HD*: UNE NOUVELLE ET RAPIDE STRUCTURE DE DONNEES DISTRIBUEE ET SCALABLE ORDONNEES

M. ARIDJ [1,2] & D.E ZEGOUR[2]

1 : Université hassiba-benbouali chlef
2 : Institut National d'Informatique Alger

Résumé

Proposées en 1993 par Litwin les Structures de Données Distribuées et Scalables SDDS sont devenus une configuration de base pour la gestion des données sur Multi ordinateur. Caractérisées par leurs transparence des traitements et indépendances des sites, les SDDS ont ouvrés de novelles perspectives de recherche pour l'organisation des données dans des environnements distribués.

Dans ce papier on propose une adaptation du hachage digital qu'est l'une des plus performante méthode d'accès aux environnements distribués toute en respectant les propriétés des SDDS.

Mots clés :hachage digital, hachage distribué, SDDS, multi ordinateur, système distribué.

1. Introduction

Un multi ordinateur consiste en un ensemble de stations de travail et de PCs reliés par un réseau à haut débit. Cette configuration offre des capacités cumulées de stockage (disque et mémoire) et de calcul inégalé par les systèmes traditionnels, offrant ainsi des nouvelles perspectives aux applications hautes performances [11], [13]. Afin de permettre l'exportation de ces performances des nouvelles structures de données sont nécessaires. Les Structures de Données Distribuées et Scalables (SDDS) [9] on été proposées en vue de répondre à cet objectifs. Les données d'une SDDS résident dans des sites serveurs et elles sont accédées à partir des sites clients. Cette nouvelle structure supporte le traitement parallèle et assure des temps d'accès aux données beaucoup plus courts que ceux aux fichiers stockés sur des disques.

Un fichier SDDS n'a pas de répertoire central d'accès contenant la définition de la structure du fichier. Il peut être étendu d'un seul site serveur de la SDDS à n'importe quel nombre de tels serveurs. L'extension se fait par des éclatements de sites que les insertions font déborder.

Chaque client a sa propre image de la structure du fichier. Les mises à jour de la structure du fichier ne sont pas envoyées aux clients d'une manière synchrone. Un client peut alors faire une erreur d'adressage par suite d'une image incorrecte.

Chaque serveur vérifie l'adresse de la requête reçue. Elle est acheminée vers un autre serveur si une erreur est détectée. Le serveur adéquat envoie alors un message correctif au client ayant fait l'erreur d'adressage, ce message est appelé : *Message d'Ajustement de l'Image* (IAM : *Image Adjustment Message).* L'IAM permet au client d'ajuster son image de manière à ne pas refaire la même erreur. Cette image n'est pas néanmoins nécessairement globalement exacte.

Plusieurs SDDSs ont été proposées. Historiquement, la première famille des SDDS est basée sur le hachage linaire : DDH [4], LH* [9]. Elle a donné lieu à de nombreuses variantes, notamment à haute-disponibilité [2], [5] [10] ,[12],[14] ,[16], …..

Le hachage digital [7] (Trie Hashing TH) est l'une des méthodes les plus efficaces pour l'accès aux fichiers ordonnés et dynamiques. La fonction de hachage est représenté par un arbre binaire appelé arbre digital (trie)ou arbre de litwin., qui s'étend et se contracte par les insertion et les suppression ce qui peut engendrer des nœuds Nils (nœud ne pointant aucune case).
Cette technique à fait l'objet de plusieurs travaux notamment dans [8], [15],[1]….

Dans cet article, nous présentons une distribution de la méthode de hachage digital dans un environnement multiordinateur selon les propriétés des SDDS. Cette nouvelle structure est baptisé HD*(hachage digital distribué). L'idée de base est de distribuer les articles est l'arbre digital sur les différents nœuds du multiodinateur. La distribution de hachage digital de base à donner naissance à une SDDS indéterministe (utilisation des multicast) due ou nœuds Nils) pour cela une version sans nœud nils du hachage digital est proposé.

La section 2 rappelle brièvement le principe du hachage digital. La section 3 présente le nouveau schéma du hachage digital sans les nœuds Nils. Le principe et les algorithmes du hachage digital distribué (HD*) sont développés dans la section 4. La section 5 est consacrée aux mesures de performances de la méthode HD*, par la suit, une étude comparative entre la méthode proposée et les autre méthodes concurrentes est présenté. Enfin nous concluons cet article à la section 6.

2. Hachage digital (HD)

Le hachage digital HD, (trie haching) est l'une des plus performantes méthode d'accès pour les fichiers dynamique et ordonnes. Un fichier HD est un ensemble d'articles, identifié chacun d'une manière unique par une clé primaire. Une clé est une séquence de digits ou caractères d'un alphabet donné. Les articles de fichier sont stockés dans des cases de taille fixe ou variable. Figure (fig.1)

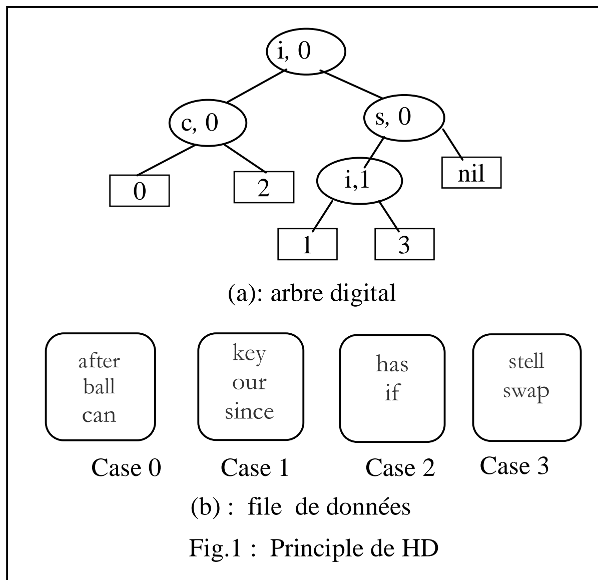


Fig.1 : Principle de HD

L'accès à un article s'effectue par le parcours de l'arbre digital jusqu'à atteindre un nœud externe qui contiendra éventuellement l'adresse de l'article.
L'insertion d'un article peut provoquer une extension de l'arbre et du fichier cependant la suppression provoque des contractions. Les algorithmes détaillés sont présentés dans Lit[81].
Le facteur de chargement est de l'ordre de 70 % pour des insertions aléatoires et de 60% à 70% pour les insertions ascendantes. Un seul accès est nécessaire à la récupération d'un article. [8].

3. Hachage Digital Sans nœuds Nils (HD_{sn})

3.1 Rappel

Rappelant que dans l'arbre digital les nœuds Nils apparaissent lorsque plus d'un digit est nécessaire pour la séquence de division. Dans la méthode de Litwin le problèmes a été résolus comme suit (Fig.2):
Soit la séquence de division suivante : $C=d_0d_1d_2\ldots\ldots d_n$ nécessaire pour l'éclatement de la case m en m'. L'arbre résultant aura la forme suivante

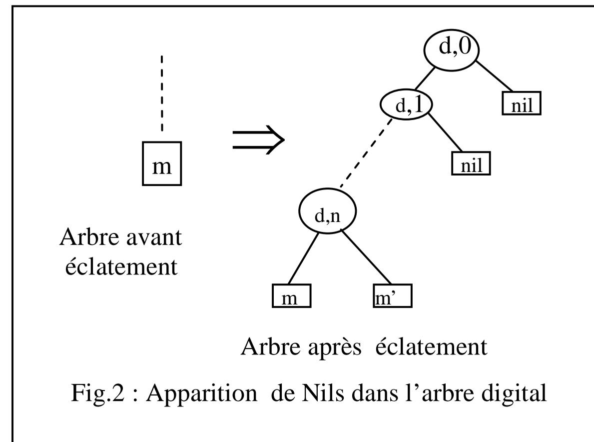


Fig.2 : Apparition de Nils dans l'arbre digital

3.2 Proposition :

Nous proposant un nouveau schéma du hachage digital que nous baptisant HDsn où les nœuds Nils seront replacés par l'adresse de la nouvelle case allouée (m') l'arbre devient alors comme suit (Fig.3)

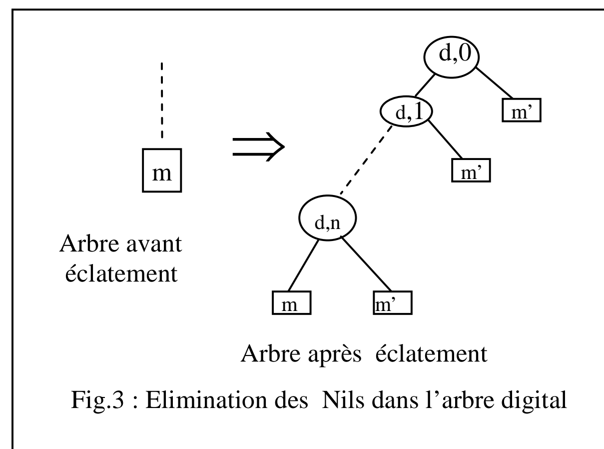
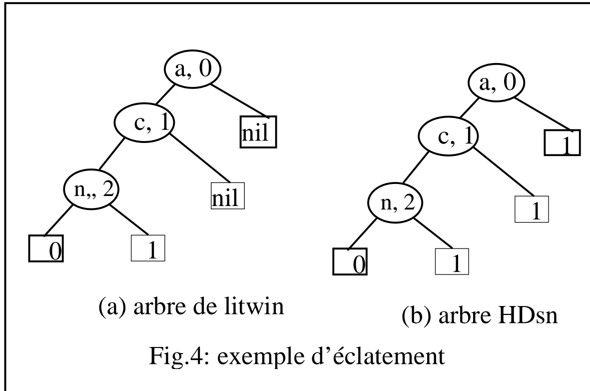


Fig.3 : Elimination des Nils dans l'arbre digital

3.3 Exemple :

Soit à insérer la suite des clés suivantes :
"abmf", "abnm", "acnm", "aczm", "aczh"
L'insertion des clés **"abmf", "abnm", "acnm", "aczm"** se fait dans la case 0 sans problème, l'insertion de la clé **"aczh"** provoque une collision;
Soit la séquence des clés suivante :
"abmf", "abnm", "acnm", "aczm", "aczh"
donc : c'=**"acnm"**, c"=**"aczh"**, alors la plus petite séquence de digits qui permet de différencier entre c' & c" est : acn
L'arbre digital devient (Fig.4) :



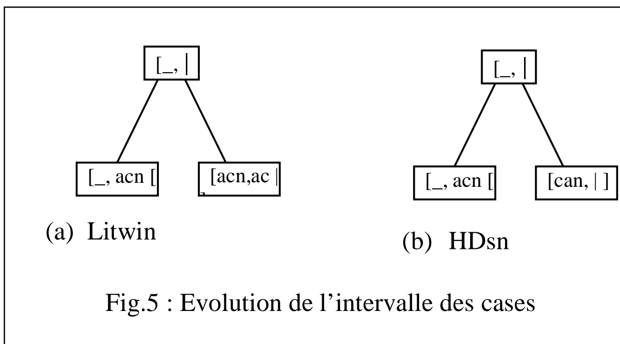
(a) arbre de litwin (b) arbre HDsn

Fig.4: exemple d'éclatement

Commentaire :

Notre proposition est motivée par l'analyse de la portée (l'intervalle) des cases. En effet si on observe sur l'exemple présidant l'évolution des intervalles des cases on constate que initialement toutes les clés seront insérées dans la case 0 : c a d que l'intervalle de la case est [_, |] (_le plus petit digit et | le plus grand digit) après l'éclatement proposée on obtient la configuration suivante (Fig 5.b) cependant l'application de l'algorithme de litwin est schématisé dans la figure 5.a

La séquence de division est osc :



(a) Litwin (b) HDsn

Fig.5 : Evolution de l'intervalle des cases

3.4 Algorithmes du HDsn:

Les algorithmes de recherche et d'insertion restent similaires à ceux proposés dans [7], cependant l'algorithme d'éclatement devient comme suit :

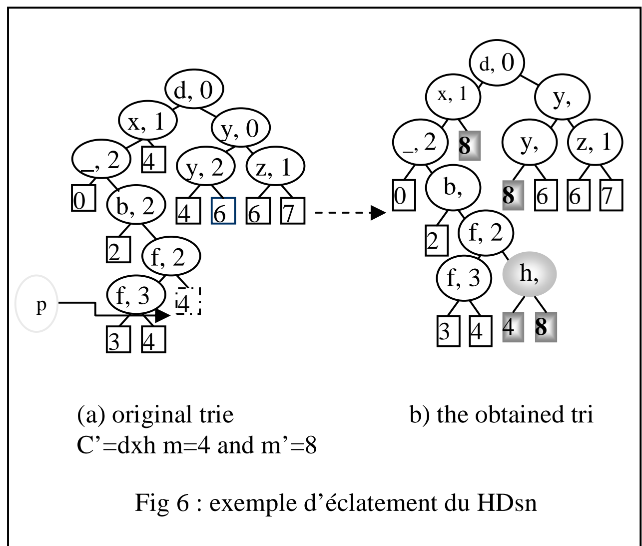
Soit :

m la case en collision, et m' la nouvelle case allouée.

Suivant(p) : une fonction qui donne la feuille qui suit le nœud p dans l'arbre.

1. construire la séquence de (b+1) clés composée des clés de la case m et la nouvelle clé à insérer.
2. calculer la séquence de division : seqdiv=$c_0c_1c_2$…..c_k : la plus petite séquence de digits permettant de distinguer la clé de milieu et la clé maximale de la séquence construite en 1
3. retrouver le nombre i de digits déjà existant dans l'arbre digital. Soit p le dernier nœud visité.
4. générer (k-i-1) avec les digits c_{i+1},c_{i+2} …..c_k et m' comme fils droit.
5. mettre le fils gauche de nœud correspondant au digit c_k à m
6. Tanque suivant(p)=m : replacer m par m'.
7. diviser la suite des (b+1) clés sur m et m'

Exemple :



(a) original trie b) the obtained tri

C'=dxh m=4 and m'=8

Fig 6 : exemple d'éclatement du HDsn

4. Le hachage digital distribué (HD*)

4.1 Concepts de base

Le schéma proposé (HD*) est basé sur l'architecture client/serveur .

Chaque serveur contient :

- La case contenant les enregistrements des fichiers (les clés correspondantes),
- Un arbre digital partiel représentant l'évolution de cette case (la suite des éclatements), cet arbre est créé (initialement avec la séquence |**I** où I est le numéro de ce serveur) et étendu à chaque éclatement de la case de ce serveur.
- Un intervalle **[Min, Max]** qui inclut toutes les clés possibles de la case, cet intervalle est nécessaire pour la recherche (l'appartenance d'une clé à la case est testée par intervalle avant le parcourt de l'arbre).

Chaque client possède un arbre qui représente son image du fichier (l'index) qui peut être incorrect.
Chaque client peut entrer dans le système avec un arbre vide. Cet arbre est mis à jour durant la phase d'adressage.

Initialement le système ne contient que le serveur **0** dont la case est vide,son l'intervalle est **[Small, Large]** où **Small** et **Large** sont respectivement la plus petite et la plus grande clé et enfin l'arbre vide (**0**).
Le fichier évolue par les éclatements dus aux collisions, à chaque collision il y a distribution des clés (du serveur éclaté) sur un autre serveur alloué. Le nombre de serveurs est théoriquement infini et chaque serveur peut être déterminé d'une manière statique ou dynamique (méthode d'allocation).
Lorsqu'une erreur d'adressage se produise, une partie de l'arbre du serveur est transférée à l'arbre du client.
Un exemple d'un fichier TH* est présenté dans la figure Fig 14.

4.3 Algorithmes du TH*

4.2.1 algorithme d'adressage : transformation clé → adresse

Pour réaliser une opération sur un fichier TH*, le client calcule l'adresse du serveur approprié en utilisant son arbre digital qui représente l'image qu'il possède sur le fichier TH*, et lorsque le serveur reçoit une requête de la part d'un client, il vérifie si cette dernière appartient à son domaine(intervalle).Si c'est le cas, il traite la requête et renvoie la réponse au client, sinon ce serveur est considéré comme un mauvais serveur, il doit construire un IAM (Image Ajustement Message) et l'envoyer vers le client pour qu'il mis a jour son arbre. L'organigramme de la figure 7 résume cette opération.

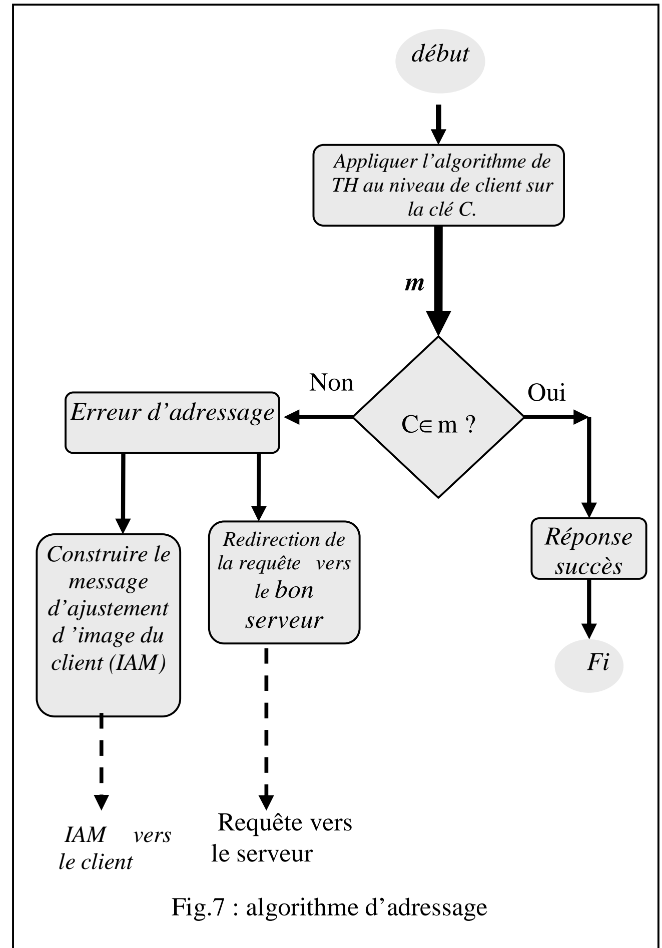


Fig.7 : algorithme d'adressage

4.2.2 Algorithme d'ajustement d'adresse :

L'erreur d'adressage est due au faite qu'un éclatement au moins est survenu au niveau de serveur qui a entraîné la modification de son intervalle, le client n'étant pas informé lors de chaque éclatement (principe des SDDSs), son arbre attribue toujours l'ancienne intervalle au serveur en question. D'où la nécessité d'un l'algorithme d'ajustement.
L'algorithme d'ajustement de l'image s'exécute à deux niveaux : niveau serveur et niveau client.

a) Au niveau du serveur :

Au niveau de l'algorithme de serveur, on procède dans un premier temps à l'identification de sa clé maximale qui n'est autre que les premiers digits de son arbre, ceci permet d'éliminer le préfixe commun avec la clé maximale calculé par le client, car les digits de ce préfixe, au nombre de N, sont déjà dans l'arbre de clients. L'algorithme est formulé comme suit :

Algorithme client :

Début

- Soit **M** : le mauvais serveur donné par l'algorithme d'adressage ;
- Soit **P** le pointeur du nœud de **M** ;
- Recevoir le sous arbre (Sq) envoyé dans IAM par le serveur.
- determiner M’ le dernier serveur dans Sq
- Remplacer le nœud externe M par le (sous) arbre (Sq) du serveur M;
- Tanque suivant(p)=M : replacer M par M’.

Fin.

b) Au niveau du client

L'exécution de l'ajustement par le client consiste à mettre à jour son arbre à travers les informations contenues dans **IAM** envoyé par le serveur. Pour ce faire il exécute l’algorithme suivant :

Algorithme serveur :

Debut

1. Appliquer l’algorithme d’adressage :
cm $_{\text{serveur}}$ la clé maximale de m selon son arbre
.

2. Déterminer le préfixe commun: partie commune entre le client & le serveur **M**;
p← cm $_{\text{serveur}}$ **– cm** $_{\text{client}}$;
Si p == cm $_{\text{serveur}}$ **alors**
Tout l'arbre du serveur sera transféré vers le client ;
Sinon
Déterminer dans l'arbre du serveur, le sous arbre (Sq) non commun qui commence par le premier digit de la séquence **P**.
Fin si;

fin

Exemple :

Les figures Fig 8 et Fig 9 donnent deux exemples d’illustration du mécanisme d’ajustement de l’image du client.

Dans la figure Fig 8 le serveur doit transmettre tout son arbre au client car il existe pas de préfix commun, cependant dans la figure Fig 9 seulement une partie de son arbre est transféré ver le client.

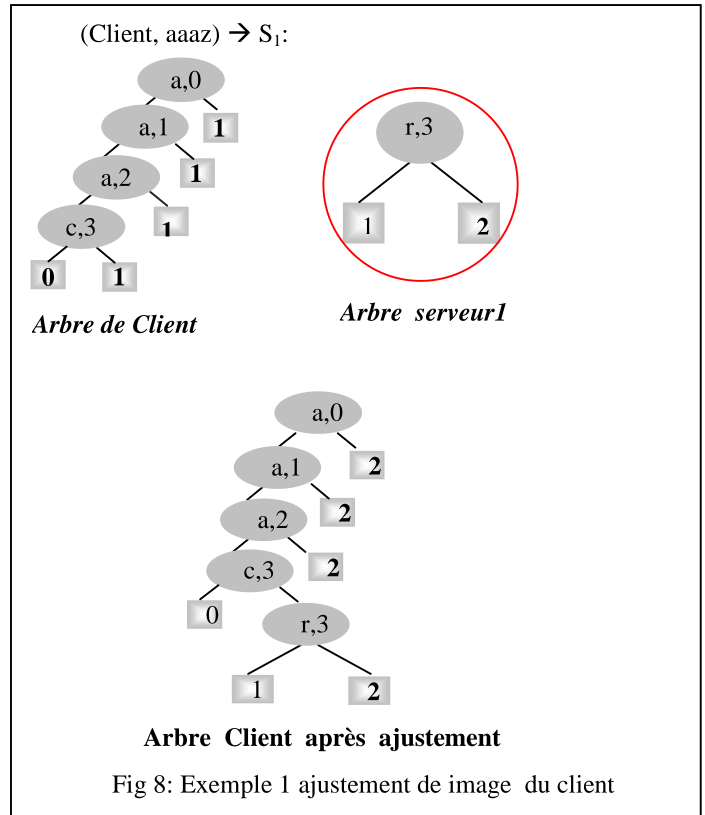


Fig 8: Exemple 1 ajustement de image du client

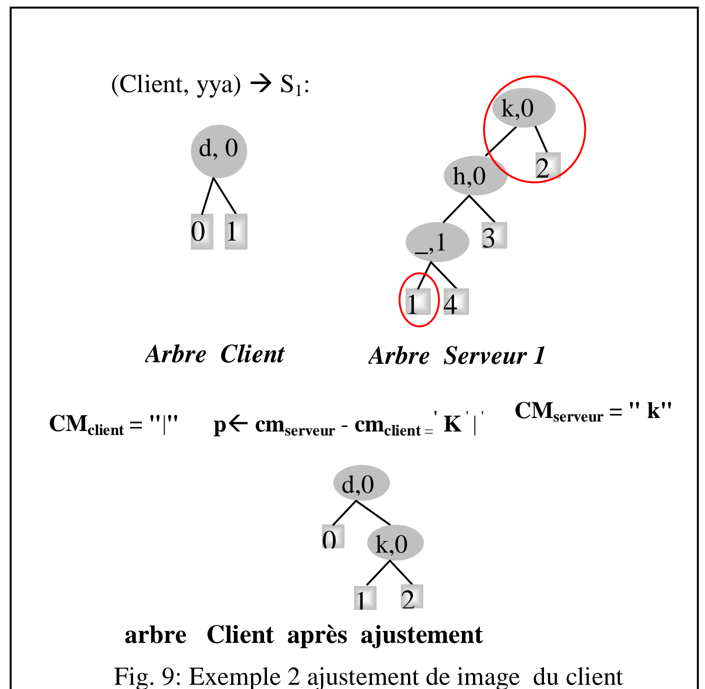


Fig. 9: Exemple 2 ajustement de image du client

4.2.3 L'algorithme d'insertion

L'insertion nécessite en premier temps une recherche sur la clé en appliquant l'algorithme de recherche. Si la clé n'existe pas dans le fichier, elle sera insérée dans le serveur approprié, si ce dernier est **saturé**, on applique l'algorithme d'**éclatement**. Le déroulement général d'une insertion dans une SDDS TH* multi client est donné par l'organigramme de la figure suivante (Fig10) :

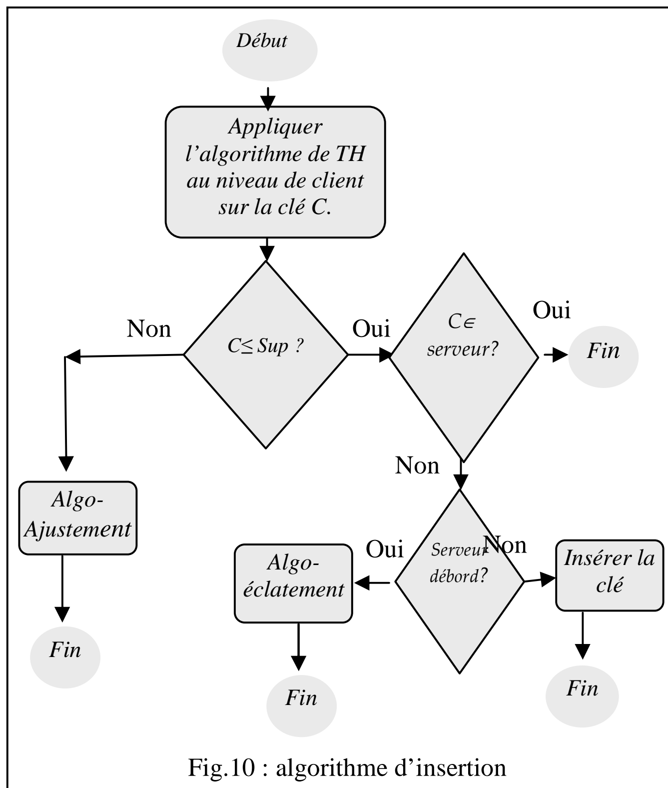


Fig.10 : algorithme d'insertion

4.2.4 L'algorithme d'éclatement d'une case

L'algorithme d'éclatement est appelé par le serveur lorsque il y'a un cas de débordement provoqué par l'insertion. Le traitement d'éclatement est réalisé par l'algorithme suivant :

Début

1. On considère la séquence des **b+1** clés de **m**, appliquer l'algorithme d'éclatement de THsw pour modifier l'arbre de **m.** et partager les clés entre **m** et **m'**;
2. Calculer les Clés maximales CM et CM' de **m** respectivement **m'** tel que : CM =**seq+'|'** et CM' est l'ancien Clé maximale de **m**, tel que **seq la séquence de division**
3. Créer au niveau du serveur **m'**, l'arbre digital vide, intervalle =
4. redistribuer les clés entre m et m'

Fin

4.2.5 . L'algorithme de requête par intervalle

Il arrive parfois qu'on veut traité non seulement une clé mais un ensemble de clés à la fois, cet ensemble est représenté sous forme d'un intervalle désigné par une borne inférieure et une borne supérieure[clé min, clé max]. Le faite que le TH* génère et manipule des fichiers mono-clés et ordonnés (l'arbre digital garde l'ordre des clés), une opération de recherche ou suppression par intervalle sera facile à traiter

Pour simplifier l'algorithme, on utilise les définitions suivantes :

M : numéro de serveur ;

Arrêter : booléen indique la fin de la requête (la borne sup de l'intervalle est atteinte).

Calcul_Fils (m) : retourne le numéro de prochain serveur qui suit le serveur m ;

a) Algorithme du client

Début

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

2- Si (clé_max ≤ CM) Alors Appliquer au niveau de serveur m l'algorithme de requête par intervalle [clé_min, clé_max] ;

Sinon Appliquer au niveau de serveur m l'algorithme de la requête par intervalle [clé_min, CM] ;

Fin Sinon ;

A la réception des réponses serveur on a trois cas :

a. ajustement : dans ce cas, on doit ajuster l'arbre du client et redimensionner la requête à nouveau.

b. arrêter==true : dans ce cas, on arrive au dernier serveur contenant certainement la clé_max de l'intervalle, on doit arrêter la requête pour envoyer la réponse vers l'application.

c. arrêter==false : dans ce cas, le client doit calculer l'adresse de prochain serveur par la fonction Calcul_Fils

3-Aller à l'étape 1 pour envoyer la requête avec le nouvel intervalle *] CM, clé_max].*

fin

b) Algorithme de serveur

Début
Si (clé_min appartient au serveur m) Alors
Si (clé_max ≤ CM) Alors
Retourner les clés de serveur m de l'intervalle [clé_min, clé_max]

Arrête<--true ;
Sinon
Retourner les clés de serveur m de l'intervalle [clé_min, CM]
Arrête<--false ;
Fin Si ;
Sinon
Envoyer un IAM (Message d'Ajustement d'image) au client ;
Fin Si ;

Fin

Commentaires

Au niveau du client, on cherche d'abord le serveur qui doit avoir la clé clé_min, on partage alors séquentiellement l'intervalle [clé_min, clé_max] à un ou plusieurs sous intervalles qui forment une partition de ce dernier, on envoie une requête à chaque serveur associé à un intervalle selon l'arbre de client.
Au niveau de serveur, on procède à la vérification que la borne inférieure de l'intervalle de la requête reçue appartient bien à son domaine (clé_min <= CM de serveur). Si ce n'est pas le cas, un IAM *(Message d'Ajustement d'image)* st envoyé au client et la requête est redirigée vers un autre serveur (qui sera certainement son fils). Sinon, si la borne supérieure est dans le domaine de serveur, on traite la requête (retourner les clés s'il s'agit d'une recherche et les supprimer lorsque la requête est une suppression), et la requête est terminée parce qu'on a arrivé à la borne supérieure de notre requête.

5. Simulation et test de performances

5.1 Environnement de simulation
Nous avons implémenté le schéma proposé en C sous LINUX mandrak 8.0 sur un multiordinateur de 4 postes (machines Pentium III, 128 Mo de RAM et 994 Mhz). Chaque machine peut contenir plusieurs serveurs et plusieurs clients (notion de machine virtuel).
Chaque serveur est composé une case de taille fixe, un arbre digital et son intervalle de clé. Chaque client possède un arbre digital qui représente son image du fichier TH*. Chaque client lance ses requêtes indépendamment des autres clients et à chaque requête des performateurs sont mesurées.

5.2 Quelques résultats
a) facteur de chargement
Nous avons fixé la taille de la case des serveurs à 1000 clés est nous avons lancés l'insertion de 100000 clés. Les variations du facteur de chargement et resumé dans le graphe de la figure fig.11 :

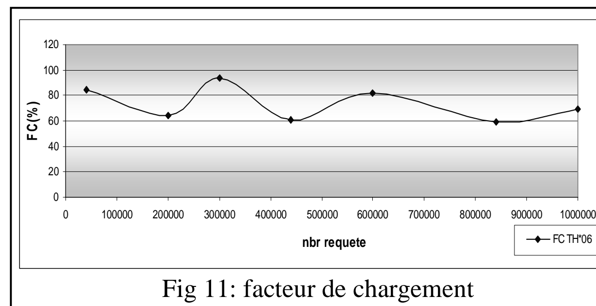


Fig 11: facteur de chargement

Commentaire
En observant la courbe de la figure Fig11 on constate que la SDDS TH* a une bonne occupation de l'espace de stockage puisque le facteur de chargement est dans l'intervalle 65 % à 93 %.
b) variation du nombre d'éclatement des serveurs
Sous les mêmes hypothèses, nous avons observés les variations des éclatements des serveurs. La figure Fig.12 montre que les éclatements augmentent linéairement en fonction du nombre d'insertion :

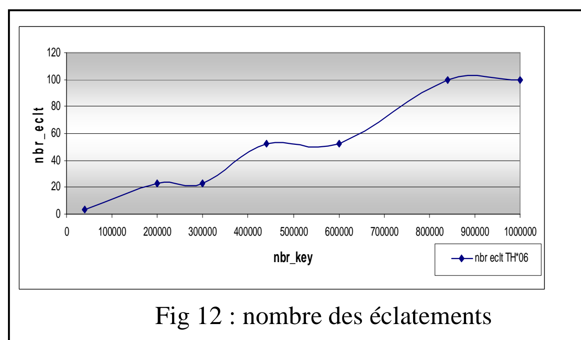


Fig 12 : nombre des éclatements

c) Temps moyen d'une recherche et d'insertion
La courbe le la figure Fig13 présente le temps moyenne d'une opération d'insertion et de recherche. Les deux courbes sont pratiquement linaire ceci implique que nous avons un temps moyen qui n'évolue avec le nombre de clés ce qui rend la méthode très scalable

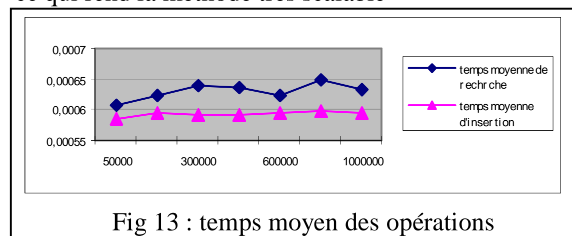


Fig 13 : temps moyen des opérations

5.3 Comparaison avec d'autre méthode

Le tableau suivant résume l'étude comparative de la méthode présentée (TH*) et les autre méthode concourantes.

	Index de client	Index de serveur	Multi cast	Converg- -nce de l'image	ord onn é	FC
RP*s	Pas d'index	(A, C)	Oui	[M/0.7*m]	Oui	≈70%
RP*c	(A, C)	Pas d'index	Oui	M-1	Oui	≈70%
RP*n	Pas d'index	Pas d'index	Oui	Pas d'image	Oui	≈70%
LH*	n,i	n,i	Oui	log M	Non	65-95%
CTH*	Arbre digital compact	Arbre digital compact	Non	Log M	Oui	65-90%
TH*	Arbre digital	Arbre digital	Non	Log M	Oui	65-90%

6. Conclusion

Ces dernières années ont vu des avancées considérables de l'informatique qui ont favorisé l'émergence de nouvelles architectures distribuées. Il s'agit essentiellement de stations de travail et de PCs, provenant du marché grand public et reliés par des réseaux à haut débit. Ces configurations offrent des capacités cumulées de stockage et de puissance de calcul quasi-illimitées et largement sous-utilisées aujourd'hui.

Cependant, les applications actuelles ne tirent pas profit au maximum de ces avancées. Pour pallier à ce manque, Les Structures de Données Scalables et Distribuées (SDDS) ont été introduites.
Les SDDS sont une étape dans l'implémentation, à base du modèle client/serveur, des systèmes de gestion de fichier et de bases de données distribuées et scalables

Dans cet article, nous avons présentés dans un premier temps une version du hachage digital sans les nœuds Nil, en suite nous l'avons adaptés pour les environnements distribués selon le model des structure de données distribuées et sclable . Cette extension a donnée naissance à une nouvelle SDDS que nous baptisant TH*.

L'étude des performances de la SDDS TH * a montrée que la méthode gade un bon facteur de chargement il est entre 65 % à 90 %, de plus le fichier SDDS TH* est ordonné ce qui rond les accès par intervalles très rapide et pour les accès aléatoire les temps de réponse sont proche des méthode concurrentes (LH* et RP*).

7. Réferances

[1] M.aridj ,D.E Zegour , **A new multi-attributes acces method for voluminoues files ,** SPECTS 2005 Summer Simulation Multiconference July 24–28, 2005 • Philadelphia,Pennsylvania, USA.

[2] M.Aridj « **LH*TH: New fast Scalable Distributed Data Structures SDDS** » NET Technologies 2005 International Conference May 29 – June 1, 2006 University of West Bohemia, Plzen, Czech Republic.

[3] Bennour, F., Diène, A. W., Ndiaye, Y. Litwin, W., Scalable and Distributed Linear
Hashing LH*LH under Windows NT. SCI-2000 Orlando, Florida, USA. July 23-26, 2000.

[4] Devine, R. Design and Implementation of DDH: Distributed Dynamic Hashing. Intl. Conf. On Foundations of Data Organizations, FODO-93. Lecture Notes in Comp. Sc., Springer-Verlag (publ.), Oct. 1993.

[5] Karlsson, J. Litwin, W., Risch, T. LH*lh: A Scalable High Performance Data Structure for Switched Multicomputers. Intl. Conf. on Extending Database Technology, EDBT-96, vignon, March 1996.

[6] Litwin, W. Linear Hashing : a new tool for file and tables addressing. Reprinted from VLDB-80 in reading in database 2-nd ed. Morgan Kaufmann Publishers, Inc.,
1994. Stonebraker , M.(Ed.).1999

[7] Litwin,W :'Trie Haching'. Proc.ACM. SIGMOD'81, pp.19-29

[8] : W.Litwin « Trie hashing : Further Properties and Performances » Int.Conf. on Foundation of Data Organization. Kyoto, May 1985, Plenum press.

[9] Neimat, M-A., Schneider, D. LH* : Linear Hashing for Distributed Files. ACM-SIGMOD Intl. Conf. On Management of Data, 1993.

[10] Litwin, W., Neimat, A.M. High Availability LH* Schemes with Mirroring, Intl. Conf on Cooperating systems, Brussels, IEEE Press 1996

[11] Tanenbaum, A., S. Distributed Operating Systems. Prentice Hall, 1995, 601.

[12] Tung, S, Zha, H, Kefe, T. Concurrent Scalable Distributed Data Structures , Proceedings of the ISCA International Conference on Parallel and Distributed Computing Systems , pp. 131-136, Dijon, France, September,1996. Edited by K. Yetongnon and S. Harini

[13] Ullman, J. New Frontiers in Database System Research. Future Tendencies in Computer Science,Control, and Applied Mathematics. Lecture Notes in Computer Science 653, Springer-Verlag, 1994. A. Bensoussan, J. P. Verjus, ed. 87- 101.

[14] Vingralek, R., Breitbart, Y., Weikum, G. Distributed File Organization with SalableCost/Performance. ACM-SIGMOD Int. Conf. On Management of Data, 1994.

[15] D.E.Zegour,W,Litwin,G.Levy Multilevel trie hashing int. conf on VLDB venise Italy 1987.

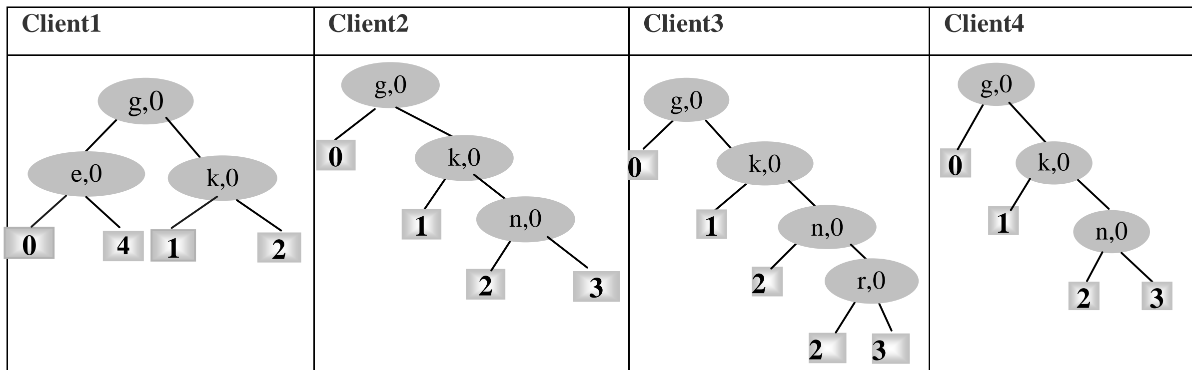
[16] D.E.Zegour Scalable compact trie haching , Elsevier information and software technology 46 P923-935 April 2004.

Exemple de TH *

Pour illustrer la méthode on se propose un schéma avec 4 clients et plusieurs serveurs. Chaque serveur est composé d' une case de capacité b=4, un intervalle de valeurs et un arbre digital.
La liste des insertions est représentée par un ensemble de couple numéro client et la clé a inséré :
(1 js), (1 hw), (3, c), (2, gwmr), (3, g), (2, km), (4, zur), (1,ewg), (3, lewhv), (2, nrq), (3, mf), (4, pem), (4, rl), (2, bqyg), (3, v), (1, j), (2, qcm), (4, czxav), (2, lhgd), (3, z), (1, lrz), (3, kiyfg), (4, pbtpr), (3, hpqtp), (4, h).

L'état final du fichier (arbre, case) est présenté dans la figure suivante :

Structure de l'arbre au niveau des clients



Structure de fichier (arbre & case) au niveau des serveurs:

	Serveur 0	Serveur 1	Serveur 2	Serveur 3
arbre	g,0 e,0 1 0 4 Sup ='e'	k,0 j,0 2 h,0 7 1 8 Sup ='h'	n,0 l,0 3 3 6 Sup ='l'	r,0 3 5 Sup ='r'
Case-	bgvg c cav cwg	h hpgtp hw	Lewhr lhgd lrz	pem rl qcm pbtpr

Fig.14 : exemple de distribution d'un fichier TH*