Hachage Digital Compact Scalable et Distribué sans Multicast

D.E ZEGOUR, Institut National d'Informatique, Alger

Hachage Digital Compact Scalable et Distribué sans Multicast

Hachage digital

- **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 (64K Ram pour 750 000 articles)**

✓ **Un accès au plus pour retrouver un article**

Hachage Digital Compact Scalable et Distribué sans Multicast

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. (64 K ram pour 1 500 000 articles)
- **Intéressante pour un environnement distribué.**

Hachage Digital Compact Scalable et Distribué sans Multicast
Comme une SDDS

- **Nous proposons une distribution de CTH relativement aux propriétés des Sdds:**

  - **Distribution des cases du fichier sur les serveurs(à raison d'un serveur par case) : scalabilité**

  - **Pas de site maître**

  - **Aucun dialogue entre les clients.**

Hachage Digital Compact Scalable et Distribué sans Multicast

Plan à suivre

- **Compact trie hashing**
- **Distribution de la méthode sur plusieurs sites.**
- **Illustration de la méthode**
- **Algorithmes de recherche et insertion**
- **Requête à intervalle**
- **Quelques test**
- **Variantes**
- **Conclusion**

Hachage Digital Compact Scalable et Distribué sans Multicast

Hachage digital compact

Insertion des clés : a , ce, dx, ef, h, x, y, kx, fe, hx, hy, yya, yyb, yyc

L'arbre : d 0 h _ 1 4 k 3 y y a 2 5 Nil \| Nil

[λ, d] : 0; [d, h_]: 1; [h_, h] : 4; Etc…

Note → _ : pp caractère; | : pg caractère; λ = '___.._'; Λ = '|||..|'

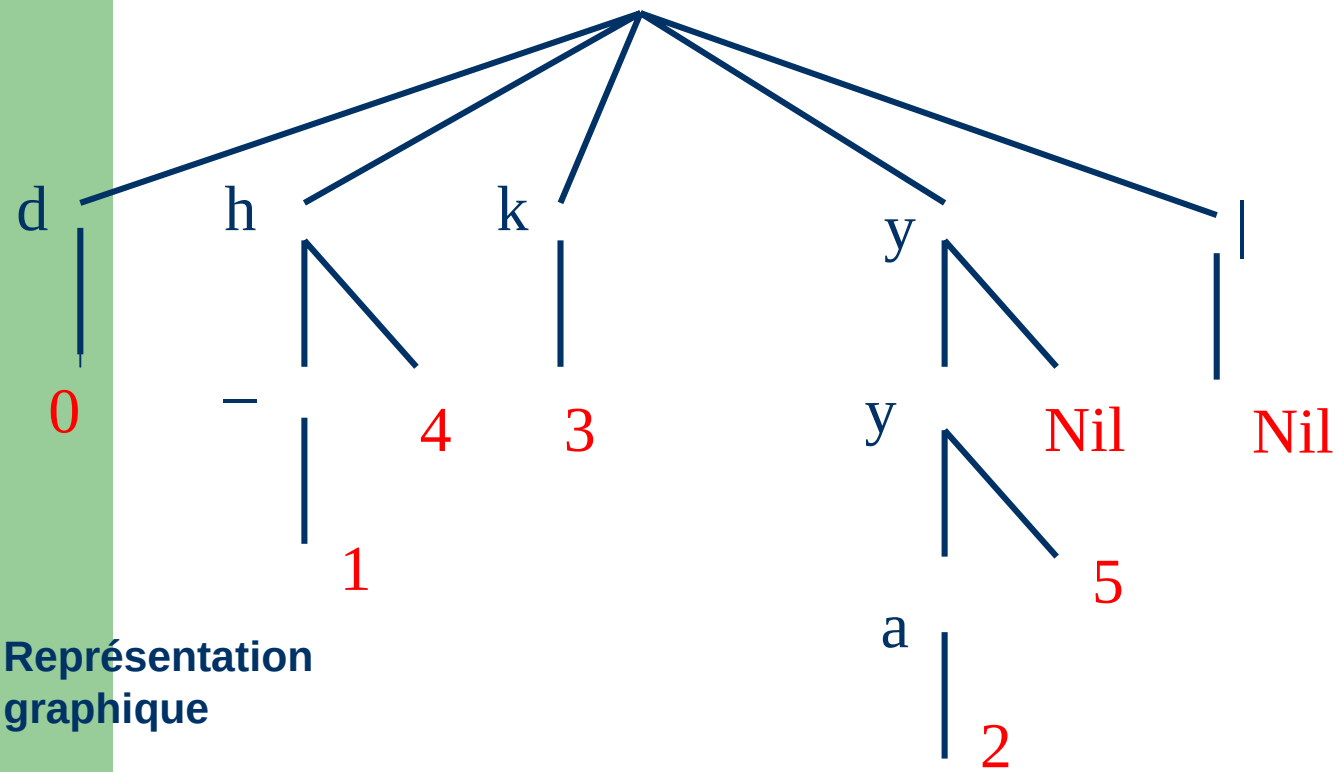
Le fichier :

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

Hachage Digital Compact Scalable et Distribué sans Multicast

Hachage digital compact

ARBRE : d 0 h _ 1 4 k 3 y y a 2 5 Nil | Nil



Représentation graphique

- **La concaténation des digits sur une branche de l'arbre représente la clé maximale de la case figurant dans la feuille**
- **forme préordre**
- **Suite de nœuds internes et externe**

Hachage Digital Compact Scalable et Distribué sans Multicast

Hachage digital Compact: Propriété et Performances

- **Les cases sont ordonnées de gauche à droite**

- **Algorithmes en mémoire :**
  - ✓ **Recherche : N/2 en moyenne (N : nombre de nœuds)**
  - ✓ **Insertion : N/2 décalages en moyenne**
  - ✓ **Encombrement : 3 octets / case en moyenne.**

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

Hachage Digital Compact Scalable et Distribué sans Multicast

Concepts

- **Au niveau de chaque serveur il y a**
  - ✓ **un arbre digital partiel**
  - ✓ **une case contenant les articles du fichier**
  - ✓ **un intervalle primaire [Min_1, Max_1]**
  - ✓ **Éventuellement une liste d'intervalles secondaires [Min_i, Max_i], $i \geq 2$**

- **L'arbre digital au niveau du serveur garde la trace de tous les éclatements sur ce serveur.**

Hachage Digital Compact Scalable et Distribué sans Multicast

Concepts

- **Au niveau de chaque client il y a un arbre digital partiel**
- **Les noeuds Nil au niveau de l'arbre du client sont représentés par un numéro de serveur négatif (réference un intervalle secondaire à l'intérieur du serveur)**
- **Tout client commence avec un arbre vide (| 0)**
- **Mise à jour progressive de l'arbre du client par un algorithme d'ajustement**

Hachage Digital Compact Scalable et Distribué sans Multicast

Concepts

- **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 (Scalabilité) .**

- **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**

✓**Les intervalles sont mis à jours**

✓**Création éventuelle de plusieurs intervalles secondaires (Nœuds Nil)**

(Le processus d'éclatement est donné plus loin)

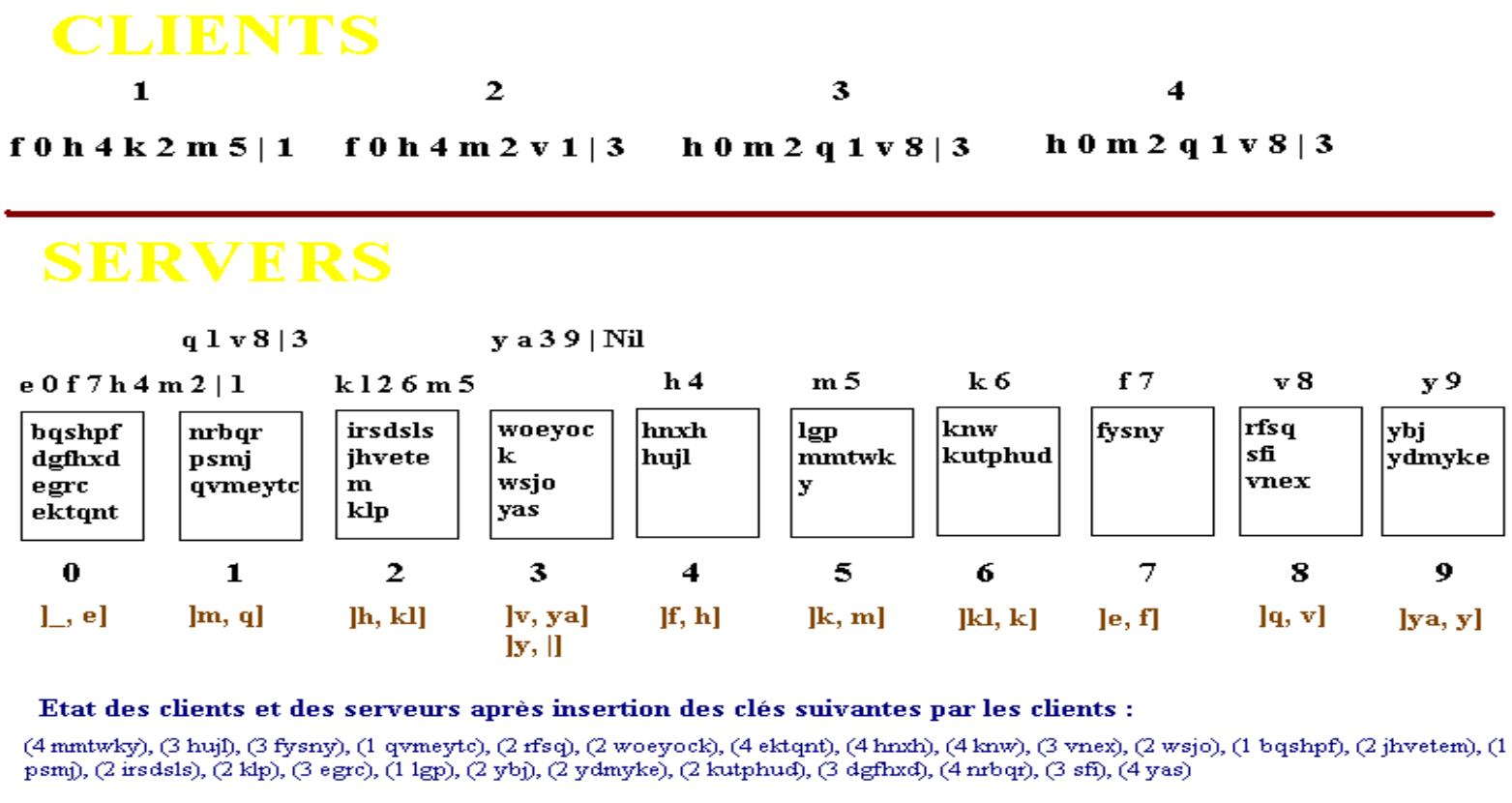
- **Initialisation du système**

Initialiser le serveur 0 avec

Case vide ; Intervalle : > λ , <= Λ ; Arbre : | 0

Hachage Digital Compact Scalable et Distribué sans Multicast

Illustration



Etat des clients et des serveurs après insertion des clés suivantes par les clients :

(4 mmtwky), (3 hujl), (3 fysny), (1 qvmeytc), (2 rfsq), (2 woeyock), (4 ektqnt), (4 hnxh), (4 knw), (3 vnex), (2 wsjo), (1 bqshpf), (2 jhvetem), (1 psmj), (2 irsdsls), (2 klp), (3 egrc), (1 lgp), (2 ybj), (2 ydmyke), (2 kutphud), (3 dgfhxd), (4 nrbqr), (3 sfi), (4 yas)

Hachage Digital Compact Scalable et Distribué sans Multicast

Transformation (Client, Clé) → (Serveur, Rang)

▪**1 Appliquer l'algorithme de transformation sur l'arbre du client. Soit m le serveur trouvé (négatif si Nil) et Cm sa clé maximale**

▪**2 Si m < 0, (cas d'un Nil), il s'agit d'un intervalle secondaire à l'intérieur du serveur -m.**

Si pas d'ajustement encore (Par la transformation en cours)

(i) Redirection vers le serveur –m.

(ii) S'il existe dans l'arbre du serveur -m un serveur s' correspondant à Cm, cela veut dire que le Nil a été remplacé par s' et donc la transformation continue dans ce serveur s' (étape 3 avec m =s'). s' remplace alors -m au niveau du client .

(iii) Dans le cas où s' n'existe pas, Retourner (-m, Rang). Rang est l'intervalle secondaire dans le serveur -m ayant Cm comme limite supérieure

Si ajustement : (Par la transformation en cours) : Retourner (m', Rang)

▪**3 Si m >0, Il s'agit de l'intervalle primaire du serveur m, alors**

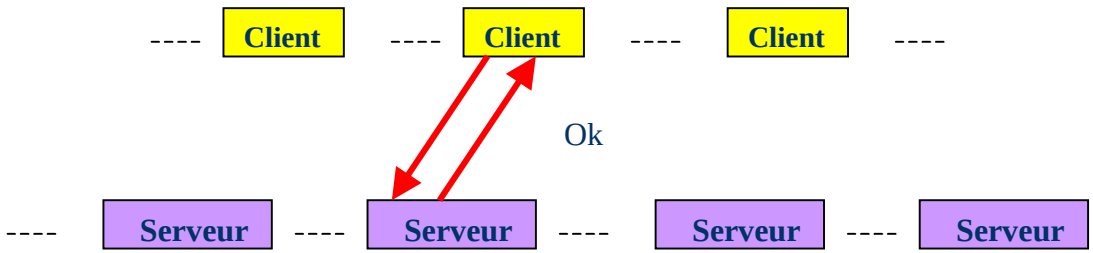
si la clé recherchée est dans cette intervalle alors Retourner (m, 1).

Autrement, Sauver m dans m', Appliquer l'algorithme de l'ajustement de l'arbre du client en fonction de l'arbre du serveur et recommencer à partir de 1

Hachage Digital Compact Scalable et Distribué sans Multicast

Transformation (Client, Clé) → Serveur

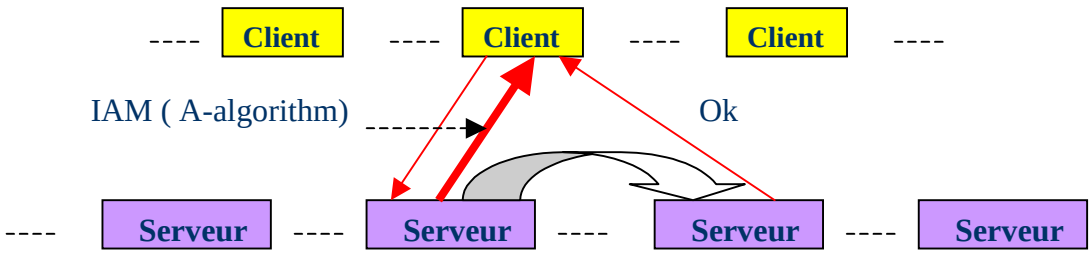
- **Bon adressage**



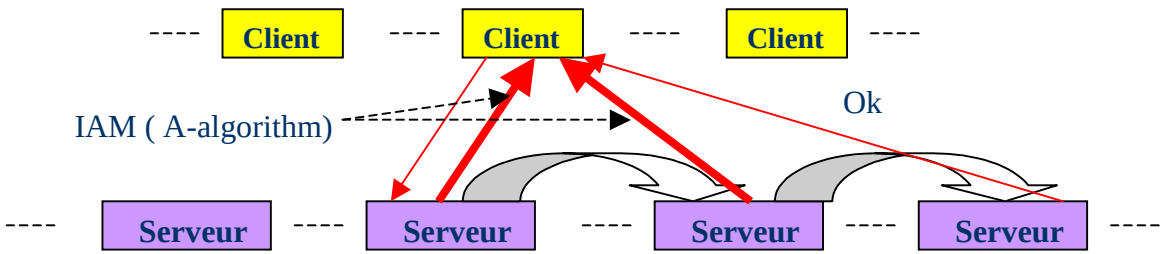
Hachage Digital Compact Scalable et Distribué sans Multicast

Transformation (Client, Clé) → Serveur

Mauvais adressage (avec 1 forward)



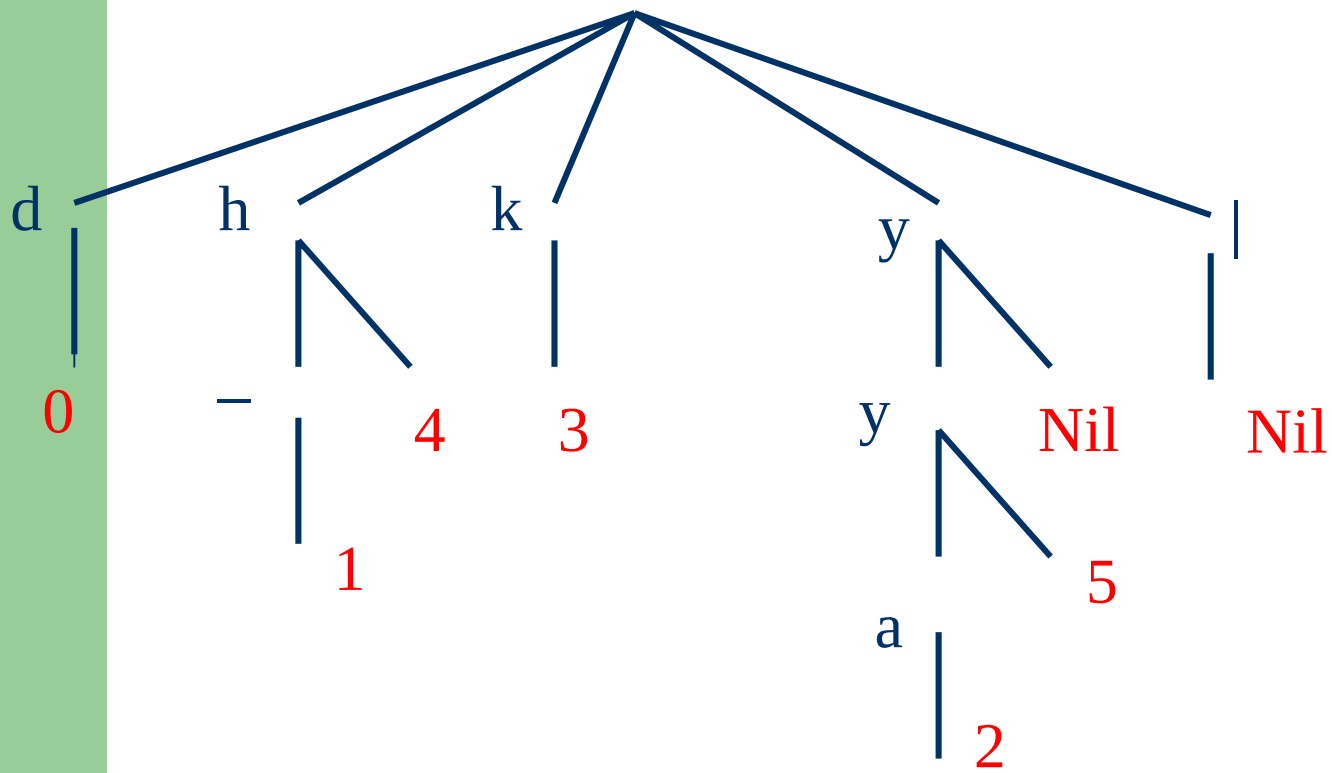
Mauvais adressage (avec 2 forward)



Hachage Digital Compact Scalable et Distribué sans Multicast

Terminologie

ARBRE : d 0 h _ 1 4 k 3 y y a 2 5 Nil | Nil



. Sous arbre de niveau donné

Niveau 0 : d0; h_14;

Niveau 1 : _1; ya25;

. Branches d'un sous arbre donné

Pour ya25 les branches sont : ya2 et y5

Si ya2 est une branche, Digit(ya2)=ya et Leaf(ya2)=2

→ Digits(branche d'un sous arbre de niveau 0) = clé maximale de Leaf(branche)

Hachage Digital Compact Scalable et Distribué sans Multicast

Ajustement de l'arbre du client : algorithme

Soit Cm = $C_0C_1...C_k$ la clé maximale du serveur trouvé (m)

(i) Determiner dans l'arbre du client le nombre (Ni) de noeuds qui precedent m.

Ne : = longueur (Cm) – Ni. Ne represente le nombre de digits qui existent au niveau de l'arbre du serveur . Ne = (k + 1) – Ni.

(ii)Determiner dans l'arbre du serveur toutes les branches b des sous arbres de niveau Ne tels que Digits(b) + '|' ≤ ($C_{Ne}C_{Ne+1},...C_k$) + '|' .

(iii) Remplacer dans l'arbre du client ($C_{Ne}C_{Ne+1},...C_k$) m par les branches trouvées

Hachage Digital Compact Scalable et Distribué sans Multicast

Ajustement de l'arbre du client : Exemple1

- **Client : … q m 16 q c 41 36 35 r e …**
- **Serveur 41 : q p h 41 238 q c 229 Nil Nil**

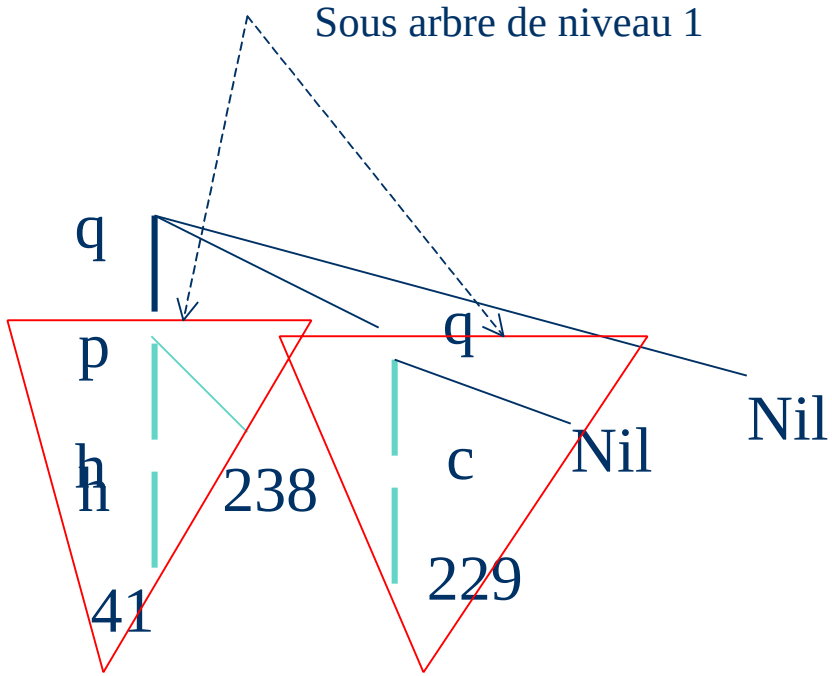
- **Ni = 2 ; Ne= 1 ; Cm='qqc' ; k = 2**
- **Les sous arbres de niveau Ne = 1 : 'p h 41 238' ; 'q c 229 Nil'**
- **Les digits des branches des sous arbres de niveau Ne=1 ≤ 'qc'+'|' : 'ph' + '|' , 'p'+'|', 'qc'+'|'**
- **Il ya remplacement de 'q c' 41 par 'p h 41 238 q c 229'**

- **L'arbre du client devient … q m 16 p h 41 238 q c 229 36 35 r e …**

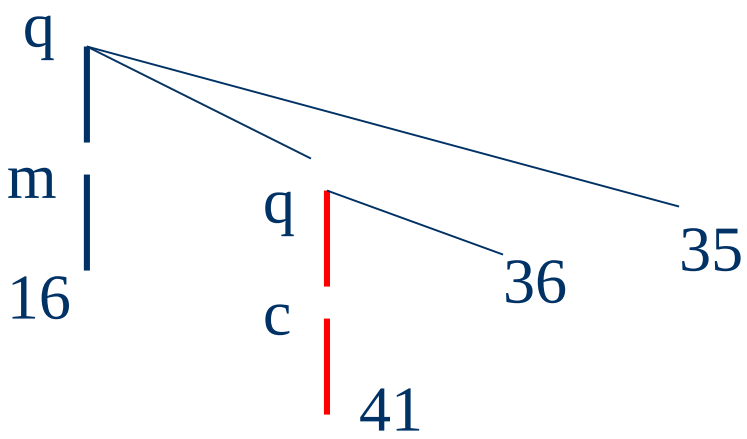
Hachage Digital Compact Scalable et Distribué sans Multicast

Ajustement de l'arbre du client : Exemple1

- Serveur 41



Client



Cm='qqc'; k=2;
Ni=2; Ne=1

Hachage Digital Compact Scalable et Distribué sans Multicast

Ajustement de l'arbre du client : Exemple2

- **Client : … t b … m 111 70 u f ...**
- **Serveur 70 : t p 70 x 211 y 147 141**

- **Ni = 0 ; Ne= 1 ; Cm='t' ; k = 0**
- **Sous arbres de niveau Ne = 1 : 'p 70' ; 'x 211' ; 'y 147' ; '141'**
- **Les branches des sous arbres de niveau Ne=1 ≤ '|' : 'p' + '|' , 'x'+'|', 'y'+'|', '|'**
- **Il y a remplacement de 70 par 'p 70 x 211 y 147 141'**

- **L'arbre du client devient … t b … m 111 p 70 x 211 y 147 141 u f ...**

Hachage Digital Compact Scalable et Distribué sans Multicast

Ajustement de l'arbre du client : : Exemple3

- **Client : … i g 3 u 33 18 j …**
- **Serveur 33 : i t 33 u 64 Nil**

- **Ni = 1 ; Ne= 1 ; Cm='iu' ; k = 1**
- **Sous arbres de niveau Ne = 1 : 't 33' ; 'u 64' ; 'Nil'**
- **Branches des sous arbres de niveau Ne=1 ≤ 'u|' : 't' + '|' , 'u'+'|'**
- **Il y a remplacement de 'u 33' by 't 33 u 64'**

- **L'arbre du client devient … i g 3 t 33 u 64 18 j …**

Hachage Digital Compact Scalable et Distribué sans Multicast

Insertion

Appliquer l'algorithme de transformation (Client, cle) → (m, Rang)

Rang <> 1 (cas du Nil)

(i) Créer un nouveau serveur, N.

(ii) Initialiser ce serveur avec une case contenant la nouvelle clé, un arbre vide et le même intervalle que celui associé à Rang

(iii) Ajuster l'arbre du serveur en remplaçant Nil par N avec élimination du serveur secondaire Rang

(iv) Ajuster l'arbre du client en remplaçant –m par N

Hachage Digital Compact Scalable et Distribué sans Multicast

Insertion

Rang = 1 (Cas d'un serveur)

(i)Si Clé n'est pas dans la case 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**

Hachage Digital Compact Scalable et Distribué sans Multicast

Éclatement

- **Former la séquence et déterminer la séquence de division Seq**
- **Éclater la case du serveur I en 2 selon Seq**
- **l'ancien serveur Ancien contient les clés <= Seq**
- **le nouveau serveur, soit Nouveau, contient le reste**

  Int(Ancien) ←] Min(Ancien) , Seq]

  Int(Nouveau) ←] Seq , Seq_{-1}]

  l'arbre du serveur nouvellement créé est vide

Seq_{-j} c'est Seq sans les j derniers digits

- **Générer (K- I – 1) intervalles secondaires correspondant à chaque Nil :**

(K:nombre de digits de Seq et I:nombre de digits qui existent déjà dans l'arbre)

]Sq_{-1}, Sq_{-2}]

]Sq_{-2}, Sq_{-3}]

…

]Sq_{-i}, Max(Ancien)]

- **Modifier l'arbre du serveur éclaté** (CTH local)

Hachage Digital Compact Scalable et Distribué sans Multicast

Requête à intervalle

Un client Client veut toutes les clés dans l'intervalle [X, Y]

1- Appliquer l'algo de transformation sur le client Client pour rechercher la clé X. Soit M le serveur trouvé et Cm sa clé maximale

2- Initialiser une pile avec Cm. (à raison d'un digit par élément)

3- Traverser l'arbre du client à partir du noeud M comme suit:

Si un digit est rencontré, il est empilé.

Si un noeud externe est rencontré, soit S ce noeud et Cm' sa clé maximale.

(i) Si Cm' ≤ Y alors si S > 0 alors Envoi_message à S contenant Cm' pour effectuer le traitement1.

Dépiler un digit de la pile et avancer dans l'arbre du client(cas : S > 0 et S < 0)

(ii) Si Cm' > Y alors Stop

Hachage Digital Compact Scalable et Distribué sans Multicast

Requête à intervalle

Un serveur S qui reçoit le message Cmax pour effectuer le traitement 1.

Traitement 1 :

- Lister toutes les clés de sa case ≤ Y

- S'il existe une clé ≥ Y alors Stop

- Traverser son arbre au moyen d'une pile . Pour tout serveur (nœud externe) S' # S tel que sa clé maximale ≤ Cmax faire Envoi d'un message au serveur S pour effectuer le traitement 2.

Hachage Digital Compact Scalable et Distribué sans Multicast

Requête à intervalle

Un serveur Serveur qui recoit un message pour effectuer le traitement2

Traitement 2 :

- Lister toutes les clés dans sa case ≤ Y

- S'il existe une clé ≥ Y alors Stop

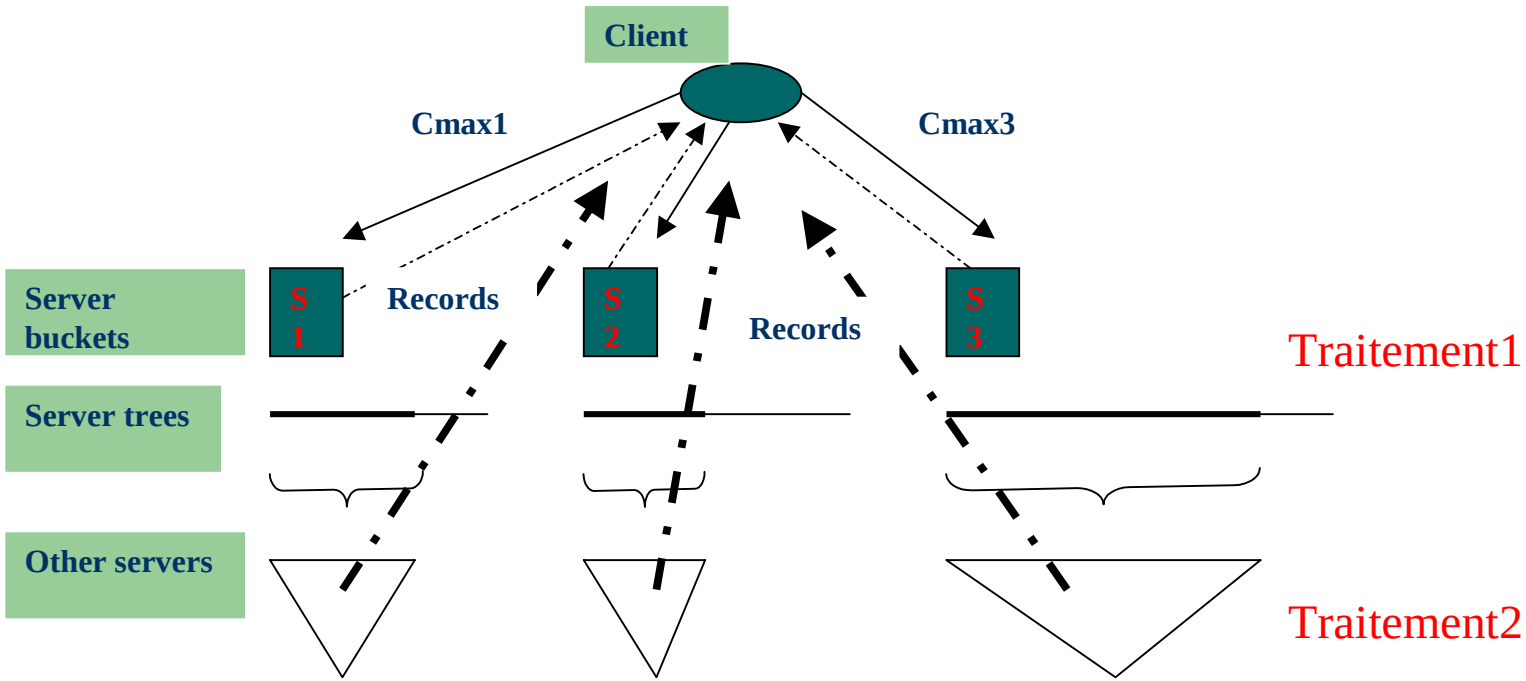
- Parcours sequentiel de l'arbre du serveur Serveur avec utilisation d'une pile

- **Si un digit est rencontré il est empilé**
- **Si un serveur ou Nil est rencontré, la pile contient la clé maximale (concatenation des digits). Si c'est un serveur S, Envoi un message à S pour effectuer le traitement2, puis dépilement**

Hachage Digital Compact Scalable et Distribué sans Multicast

Requête à intervalle

-



Hachage Digital Compact Scalable et Distribué sans Multicast

Propriétés

- Les intervalles des serveurs (primaires et secondaires) engendrent une partition dans l'ensemble des clés.

- A partir d'un serveur avec l'intervalle]bi, bs], on peut retrouver tous les serveurs dont les intervalles se suivent dans la partition.

- A partir du serveur 0, on peut retrouver l'arbre réel du système.

- tout client couvre toute la partition.

Hachage Digital Compact Scalable et Distribué sans Multicast

Test

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 %

Hachage Digital Compact Scalable et Distribué sans Multicast

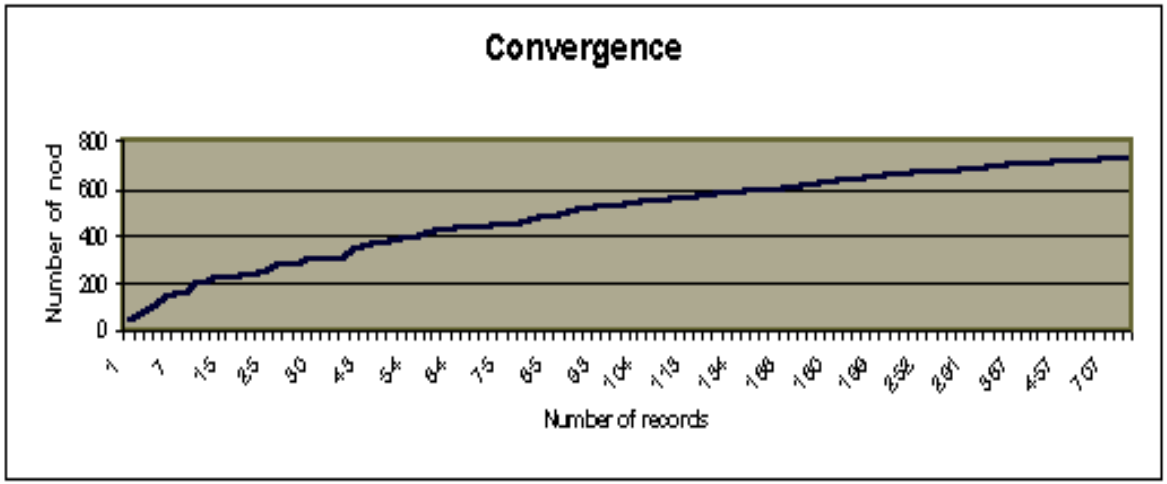
Test

```
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 %
```

Hachage Digital Compact Scalable et Distribué sans Multicast

Test



Très rapide pour les premières insertions, ensuite ça se stabilise à un moment donné.

Hachage Digital Compact Scalable et Distribué sans Multicast

Comparaison avec RP*S

Client : **RP*s : table et CTH* : arbre compact. Avantage 3 octets pour adresser un serveur**

- **Serveur : RP*s : 2 types de serveur : données et index; CTH*: 1 type de serveur**
- **En cas d'erreur d'adressage: RP*S: parcours des nœuds ascendants et descendants; CTH* : parcours des nœuds descendants**
- **Éclatement : RP*S : plus complexe (Principe des B-arbres) → modification de plusieurs serveurs; CTH* : simple (pas de cascade) → modification d'un seul serveur**
- **Ajustement de l'image du client :RP*S pas toujours; CTH* toujours. Algorithme plus simple dans CTH*.**
- **Opérations (Recherche, Requête à intervalle, ..) : RP*S : montée et descente dans le B-arbre; CTH* : uniquement descente dans les serveurs descendants**

Hachage Digital Compact Scalable et Distribué sans Multicast

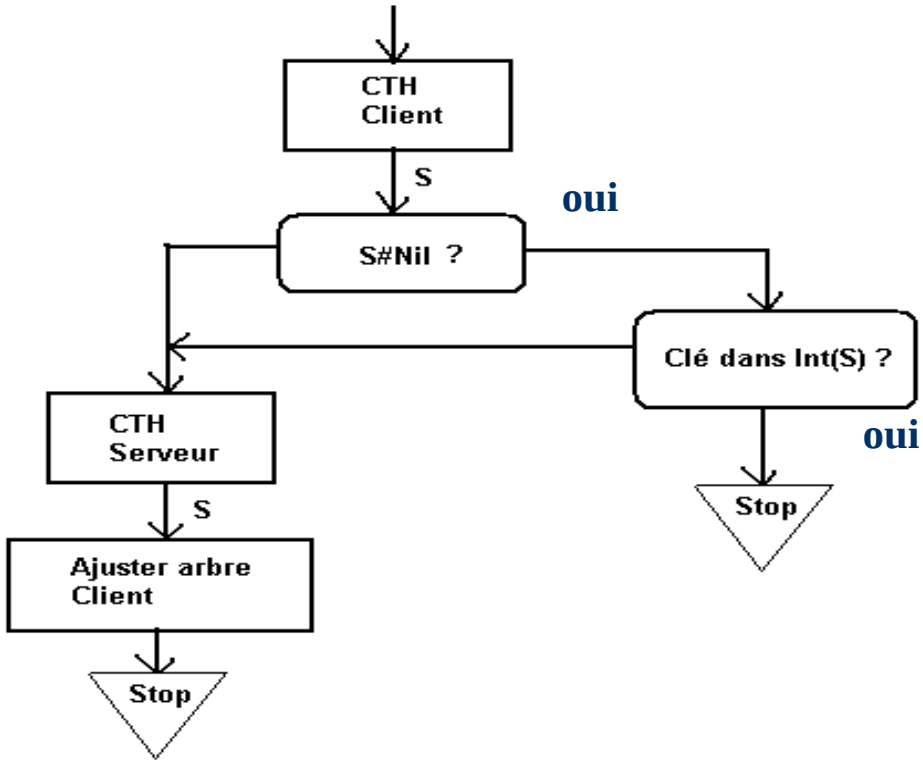
Variante 1 : CTH* avec arbre central

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

Remarque : si on découpe l'arbre sur plusieurs serveurs, on trouve un schéma similaire à celui de RP*S.

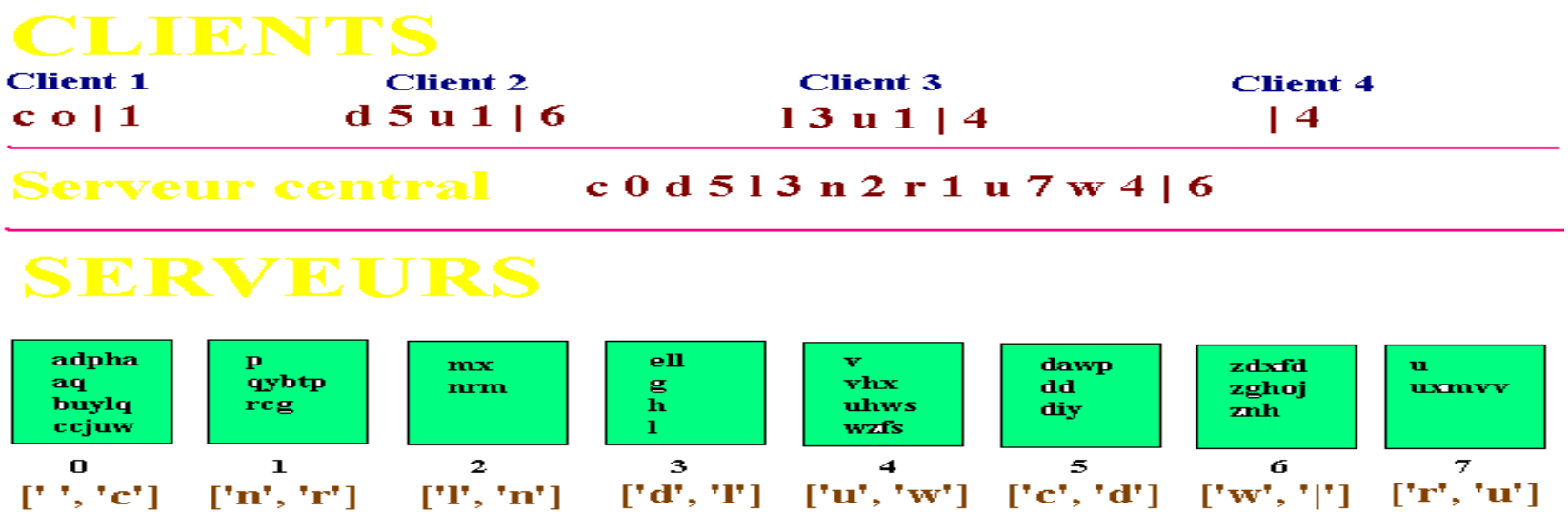
Hachage Digital Compact Scalable et Distribué sans Multicast

Variante 1 : CTH* avec arbre central



Hachage Digital Compact Scalable et Distribué sans Multicast

Variante 1 : CTH* avec arbre central



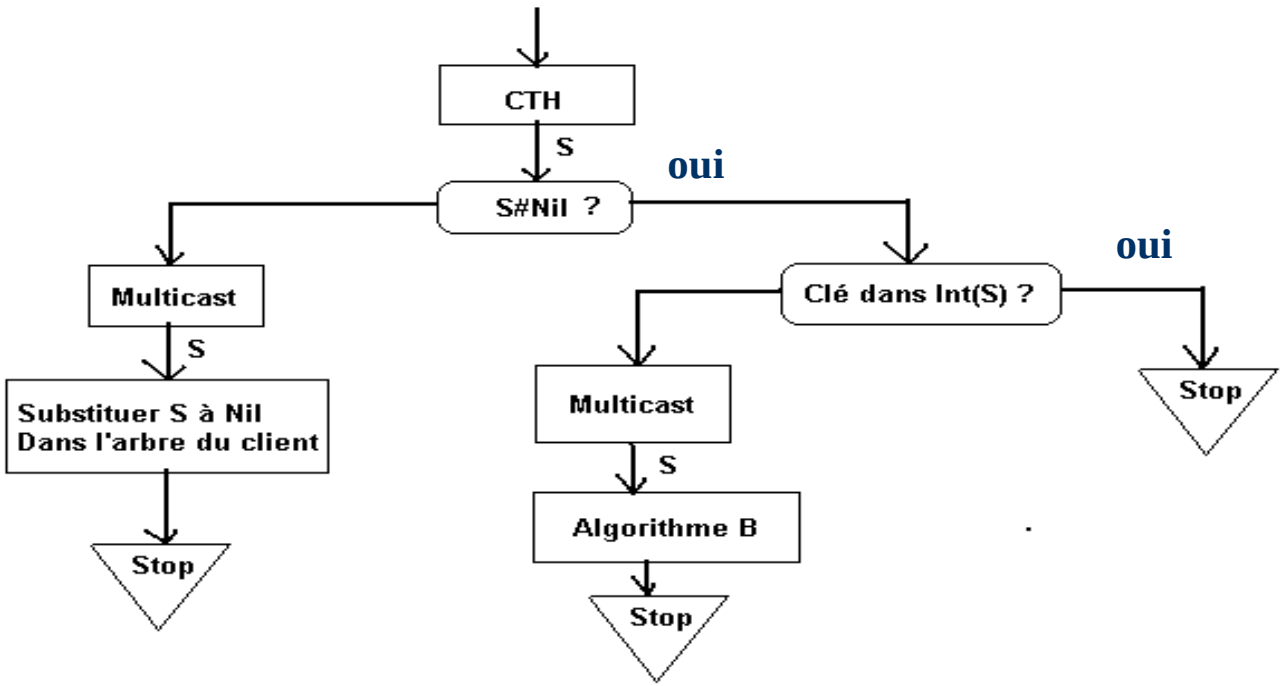
Hachage Digital Compact Scalable et Distribué sans Multicast

Variante 2 : CTH* avec Multicast intense

- **Pas d’arbre au niveau des serveurs**
- **Pas d’arbre central**

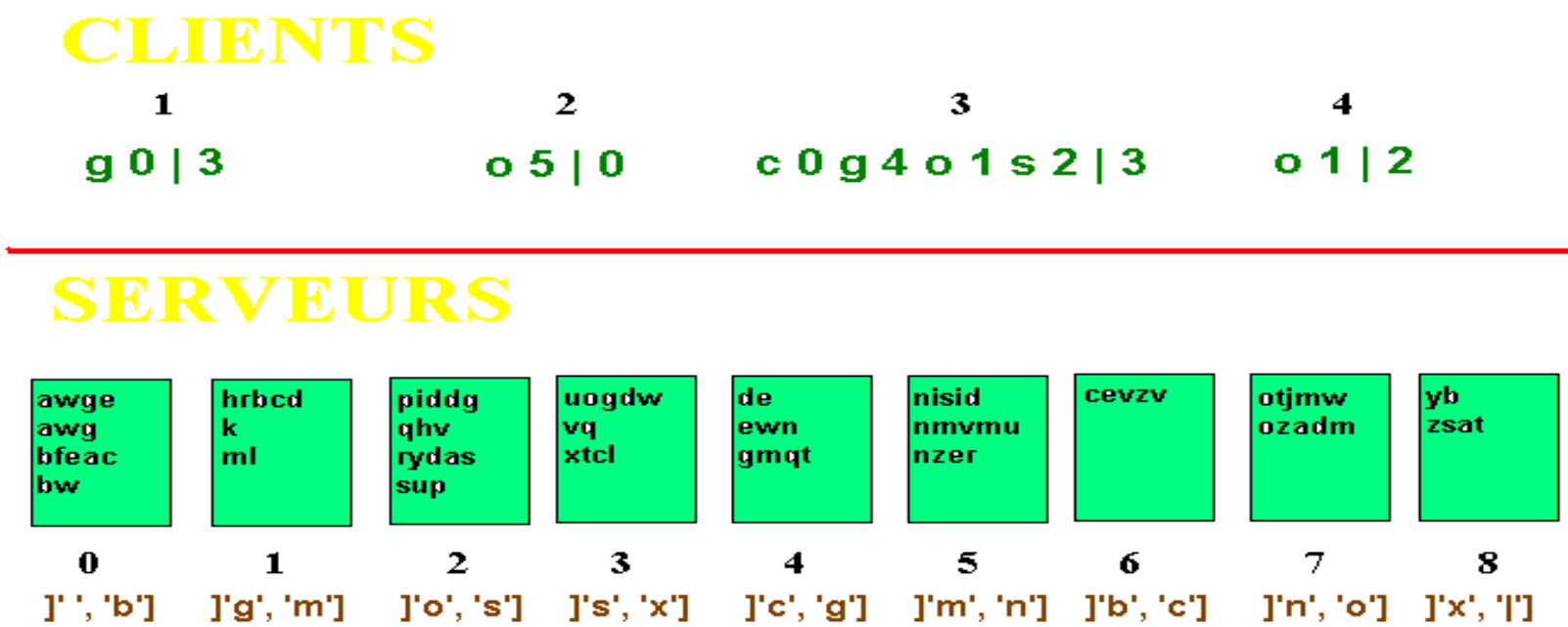
Hachage Digital Compact Scalable et Distribué sans Multicast

Variante 2 : CTH* avec Multicast intense



Hachage Digital Compact Scalable et Distribué sans Multicast

Variante 2 : CTH* avec Multicast intense



Hachage Digital Compact Scalable et Distribué sans Multicast

Conclusion

- **Généralisation du hachage digital pour les environnements distribués**
- **Préservation de l'ordre des articles : facilite les opérations de parcours séquentiel et de requêtes à intervalle.**
- **Toutes les opérations sont réalisées sans multicast**
- **Résultats de tests :**
  - le nombre moyen de forward est faible (0.4),
  - Quelques nœuds (4 à 6) sont transférés d'un serveur vers un client pour la mise à jour de son image.
  - 3 octets pour adresser un serveur
- **Un protocole de communication existe (Plate forme Linux)**
- **Un protocole de communication en cours (Plate forme Windows)**

- **Travaux futurs**
  - Toutes les variantes de LH* peuvent s'intégrer dans CTH*.
  - Test en cours dans un environnement réel
  - Comparaison approfondie avec RP*S et les autres SDDS.