HAL
archives-ouvertes.fr

Systeme de gestion de fichiers base sur le hachage sequentiel binaire

J. Kouacou Gnrangbe

▶ To cite this version:

J. Kouacou Gnrangbe. Systeme de gestion de fichiers base sur le hachage sequentiel binaire. RR-1043, INRIA. 1989. <inria-00075515>

HAL Id: inria-00075515
https://hal.inria.fr/inria-00075515

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.(1)39 63 55 11

Rapports de Recherche

N° 1043

Programme 4

SYSTEME DE GESTION DE FICHIERS BASE SUR LE HACHAGE SEQUENTIEL BINAIRE

Jean KOUACOU GNRANGBE

Mai 1989

RR-1043

SYSTEME DE GESTION DE FICHIERS BASE SUR LE HACHAGE SEQUENTIEL BINAIRE.

Par JEAN KOUACOU GNRANGBE
Professeur d'Informatique
INSET - COTE D'IVOIRE

A FILE HANDLING SYSTEM BASED ON SEQUENTIAL BINARY HASHING

ABSTRACT

This paper describes a new method of file management system (F.M.S.) based on Sequential Binary Trie Hashing, we proposed in 1986 and which improves Trie Hashing performances (Litw81,..84).
Improvements concerned on one side with memory size necessary to the hashing function and on the either side with access performances for large files.

The paper describe the data structures, the function and the user guide of the F.M.S.

RESUME:

Ce pepier est une description de système de gestion des fichiers (S.G.F) basé sur le HACHAGE SEQUENTIEL BINAIRE. Le hachage séquentiel binaire est une nouvelle méthode de gestion de fichiers que nous avons proposée en 1986 et qui améliore les performances du hachage digital (Litw81,..84). Les améliorations apportées résident d'une part dans la taille mémoire nécessaire à la fonction de hachage, et d'autre part dans les performances d'accès pour les grands fichiers.

Le papier décrit la structure des données, les fonctions et le mode d'utilisation du S.G.F.

I STRUCTURE DES DONNEES

1) *Principes*

P1) L'alphabet utilisé dans le hachage séquentiel binaire est constitué par la configuration binaire des caractères de la clé. Cela signifie qu'une clé est une chaine de bits, numérotés de 0,1,2,.....,n.
Donc dans la méthode, la fonction d'accès transforme la clé en chaine de bits.

P2) La fonction du hachage génère un arbre digital binaire en mémoire centrale. Cela signifie que les noeuds de l'arbre sont des bits (0s et 1s).
Nous avons choisi de représenter l'arbre digital en mémoire centrale par deux tables de longueur dynamique:

- l'une notée **B**, représente la structure de l'arbre. **B** est donc une **table de bits**.
- l'autre notée **T**, contient les feuilles de l'arbre. **T** est appelée **table des pointeurs**.

B et T sont dans le parcours postfixé (ou préordre) de l'arbre.

p3) Pour le calcul d'adresse, la fonction utilise une clé c' (chaine de bits) qui est formée par les 5 derniers bits de la représentation binaire de chaque caractère de la clé. Ceci permet de minimiser le nombre de noeuds "nil" qui occupent inutilement de la place en mémoire centrale.

2) Constitution de l'arbre digital

L'arbre digital s'étend par les éclatements des cases (pages) qui débordent. Chaque éclatement partage les clés en deux parties: une partie reste dans la case courante et l'autre partie est transférée dans une nouvelle case allouée à la fin du fichier.
A chaque éclatement, on insère au moins deux bits dans B et un élément dans T.

2.1) Description de la constitution d'un fichier

2.1.1) Eclatement

Initialement la table B est vide (elle ne contient aucun élément),

- on alloue la première case du fichier; c'est la case d'adresse 0.
- on affecte le premier élément à T; c'est l'élément qui contient la première adresse, c'est à dire 0.

Pour les B premières insertions, il n'y a pas de calcul d'adresse; tous les articles sont insérés à l'adresse 0. L'insertion du $(b-1)^{ème}$ article provoque une collision.

2.1.1.1 Résolution de la collision

- on alloue une nouvelle case à la fin du fichier. Il en sera ainsi pour chaque éclatement. Pour ce premier éclatement, c'est la case d'adresse 1.

- on insère un nouvel élément dans T. Il en sera ainsi pour chaque éclatement. Pour ce premier éclatement, c'est l'élément qui contient l'adresse 1.

- on examine le ***bit suivant*** de chaque clé, y compris celle qui déborde. Il en sera ainsi pour chaque éclatement. Pour ce premier éclatement, le bit examiné sera le premier bit, c'est à dire le bit d'indice 0.

Si le bit examiné vaut 0, la clé concernée reste dans l'ancienne case (case d'adresse 0). Si le bit examiné vaut 1, la clé concernée est transférée dans la nouvelle case (case d'adresse 1).

- on insère le couple de bits 01 dans la table B.

2.1.1.2) Cas de noeud "nil"

C'est le cas où tous les bits examinés ont la même valeur. Toutes les clés sont de ce fait dirigées vers la même case. La collision n'est donc pas résolue. On dit qu'il y a formation de noeud "nil". Dans ce cas:

- on insère un élément dans T. On met dans cet élément la valeur particulière "nil". Cela signifie qu'il ne pointe pas de case sur le fichier.

- on insère deux bits dans la table des bits B.

Et on réitère l'opération d'éclatement.

2.1.2) Fusion

Au fur et à mesure des suppressions d'articles, les cases du fichier se vident et le taux de remplissage baisse. Pour éviter la détérioration du taux de remplissage, il faut fusionner les cases ayant un nombre d'articles inférieur à un certain seuil.

2.1.2.1 Principes

P4) Nous envisageons la fusion que lorsque la case courante (case où il ya eu la suppression) est à moitié vide.

P5) Nous ne fusionnons que des cases logiquement consécutives. Les cases logiquement consécutives sont des cases pointées par des noeuds externes frères.

P6) Des deux cases qui fusionnent,

- celle qui a la plus petite adresse reçoit les articles.

- et celle qui a la plus grande adresse reçoit les articles de la dernière case du fichier.

La dernière case est restituée au système.

Conséquences: la fusion des cases induit celles des noeuds de l'arbre digital (B et T). Dans le cas où la fusion est possible, la table T rétrécit

d'un élément et la table B de deux bits.

P7) Dans le cas où une case devient vide et qu'elle ne peut pas être fusionnée (principe P5 non satisfait),

- l'élément correspondant dans T est porté "nil"
- les articles de la dernière case du fichier sont transférés dans la case qui est vide.
- la dernière case est restituée au sytème
- le pointeur (l'élément de T) de la dernière case est mis à jour: il porte maintenant l'adrese de la case qui était vide.

3) Requête à intervalle

Algorithme

données:

clesup = clé correspondant à la borne supérieure de la requête.
cleinf = clé correspondant à la borne inférieure de la requête.
adr1 = numéro dans T de la borne inférieure de l'intervalle.
adr2 = numéro dans T de la borne supérieure de l'intervalle.
b = capacité d'une case.

variables

i,j = entiers.
case = tampon.
F = fichier.

```
début
    pour i:= adr1 à adr2 faire
    si T[i] <> nil alors
        début
            lire(F,T[i]);
            case:= F,t[i];
            pour j:= 1 à b faire EXAMEN DES CASES;
        fin
fin
```

L'examen des cases dépend de la requête:

- pour l'intervalle fermé, l'examen se fait comme suit

si (case[j]≥cleinf) et (case[j]≤clesup) alors écrire(case[j]);

- pour l'intervalle ouvert, on a:

si (case[j]>cleinf) et (case[j]<clesup) alors écrire(case[j]);

- pour l'intervalle semi ouvert à gauche, on a:

si (case[j]>cleinf) et (case[j]≤clesup) alors écrire(case[j]);

- pour l'intervalle semi ouvert à droite, on a:

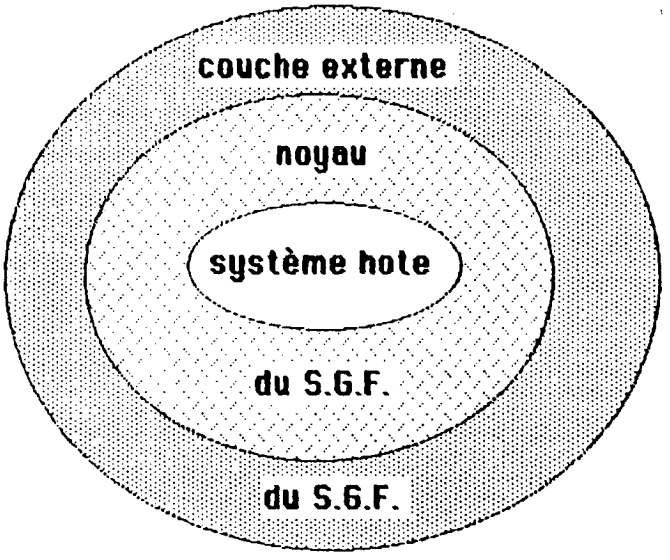
si (case[j]≥cleinf) et (case[j]<clesup) alors écrire(case[j]);

II ARCHITECTURE DU S.G.F

Notre S.G.F. a été construit autour du système UNIX.

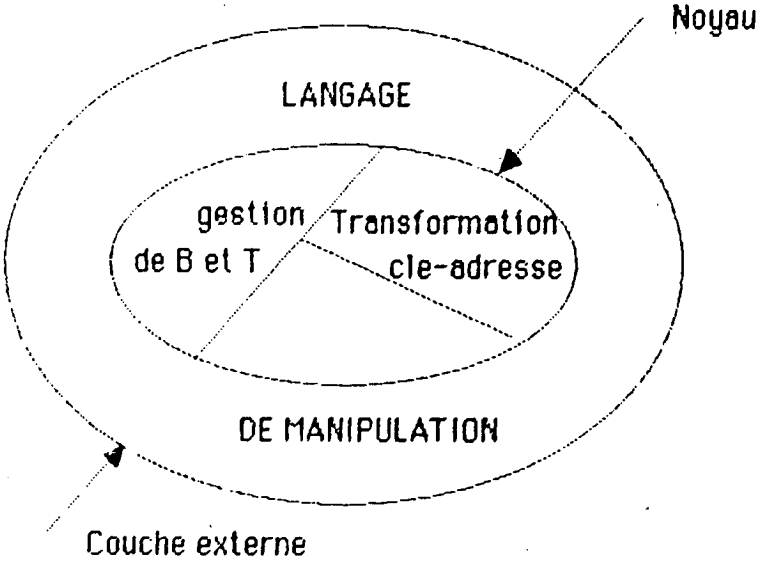
1) Composantes physiques

Il est composé de deux couches: une couche interne qui est le noyau et une couche externe. L'articulation physique de ces couches est la suivante:



structure physique du S.G.F

2) Fonction des couches



La figure ci-dessus montre les fonctions des deux couches de notre système.

3) Fonctionnement

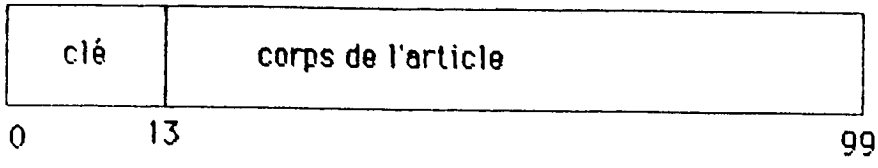
Quand un utilisateur veut créer un fichier de données, le système crée les

deux tables B et T correspondant au fichier de données. La gestion de ces deux tables est totalement transparente par rapport à l'utilisateur.

3.1) Organisation

3.1.1 structure de l'article du fichier

La clé logique est toujours au début de l'article. Sa longueur est de 14 caractères, mais peut atteindre jusqu'à 19 caractères.
La taille d'un article est de 100 caractères, mais peut atteindre jusqu' à 200 caractères.



structure d'un article

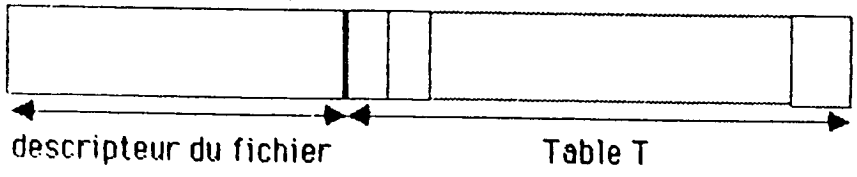
3.1.2 Descripteur du fichier

nom du fichier	capacité d'une case	taille de B(en bits)	taille de T(en éléments)	nombre de cases	nombre d'articles

Les 2 premières composantes sont des paramètres statiques. Les autres paramètres sont dynamiques et varient suivant l'évolution du fichier.

3.1.3 Articulation physique

Dans notre organisation, la table des pointeurs T et le descripteur sont dans la même structure.



Cette structure est donc composée d'une partie fixe qui est le descripteur du fichier et d'une partie dynamique qui est la table T.

III Exploition du S.G.F

1) Primitives d'appels

- CREATEF ---> pour créer un fichier.
- EDITEF ---> pour éditer un fichier.

- PUTREC ---> pour insérer un article.
- VIEWREC ---> pour consulter un article.
- MODIFREC ---> pour modifier un article.
- DELETREC ---> pour supprimer un article.
- REQUETE ---> pour la lecture séquentielle ordonnée.
- OPENF ---> pour ouvrir un fichier.
- CLOSEF ---> pour fermer un fichier.

2) Exploitation en mode interactif

Le mode interactif a été conçu pour des démonstrations. De ce fait, un fichier a été créé. L'ouverture et la fermeture de ce fichier sont faites automatiquement. L'ouverture est faite à l'activation du S.G.F; la fermeture est faite quand l'utilisateur décide de quitter le système.

L'appel du S.G.F se fait en frappant 'hb' (hachage binaire). A l'activation, un menu est proposé. En plus des primitives que nous venons de citer, le mode interactif permet de lister la table des bits et la table des pointeurs.

3) Exploitation en mode batch

Cette version a été écrite sous forme d'un module Pascal. Le programme utilisateur l'appelle en utilisant la clause USE dans l'entête de son programme, comme ceci:

```
program toto(.....); use sgf;
begin

.

end.
```

3.1 Manuel d'utilisation

3.1.1 Primitive opérant sur un fichier

3.1.1.1 Création d'un fichier

Pour créer un fichier, l'utilisateur fournit le chemin d'accès (pathname) et la capacité d'une case, comme ceci:

```
createf(CHEMIN,b);
```

où CHEMIN est le chemin d'accès et b la capacité d'une case.

Les valeurs de b permises actuellement sont telles que 10 b 100. Le nom du fichier ne doit pas dépasser 14 caractères.

3.1.1.2 Ouverture d'un fichier

La primitive openf est une fonction booléenne. La syntaxe est la suivante:

```
if openf(CHEMIN) then ...
```

3.1.1.3 Fermeture d'un fichier

```
closef( CHEMIN);
```

3.1.2 Primitives opérant sur les articles

3.2.2.1 Insertion d'un article

La syntaxe est la suivante:
putrec(CHEMIN,ART);
Où ART est l'article qu'on veut insérer dans le fichier. ART est un string;
La longueur maximum actuelle de ART est de 20 caractères (pour la démo).

3.1.2.2 Consultation d'un article

viewrec(CHEMIN,CLE,REP,ARTI);
Où CLE est la clé de l'article que l'on veut consulter; elle est fournie par l'utilisateur.
REP est une variable booléenne qui vaudra 'true' si l'article est dans le fichier, ARTI contiendra l'article recherché.
Donc REP et ARTI sont les variables qui contiennent le résultats de la requête.
Ex:
viewrec (CHEMIN, CLE , REP, ARTI);
if REP then writeln (ARTI);

3.1.2.3 Suppression d'un article

deletrec (CHEMIN, CLE, REP);
REP est une variable booléenne. Si REP vaut 'true' alors l'article se trouvait dans le fichier et il été supprimé. Sinon, l'article ne s'y trouvait pas.

CONCLUSION

L'implémentation que nous venons de décrire réalise un système de gestion de fichiers inconnu jusqu'alors.

Comparativement aux méthodes existantes, avec les mêmes ressources (place mémoire et un accès disque par recherche), notre méthode permet de gérer un fichier plus grand. Par exemple avec un tampon de 32 K, notre technique permet de gérer un fichier de 14560 cases, soit 1456000 articles (pour b = 100 articles); alors que le hachage digital n'offre que 5400 cases.

Ces performances place notre méthode parmi les meilleures actuelles; elle est notamment plus performante que les arbres_B (ang. B_trees) introduits par BAYER en 1977.

Ce S.G.F est actuellement disponible sur l'ordinateur SM90, système UNIX à L'INRIA (Institut National de la Recherche en Informatique et Automatique Rocquencourt. FRANCE), Bât. 17. Une implémentation sur PC est envisagée à l'INRIA et à L'INSET.

BIBLIOGRAPHIE

BAYER 1972, Organization and maintenance of large orderer indexes. Acta Informatica

KNUTH 1973, the Art of Computer Programming. Addison Wesley.

BAYER 1977, Prefix B_trees. ACM TODS 2,1

LARSON 1978, Dynamic hashing. Bit 18.

COMER 1979, the organization B_tree. ACM Comp. SURV.

LOMET 1979, Multitable search for B_tree files. ACM SIGMOD

FAGIN, NIEVERGELT, PIPPENGER, STRONG 1979, Extendible hashing - a fast access method for dynamic files

DE JONGE, JANENBAUM 1981, A Fast Tree-based Access method for

Dynamic files - University of Amsterdam

LITWIN 1981, Trie Hashing. SIGMOD 81 ACM

SCHOLL 1981, New File Organization Based on Dynamic Hashing. ACM TODS.

ORENSTEIN 1983, A Dynamic Hash File for Random And Sequential

Accessing. VLD 13

FLAJOLET 1983, On the Performance Evaluation of Extendible Hashing and the searching.

LITWIN 1984, Data Access Methods and Structures to Enhance Performance.

LITWIN 1985, Trie hashing: Further properties and performance. Int. Conf. on Fondation of Data organization Kyoto.

DATE 1986, An Introduction to relational Database Systems - 4-th es.

Addison Wesley

KOUAcou 1986, Le hashage Séquentiel Binaire - Thèse de 3° cycle - Univ Paris Dauphine.

Etude des Performances du Hashage Séquentiel Binaire - Rapport de recherche INRIA.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique