Deterministic Scheme to Distributed Trie Hashing

D.E ZEGOUR,

Institut National d'Informatique, Alger

Trie hashing (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

Compact trie hashing (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.

Deterministic Scheme to Distributed Trie Hashing

- 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)

  - Pas de site maître

  - Aucun dialogue entre les clients.

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 et suppression

- Variantes
- Conclusion

Compact Trie Hashing

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

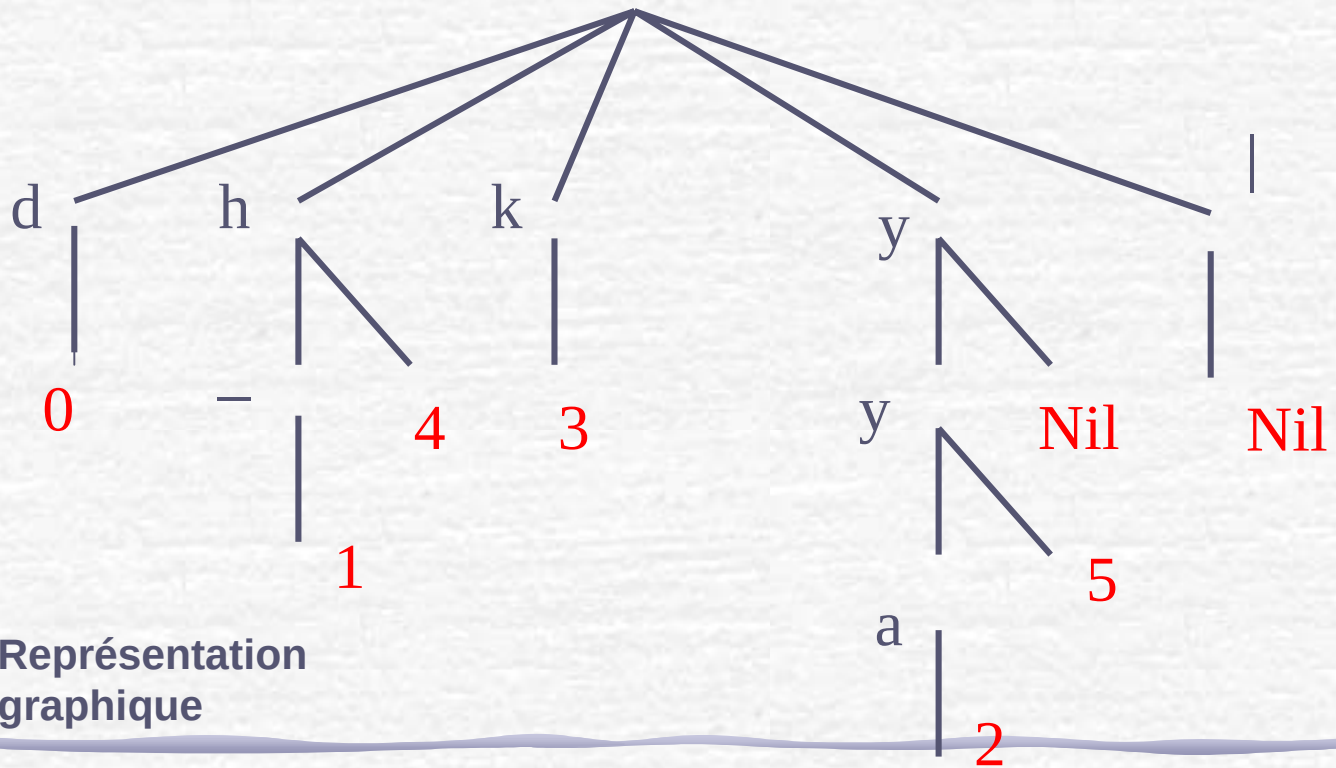
L'arbre : d 0 h _ 1 4 k 3 y y a 2 5 Nil | Nil

Le fichier :

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

Compact Trie Hashing

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**

Compact Trie Hashing

Propriété et Performances

- 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

Compact Trie Hashing

Modification

- **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

Compact Trie Hashing

Modification

- **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

Compact Trie Hashing

Modification

Éclatement (deuxième manière)

1. Éclater m normalement

2. Si la séquence de division correspond à la clé max d'un nœud Nil,

✓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

Compact Trie Hashing

Comparaison

Éclatement (première manière)

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

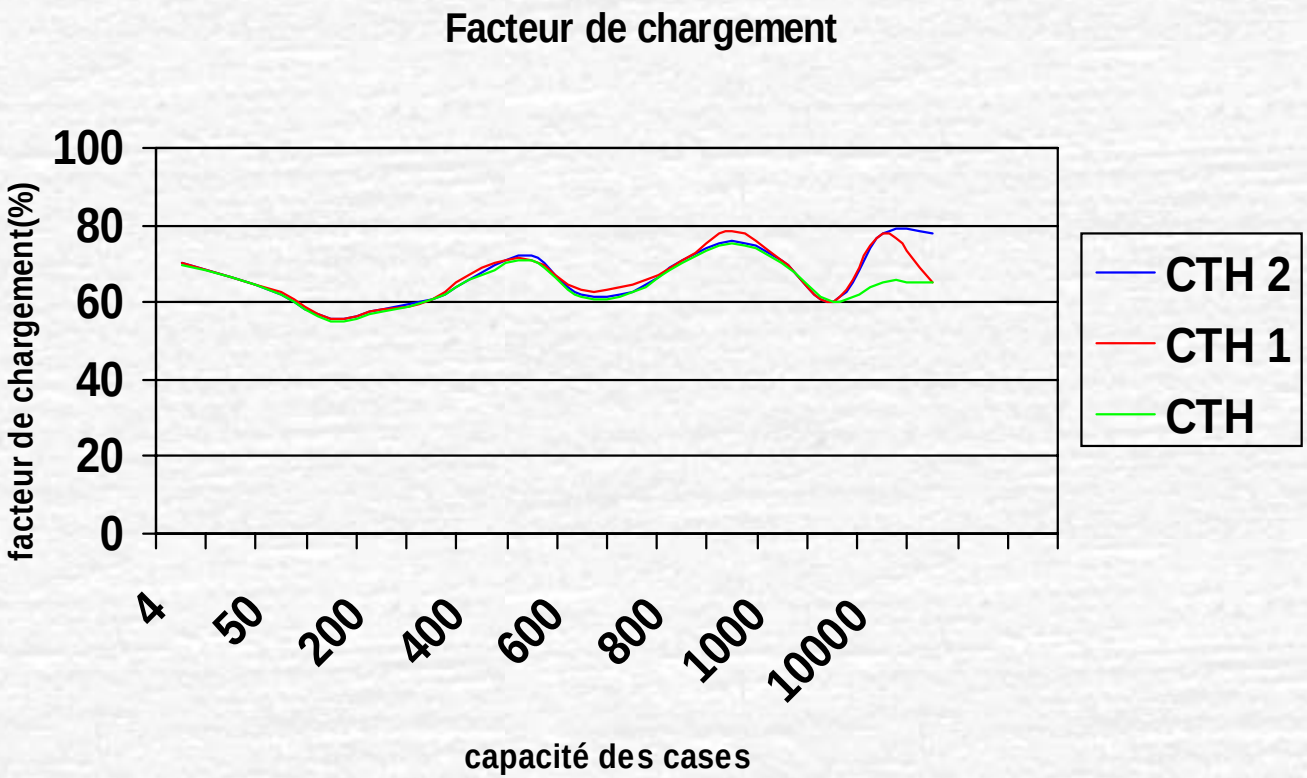
Éclatement (deuxième manière)

- ✓**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.**

Compact Trie Hashing

Comparaison



Deterministic Scheme to Distributed Trie Hashing

Taux de nœuds Nil

- **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

Deterministic Scheme to Distributed Trie Hashing

Concepts

- Au niveau de chaque serveur il y a

  AVANT

  - ✓ un arbre digital partiel
  - ✓ une case contenant les articles du fichier
  - ✓ un intervalle primaire $[Min_1, Max_1]$
  - ✓ Éventuellement une liste d'intervalles secondaires $[Min_i, Max_i]$, $i \geq 2$

  - ✓ **MAINTENANT**
  - ✓ 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.

Deterministic Scheme to Distributed Trie Hashing

Concepts

- Au niveau de chaque client il y a un arbre digital partiel

AVANT

- 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.

Deterministic Scheme to Distributed Trie Hashing

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

AVANT

✓Les intervalles sont mis à jours

✓Création éventuelle de plusieurs serveurs virtuels (Nœuds Nil)

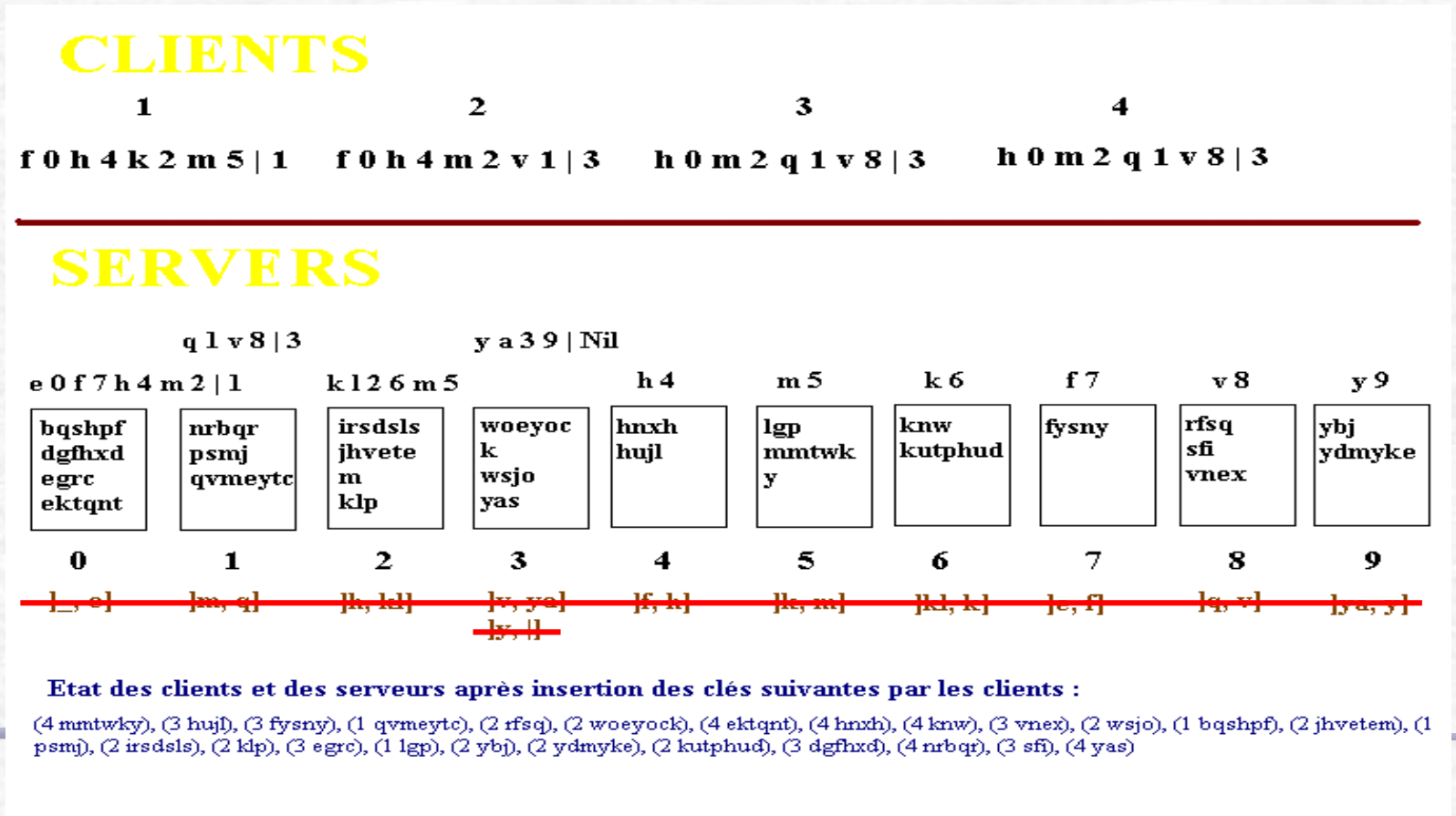
Initialisation du système

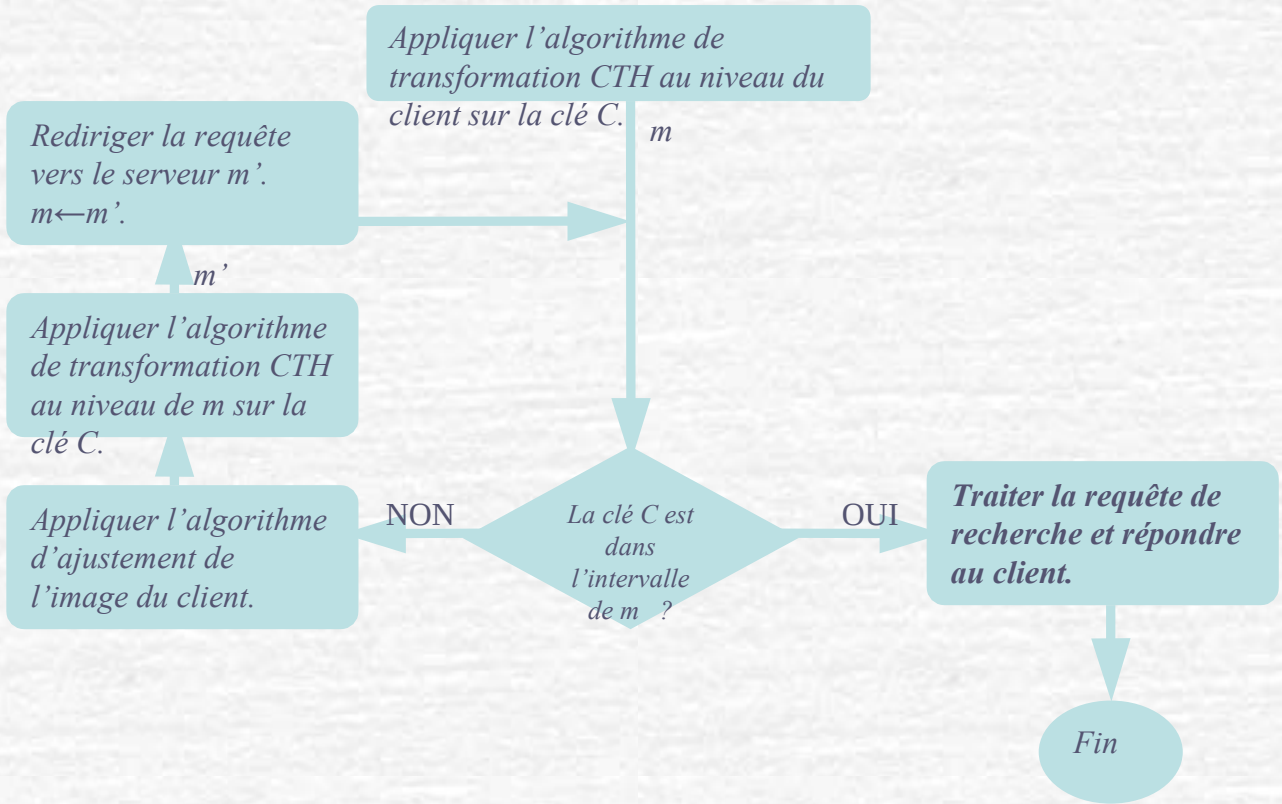
- Initialiser le serveur 0 avec

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

Deterministic Scheme to Distributed Trie Hashing

Illustration



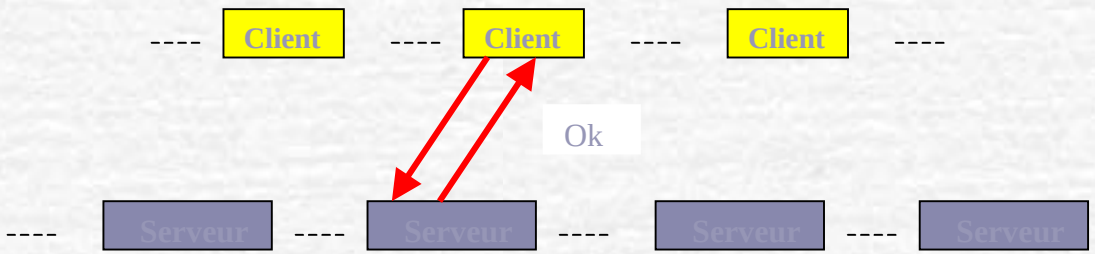


Transformation (Client,clé) → Serveur

Deterministic Scheme to Distributed Trie Hashing

Transformation (Client, Clé) → Serveur

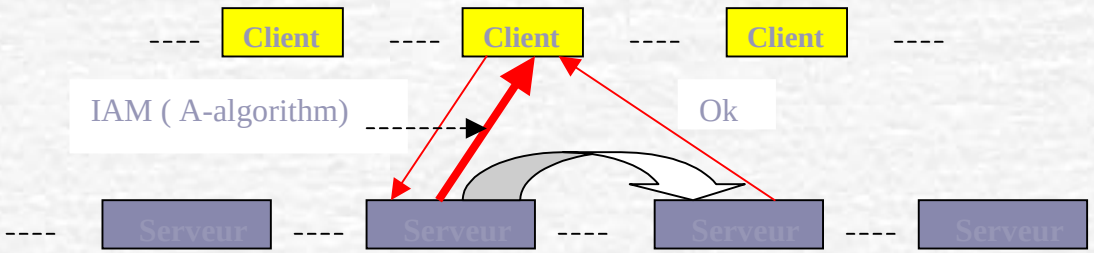
- Bon adressage



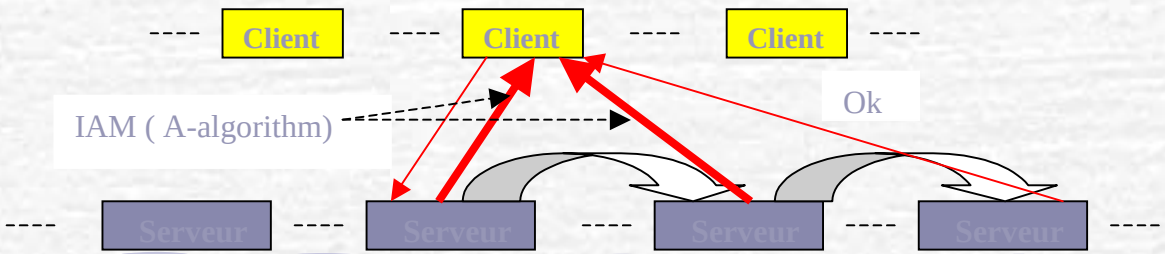
Deterministic Scheme to Distributed Trie Hashing

Transformation (Client, Clé) → Serveur

Mauvais adressage (avec 1 forward)



Mauvais adressage (avec 2 forward)



Deterministic Scheme to Distributed Trie Hashing

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 Digit (b) +'|\|...|' ≥ CM, $b=b_1\ b_2\ ...b_{Taille(b)}$***

- ***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 Digit (b)

- m' est le serveur qui correspond à CM' dans l'arbre du serveur

Deterministic Scheme to Distributed Trie Hashing

Ajustement de l'arbre du client : Exemple1

- 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

Deterministic Scheme to Distributed Trie Hashing

Ajustement de l'arbre du client : Exemple2

- Arbre du client : b a b a a a 0 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 0 Nil 1 Nil Nil Nil | 2

Deterministic Scheme to Distributed Trie Hashing

Ajustement de l'arbre du client :Exemple3

- Arbre du client : a 0 b a a b a 1 Nil Nil 3 Nil | 2
- Serveur 0 : a a a 0 6 5 | 1
-
- Données client : m=0 et CM'= 'a'
- Données serveur : CM='aaa' ; m'=5 ; P = a a a 0 6 5
-
- 1) → b ='a0'
- 2) M = 1
- 3) b_{M+1} *est 0* → a a a 0 6 5 b a a b a 1 Nil Nil 3 Nil | 2

Deterministic Scheme to Distributed Trie Hashing

Insertion

(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

Deterministic Scheme to Distributed Trie Hashing

Éclatement d'un serveur

Soit m le serveur éclaté et m' le nouveau serveur à créer, CM de longueur k la clé maximale de m avant l'éclatement.

Principe :

1) Appliquer l'algorithme d'éclatement de CTH pour modifier l'arbre de m et partager les clés entre m et m' ;

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

Deterministic Scheme to Distributed Trie Hashing

Requête à intervalle (Coté Client)

Appliquer l'algorithme de recherche sur la clé clé_min soit m le serveur trouvé et CM sa clé maximale ;

Si (clé_max = CM) : Appliquer au niveau de serveur m REQ [clé_min, clé_max] ;

Sinon Appliquer au niveau de serveur m REQ[clé_min, CM] ; Fin Sinon ;

Tant que (clé_max > CM) Faire

CM=suivant (CM) ; m=prochain (m) ;

Si (clé_max = CM) Alors;

Appliquer au niveau de serveur m REQ [clé_min, clé_max] ;

Sinon

Appliquer au niveau de serveur m REQ [clé_min, CM] ;

Deterministic Scheme to Distributed Trie Hashing

Requête à intervalle(Coté serveur)

Retrouver m' le serveur de clé maximale CM qui doit contenir clé_min

Si (m' <> m)

Appliquer au niveau du serveur m' REQ [clé_min, clé_max] ;

Envoyer un IAM au client ;

Sinon

Si (clé_max = CM) Alors

Retourner les clés du serveur m de l'intervalle [clé_min, clé_max]

Sinon

Retourner les clés du serveur m de l'intervalle [clé_min, CM]

Envoyer un IAM au client ;

Deterministic Scheme to Distributed Trie Hashing

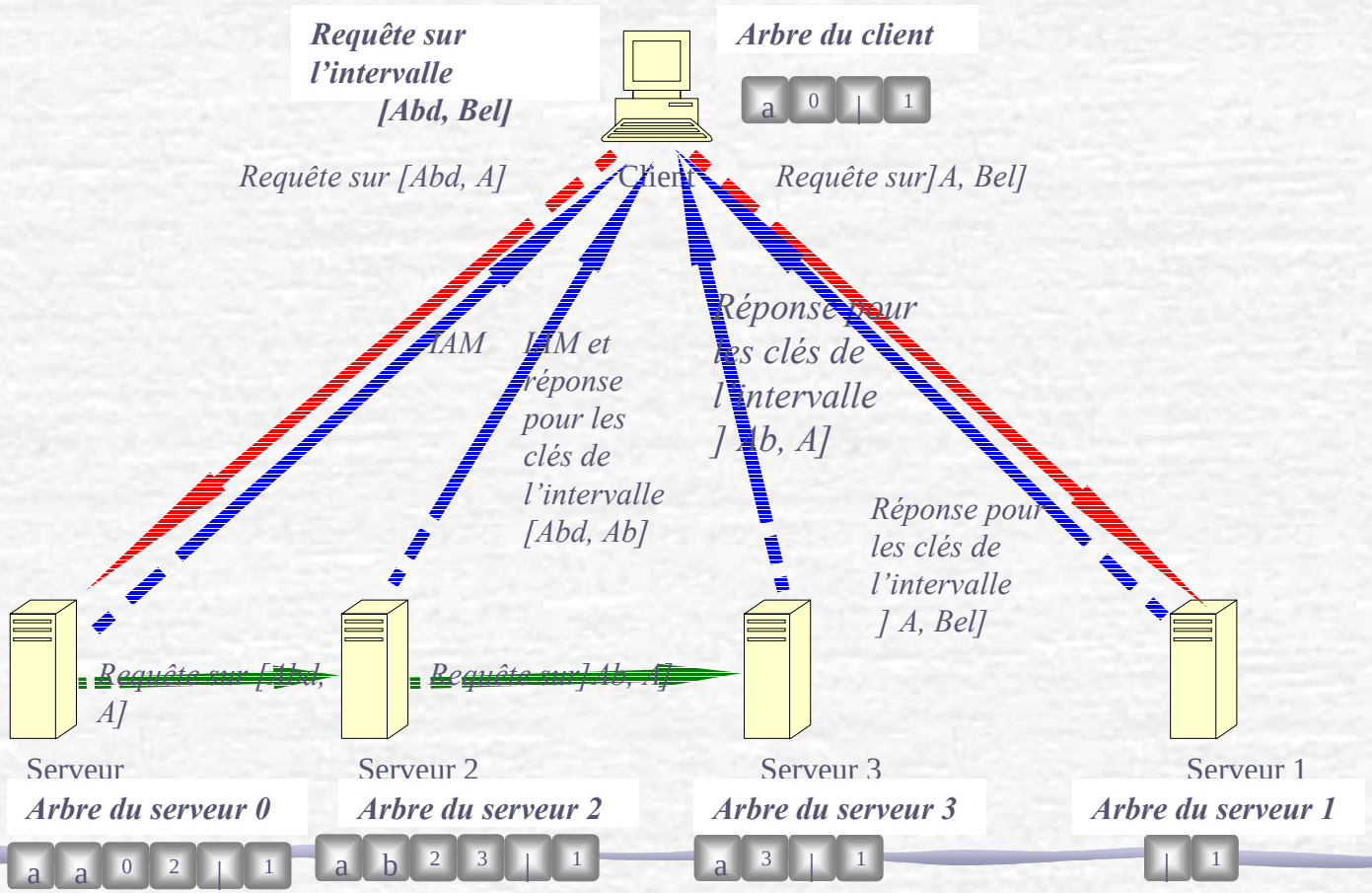
Requête à intervalle(Coté serveur)

(Suite)

```
Tant que (clé_max > CM) Faire
    CM=suivant (CM) ; m=prochain (m) ;
    Si (clé_max = CM)
        Appliquer au niveau du serveur m REQ [clé_min, clé_max] ;
     Sinon
        Appliquer au niveau du serveur m REQ  [clé_min, CM] ;
```

Deterministic Scheme to Distributed Trie Hashing

Requête à intervalle (Concret)



Deterministic Scheme to Distributed Trie Hashing
Performance

Méthode probabiliste

Utilisation d'un time out

Méthode déterministe

Chaque serveur recevant la requête envoie au client les enregistrements qui répondent à la requête, en plus il envoie son intervalle

Le client attend donc que l'union des intervalles des serveurs répondants recouvre l'intervalle initial de la requête.

Deterministic Scheme to Distributed Trie Hashing

Performance

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 %

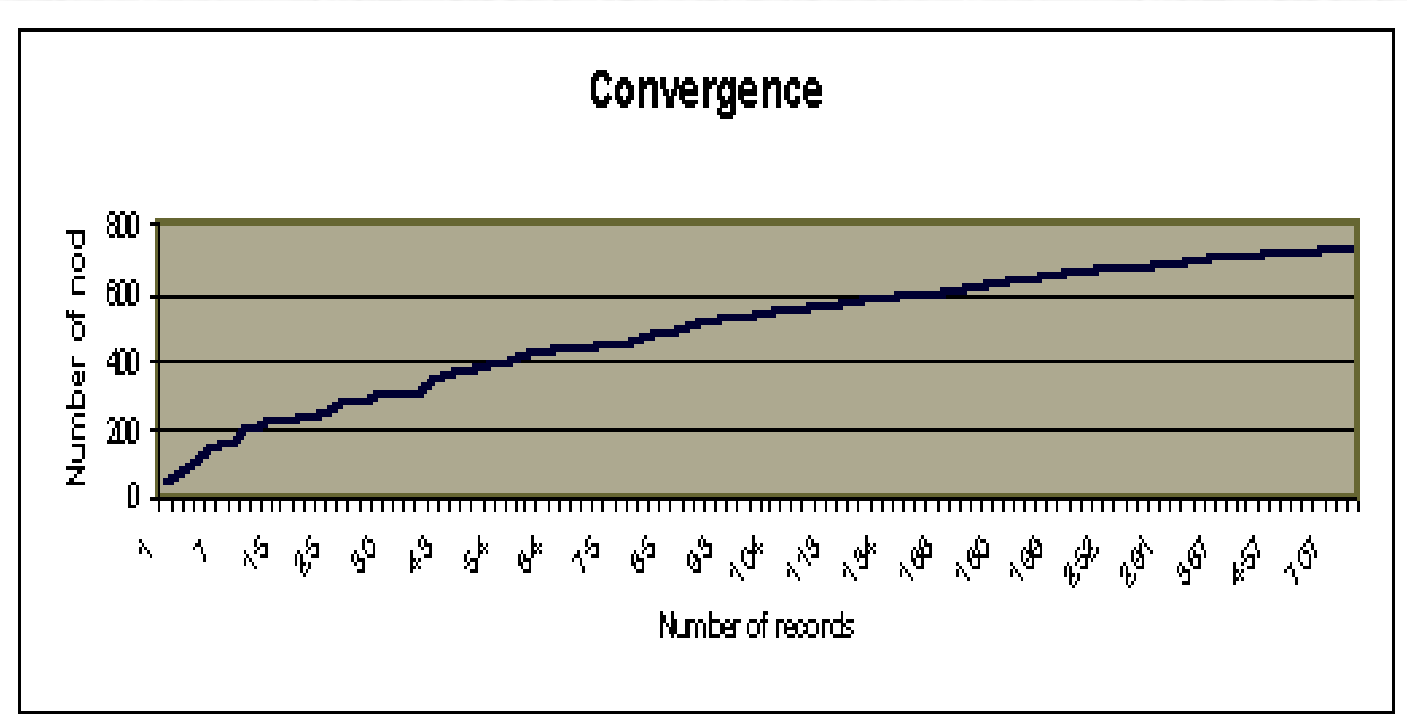
Deterministic Scheme to Distributed Trie Hashing
Performance

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 %

Deterministic Scheme to Distributed Trie Hashing

Performance



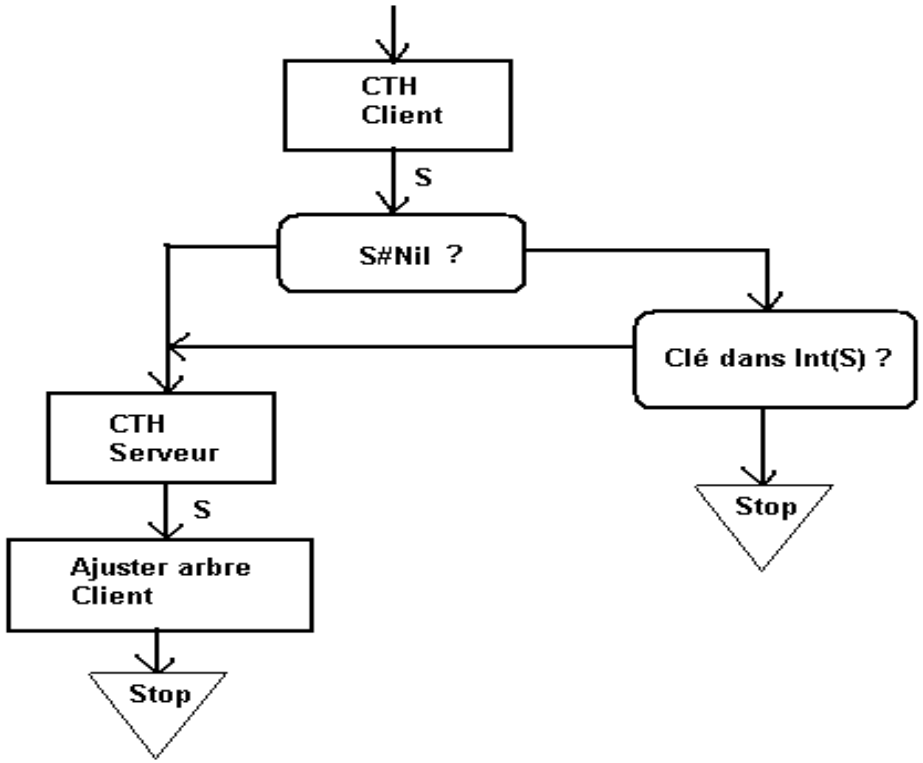
Deterministic Scheme to Distributed Trie Hashing
Variantes

CTH* avec arbre central

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

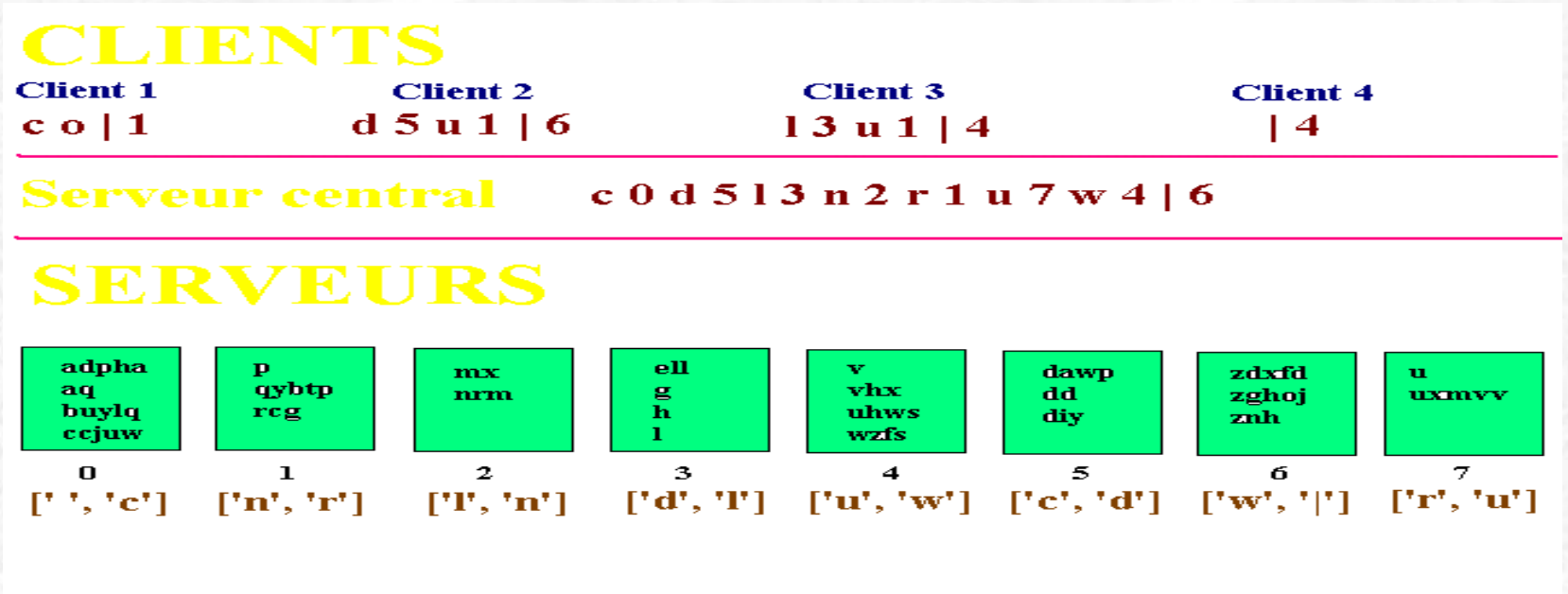
Deterministic Scheme to Distributed Trie Hashing

Variantes



Deterministic Scheme to Distributed Trie Hashing

Variantes



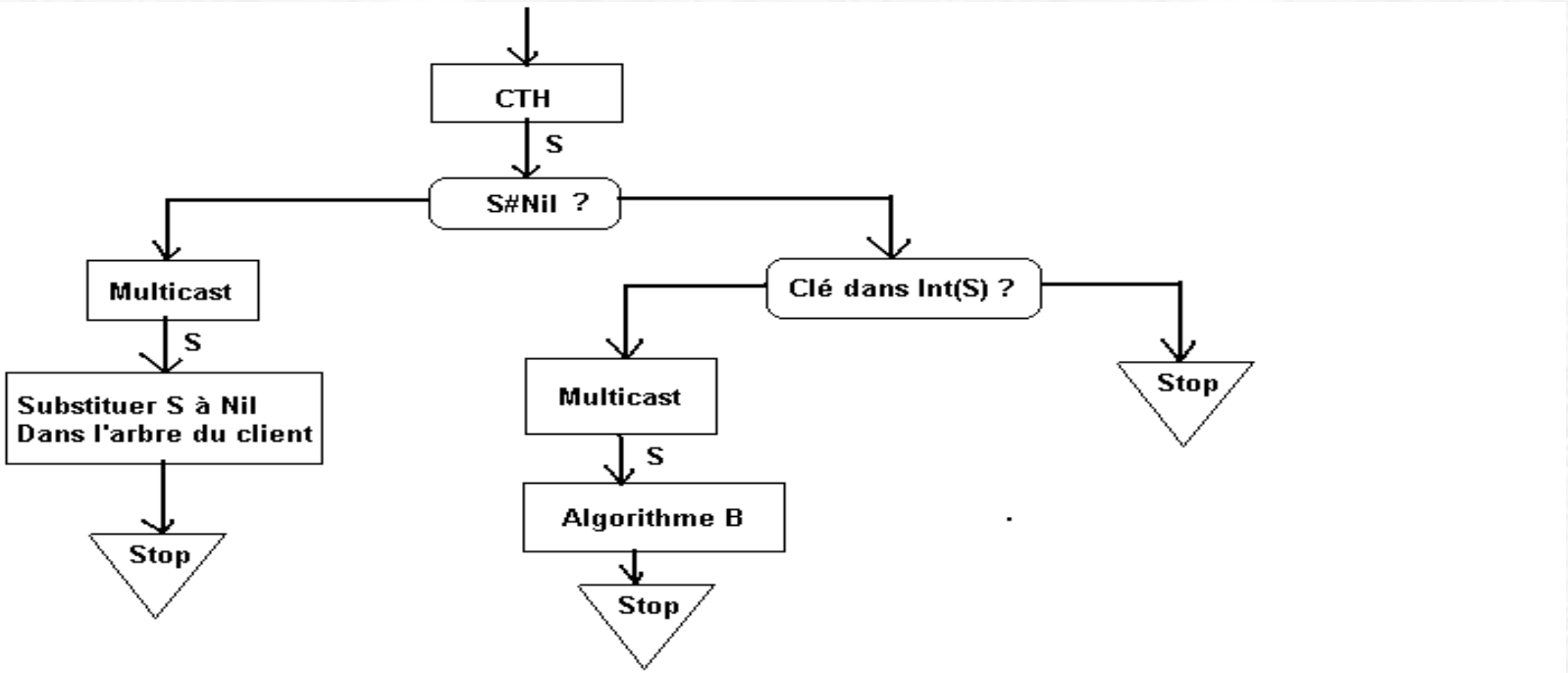
Deterministic Scheme to Distributed Trie Hashing
Variantes

CTH* avec 'Multicast'

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

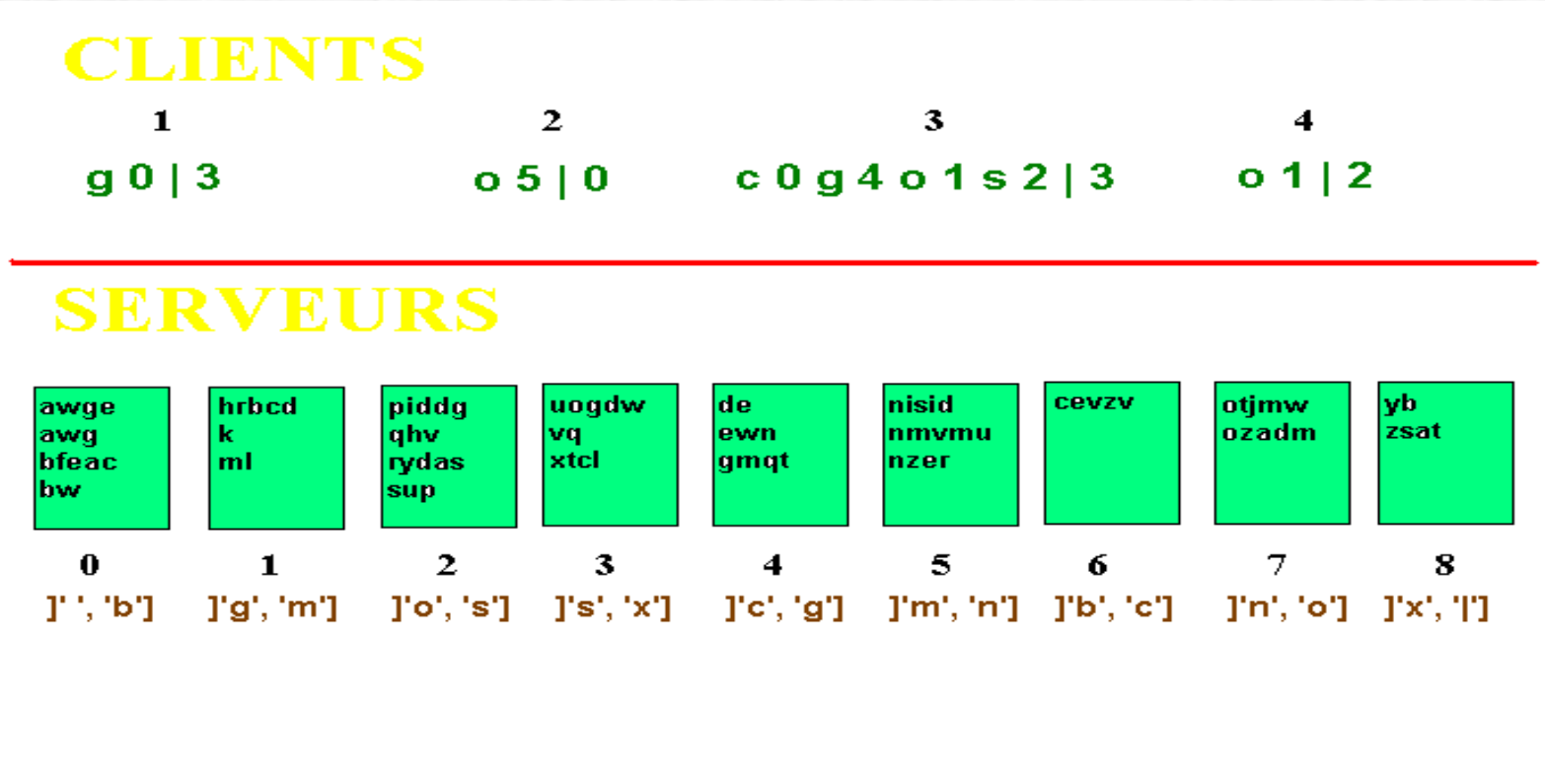
Deterministic Scheme to Distributed Trie Hashing

Variantes



Deterministic Scheme to Distributed Trie Hashing

Variantes



Deterministic Scheme to Distributed Trie Hashing

Conclusion

- Généralisation de TH
- Préservation de l'ordre des articles : facilite les opérations de parcours séquentiel et de requêtes à intervalle.
- Toutes les opérations sans multicast
- Un protocole de communication existe (Plate forme Linux et Windows)
-
- 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*.