 'h'
K = 1 ; I = 0

CTH

Mécanisme de construction

6. Après insertion de hx

L'arbre : d 0 h 1 k 3 | 2



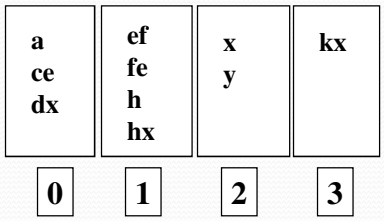
CTH

Mécanisme de construction

7. Insertion de hy

hy

L'arbre : d o h 1 k 3 | 2



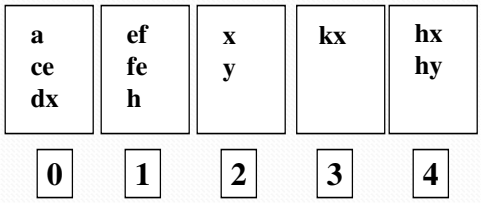
Collision sur case 1
Clé maximale ='h|…'
ef fe h hx hy
Séquence de division = 'h_'
K = 2 ; l = 1

CTH

Mécanisme de construction

8. Après insertion de hy

L'arbre : d o h _ 1 4 k 3 | 2



CTH

Mécanisme de construction

9. Insertion de yya yyb

yyc

L'arbre : d o h _ 1 4 k 3 | 2

0	1	2	3	4
a ce dx	ef fe h	x y yya yyb	kx	hx hy

Collision sur case 2
Clé maximale ='||…'
x y yya yyb yyc
Séquence de division = 'yya'
K = 3 ; l = 0

CTH

Mécanisme de construction

9. Après insertion de yyc

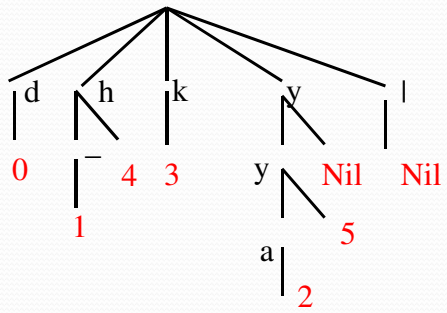
L'arbre : d o h _ 1 4 k 3 y y a 2 5 Nil | Nil

0	1	2	3	4	5
a ce dx	ef fe h	x y yya	kx	hx hy	yyb yyc

CTH

Mécanisme de construction

ARBRE : d o h _ 1 4 k 3 y y a 2 5 Nil | Nil



Représentation graphique

- Arbre = suite de nœuds internes et externes
- Arbre = forme préordre
- La concaténation des digits sur une branche de l'arbre représente la clé maximale de la case figurant dans la feuille

CTH

Expansion de l'arbre

Soit **m** la case à éclater. **Cm** sa clé maximale (Cm=d1d2...)

- Former la séquence ordonnée des clés de cette case augmentée de la clé qui a provoqué la collision.

Soit **C'** la clé du milieu et **C''** la dernière.

- Déterminer la plus petite séquence de digits dans C' qui permet de distinguer C' de C''. Soit $Seq = C'_1 C'_2 \ldots C'_K$ cette séquence.

Soit **I** les premiers digits de cette séquence qui existent déjà dans l'arbre.

CTH

Expansion de l'arbre (Suite)

Soient:

Ind_d : l'indice dans l'arbre du premier digit (c'est d1)de la clé maximale cm de la case surchargée

Ind_m : l'indice de la case surchargée

```
Si  I <> 0
      Si Cm  est un préfixe de Seq
            Ind_d := Ind_m
      Sinon
            Ind_d := l'indice du premier digit différent entre Cm et  Seq
      Fsi
Fsi
```

CTH

Expansion de l'arbre (Suite)

Cas k- I = 1
Remplacer m par M
insérer à la position Ind_d : C'_k **m**

Cas k- I > 1
Remplacer m par Nil
Insérer à la position Ind_d
C'_{i+1} C'_{i+2} .....C'_k **m M** Nil_1 Nil_2 ...Nil_{k-i-2}

Dans le premier cas deux nœuds sont rajoutés.

dans le second cas 2(K - I) nœuds sont rajoutés.

M étant la prochaine case à allouer au fichier

CTH

Expansion de l'arbre

Exemple

Arbre : b p o 8 d 2 ....

Collision sur o

Cm = 'bp'

Supposons Seq = 'bn'

Cm n'est pas un préfixe de Seq → ind_d = 2

Remplacer o par 22 (22 : prochaine case)

Insérer à la position 2 : n o

Ce qui donne b n o p 22 8 d 2 ...

CTH

Recherche

```
Init( P ) ;  I := 1;  Trouv := Faux
Tq Non Trouv :
    Si Interne(Arbre[ I ])
        Empiler(P, Arbre[ I ] )
        I := I + 1
    Sinon
        Si C <= ( P )
            Trouv := Vrai
        Sinon
            Depiler(P, V )   // V non utilisée
            I := I + 1
        Fsi
    Fsi
Ftq
```

Considérations

Arbre : table contenant l'arbre.

P : pile

(P) désigne le contenu de la pile en commençant par le fond si la pile n'est pas vide, ':' sinon.

CTH

Propriétés et performances

- **Propriété**

  Les cases sont ordonnées de gauche à droite

- **Algorithmes en mémoire :**
  - ✓ **Recherche : N/2 en moyenne**
  - ✓ **Insertion : N/2 décalages en moyenne**
  - ✓ **Encombrement : 3 octets / case en moyenne.**

- **Algorithmes sur disque :**
  - ✓ **1 accès au plus pour retrouver un article**

CTH

Modification du schéma de base

- **Le problème**

Au moment de l'insertion on peut être amener à remplacer Nil par une case contenant une seule clé → chuter le facteur de chargement.

Dans un environnement distribué créer tout un serveur avec une seule donnée !!!

- **Solution**

Retarder le remplacement des Nils par des cases en insérant les clés dans la case non Nil la plus proche à droite.

Les algorithmes de recherche et insertion sont modifiés en conséquence

CTH

Modification du schéma de base

- **Recherche**

Ne retourne jamais Nil

- **Éclatement (première manière)**

Lorsque une case m déborde, trouver le nœud externe précédent, soit M.
Si M est Nil et il existe des clés C telle que C ≤ Clé max(M)
alors
Remplacer Nil par une nouvelle case et
Déplacer ces clés C dans cette case.

Si M est non Nil,
faire un éclatement normal

CTH

Modification du schéma de base

Éclatement (deuxième manière)

1. Éclater m normalement

2. Si la séquence de division correspond à la clé max d'un nœud Nil,
Faire a) OU b)

- ✓ **a) ce dernier est remplacé par la case éclatée m.**

La nouvelle case créée, soit N remplace m et les clés de m > à la séquence de division seront transférées dans N.

- ✓ **b) laisser le Nil**

CTH

Modification du schéma de base / Comparaison

Éclatement (première manière)

Taux de nœuds Nil légèrement supérieur à celui de la version de base.

Éclatement (deuxième manière)

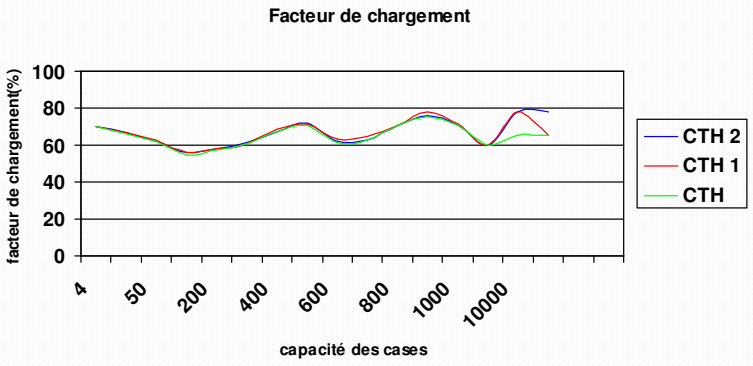
- ✓**Taux de nœuds Nil plus important car le cas ou la séquence de division correspond à un nœud Nil est rare.**
- ✓**Taille de l'arbre augmente**

→ **Intéressante dans un environnement distribué.**

- ✓**Cas 2b) garder un nœud Nil en plus et éviter de faire un accès au serveur père pour voir s'il y a un nœud Nil qui le précède.**

CTH

Modification du schéma de base / Performance



CTH

Modification du schéma de base / Performances

Taux des nœuds Nil (en %)

100 000 clés aléatoires

Capacité →	4	10	50	100	200
Variante1	0,75	0,40	0,53	0,91	0,73
Variante2	0,76	0,40	0,53	0,90	0,72
Base	0,35	0,16	0,09	0,06	0,00

CTH*

Concept

- **Au niveau de chaque serveur il y a**
  - ✓ **Un arbre digital partiel**
  - ✓ **Une case contenant les articles du fichier**
- **L'arbre digital au niveau du serveur garde la trace de tous les éclatements sur ce serveur.**

CTH*

Concept

- Au niveau de chaque client il y a un arbre digital partiel
- Tout client commence avec un arbre vide (| o)
- Mise à jour progressive de l'arbre du client.

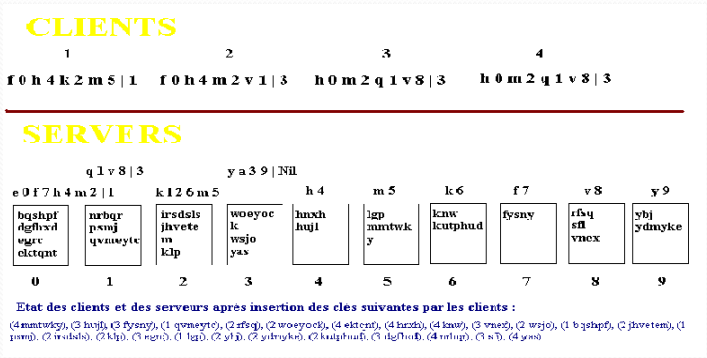
CTH*

Concept

- L'expansion du fichier se fait à travers les collisions.
- A chaque collision il y a distribution du fichier (du serveur éclaté) sur un nouveau serveur.
- Quand une case (serveur) éclate , il y a
  - ✓Extension de l'arbre du serveur
  - ✓Initialisation d'un nouveau serveur avec un arbre vide
  - ✓Partage des articles entre l'ancien et le nouveau
- Initialisation du système
  Serveur o avec case vide ; Intervalle =]- ∞, + ∞[; Arbre : | o

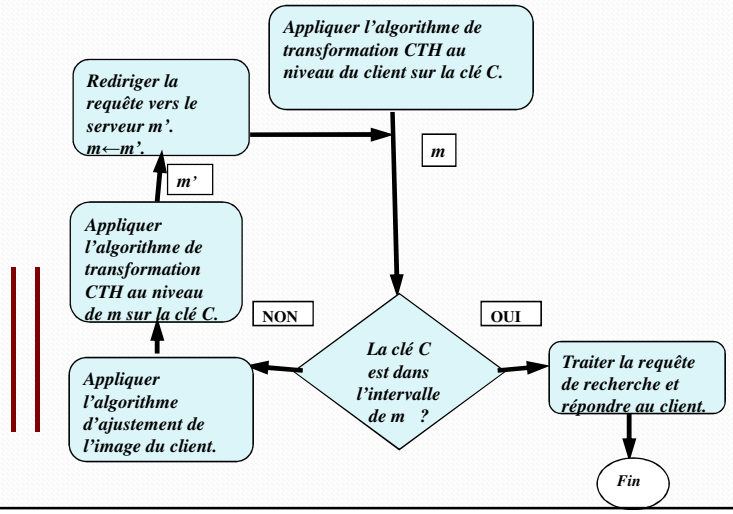
CTH*

Illustration



CTH*

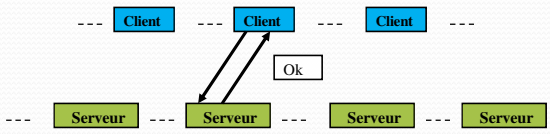
Transformation (Client, clé) → Serveur



CTH*

Transformation (Client, clé) → Serveur

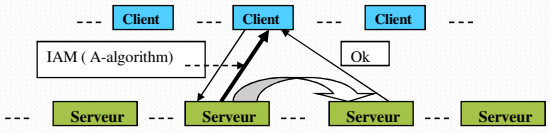
Bon adressage



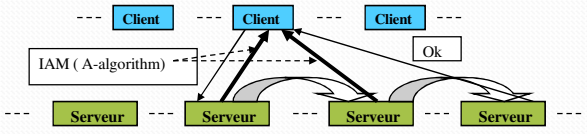
CTH*

Transformation (Client, clé) → Serveur

Mauvais adressage (avec 1 forward)



Mauvais adressage (avec 2 forward)



CTH*

Ajustement de l'arbre du client

Soit CM' = $C_0C_1...C_k$ la clé maximale du serveur trouvé (m) dans l'arbre du client
CM, la clé maximale du serveur m selon l'arbre du serveur

- ***1) Trouver dans l'arbre du client la première branche b qui vérifie $b+'||...|' \geq CM$ avec $b=b_1 b_2 ...b_{Taille(b)}$***

- ***2) Remplacer la séquence de l'arbre du client (b_{M+1} ...m) par la séquence de l'arbre du serveur ($C_{M+1}...C_k$ m...m')***

- ***M étant la taille de préfixe commun entre CM et b***
- **m' est le serveur qui correspond à CM' dans l'arbre du serveur**

CTH*

Ajustement de l'arbre du client / Exemple

- **Arbre du client : a _ 0 Nil | 1**
- **Serveur1 : a a _ 1 Nil Nil | 2**

- **Données client : m=1 et CM'= '|'**
- **Données serveur : CM='aa_' ; m'=2 ; P = a a _ 1 Nil Nil | 2**

- **1) b ='aNil'**
- **2) M = 1**
- **3) b_{M+1} *est Nil***

→ **a _ 0 a _ 1 Nil Nil | 2**

CTH*

Ajustement de l'arbre du client / Exemple

- Arbre du client : b a b a a a o Nil Nil Nil Nil Nil | 1
- Serveur 1 : b a b a 1 Nil Nil Nil | 2

- Données client : m=1 et CM'= '|'
- Données serveur : CM='baba' ; m'=2 ; P = b a b a 1 Nil Nil Nil | 2

- 1) b ='babaNil'
- 2) M = 4
- 3) b_{M+1} *est Nil*

→b a b a a a o Nil 1 Nil Nil Nil | 2

CTH*

Ajustement de l'arbre du client / Exemple

- Arbre du client : a o b a a b a 1 Nil Nil 3 Nil | 2
- Serveur o : a a a o 6 5 | 1

- Données client : m=o et CM'= 'a'
- Données serveur : CM='aaa' ; m'=5 ; P = a a a o 6 5

- 1) b ='ao'
- 2) M = 1
- 3) b_{M+1} *est o*

→a a a o 6 5 b a a b a 1 Nil Nil 3 Nil | 2

CTH*

Insertion

(i)Si Clé n'est pas dans la case (serveur) et case non pleine insérer tout simplement Clé dans la case et l'algorithme se termine.

(ii) Si Clé n'est pas dans la case et celle-ci est pleine il y a collision.

- ✓ **Éclatement du serveur**

CTH*

Eclatement d'un serveur

1. Appliquer l'algorithme d'éclatement de CTH (Deuxième manière) pour modifier l'arbre du serveur m

2. Partager les clés entre m et m' ;

3. Au niveau du serveur m', initialiser l'arbre avec les digits de CM suivis d'une feuille m' suivie de k-1 nœuds Nils.

m : serveur éclaté
m' : nouveau serveur à créer
CM: clé maximale de m de longueur k.

Garder un nœud Nil en plus et éviter de faire un accès au serveur père pour voir s'il y a un nœud Nil qui le précède.

CTH*

Performances

Scalable Distributed Compact Trie Hashing
Variante 1 : Client Trees, Server trees
D.E ZEGOUR

Number of keys inserted : 500
Bucket capacity : 4
Random insertions
Number of servers generated : 172
Number of nodes generated on the tree of the client 1 : 194
Number of nodes generated on the tree of the client 2 : 222
Number of nodes generated on the tree of the client 3 : 216
Number of nodes generated on the tree of the client 4 : 190
Number of calls to algorithm A :198
Average number of forwards per insert : 0.40
Average number of nodes transferred by algorithm A: 4.47
load factor of server buckets : 72.67 %

CTH*

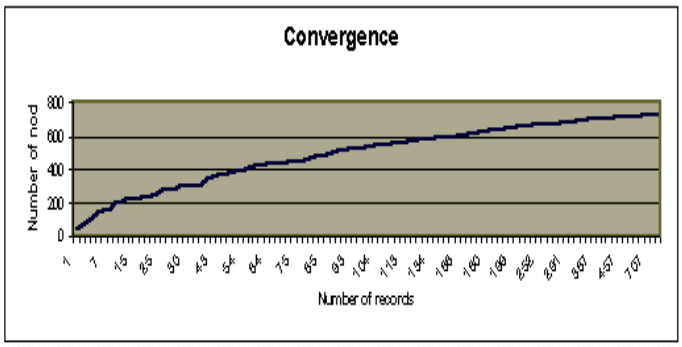
Performances

Scalable Distributed Compact Trie Hashing
Variante 1 : Client Trees, Server trees
D.E ZEGOUR

Number of keys inserted : 2000
Bucket capacity : 4
Random insertions
Number of servers generated : 717
Number of nodes generated on the tree of the client 1 : 792
Number of nodes generated on the tree of the client 2 : 822
Number of nodes generated on the tree of the client 3 : 806
Number of nodes generated on the tree of the client 4 : 900
Number of calls to algorithm A :753
Average number of forwards per insert : 0.38
Average number of nodes transferred by algorithm A: 4.76
load factor of server buckets : 69.74 %

CTH*

Performances



CTH*

Variantes

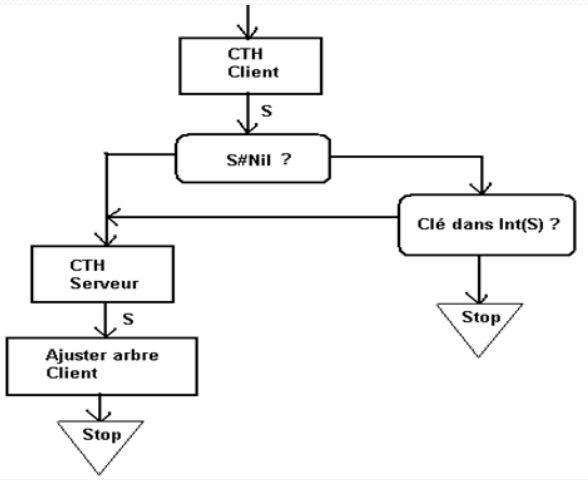
CTH* avec arbre central

- **Pas d'arbre au niveau des serveurs**
- **Arbre réel au niveau d'un serveur**
- **Pas de multicast.**

CTH*

Variantes

CTH* avec arbre central



CTH*

Variantes

CTH* avec arbre central

CLIENTS

Client 1	Client 2	Client 3	Client 4
c o | 1	d 5 u 1 | 6	l 3 u 1 | 4	| 4

Serveur central c 0 d 5 l 3 n 2 r 1 u 7 w 4 \| 6

SERVEURS

0	1	2	3	4	5	6	7
adpha aq buylq ccjuw	p qybtp rcg	mx nrm	ell g h l	v vhx uhws wzfs	dawp dd diy	zdxfd zghoj znh	u uxmvv
[' ', 'c']	['n', 'r']	['l', 'n']	['d', 'l']	['u', 'w']	['c', 'd']	['w', '|']	['r', 'u']

CTH*

Variantes

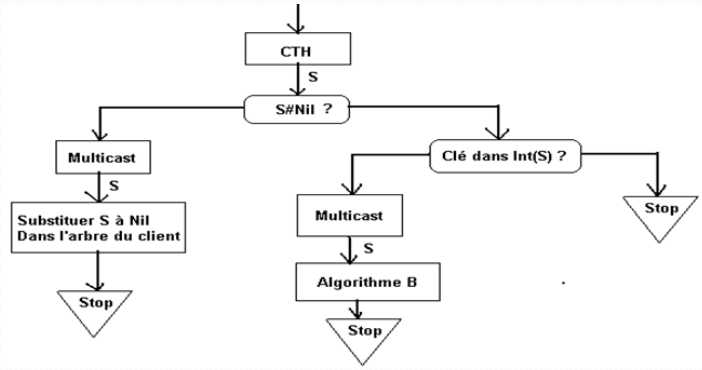
CTH* avec 'Multicast'

- Pas d'arbre au niveau des serveurs
- Pas d'arbre central

CTH*

Variantes

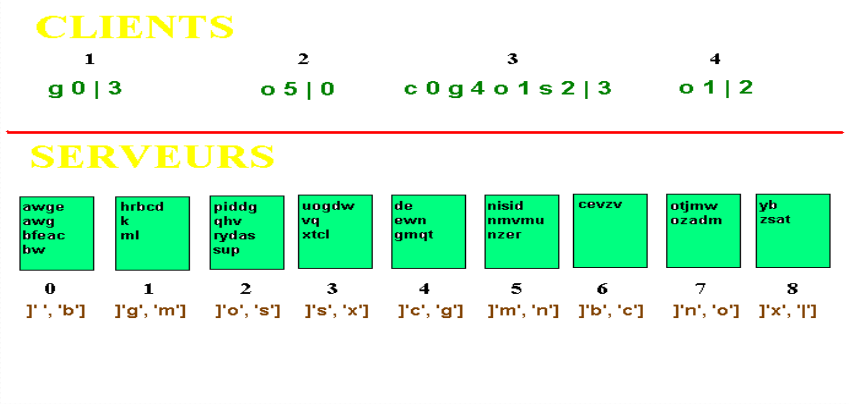
CTH* avec 'Multicast'



CTH*

Variantes

CTH* avec 'Multicast'



CTH*

Conclusion

- Généralisation de TH pour le distribué
- Préservation de l'ordre des articles : facilite les opérations de parcours séquentiel et de requêtes à intervalle.
- Résultats remarquables:
  - Sur 1000 recherches, 400 avec une seule Redirection (forward) en moyenne
  - Chargement des serveurs : 70 %
  - IAM : Quelques octets (< 5)
- Toutes les opérations sans multicast
- Deux variantes sont proposées
- La méthode de base avec chaînage des cases (comme les arbres B+)
- Toutes les variantes de LH* peuvent s'intégrer dans CTH*.