RENPAR'14 / ASF / SYMPA
Hamamet, Tunisie, 10 – 13 avril 2002

CacheFS : un système de fichiers distribué all-cache

Momtchil MOMTCHEV et Philippe MARQUET

Laboratoire d'informatique fondamentale de Lille
Université des sciences et technologies de Lille
France
Momtchil.Momtchev@lifl.fr, Philippe.Marquet@lifl.fr

Résumé
CacheFS est un système de fichiers compatible VFS et complètement distribué, sans notion de serveur, dans lequel la capacité de stockage d'un nœud est géré comme un cache du système global. Une première implantation de ces principes a été réalisée sous GNU/Linux.

Mots-clés : all-cache, système de fichiers distribué, clusters

1. Systèmes de fichiers distribués

Les systèmes de fichiers distribués permettent l'accès, sans besoin de recopie locale explicite, à des ensembles de fichiers répartis sur plusieurs nœuds d'un réseau local. Leur importance est en progression constante, en concordance avec la tendance actuelle connectant des postes de travail peu coûteux au travers des réseaux rapides.
Les motivations menant à l'utilisation d'un système de fichiers distribué sont multiples. On peut simplement vouloir obtenir une vue globale et constante des ensembles de données sur un réseau. On peut rechercher des performances d'entrée/sorties plus élevées grâce à leur parallélisation. On peut aussi désirer une capacité de stockage supérieure.
CacheFS est un système de fichiers compatible VFS et complètement distribué, sans notion de serveur, dans lequel la capacité de stockage d'un nœud est géré comme un cache du système global.

Une vaste gamme de systèmes de fichiers distribués a déjà été proposée. Nous passons rapidement en vue leur caractéristiques dans le but de positionner CacheFS.
On peut distinguer deux classes de systèmes de fichiers distribués : ceux dans lesquelles tous les nœuds sont équivalents et ceux qui partitionnent les nœuds en nœuds clients et nœuds serveurs. La plupart des systèmes de fichiers distribués commerciaux disponibles (NFS, AFS, Coda) relèvent de la seconde catégorie. Bien que ces systèmes aient leurs avantages, ils ne profitent pas complètement de la nature d'un réseau de stations de travail et notamment de la capacité de stockage propre à chaque nœud. Toute station de travail moderne est équipée d'un disque dur. Dans une configuration classique de partage de données via un réseau, le disque dur local contient uniquement le système d'exploitation, ou parfois reste même non utilisé. Les données « utiles » sont stockées dans ce cas sur un des (ou plusieurs, si le système supporte cette possibilité) nœuds et souvent c'est la performance de ce nœud qui limite la performance en entrées/sorties de toutes les stations. De plus, la défaillance du nœud serveur implique la défaillance du réseau tout entier. La centralisation de toutes les opérations oblige aussi une synchronisation avec un grain assez fin, de laquelle résultent beaucoup de communications, comme illustré sur la figure 1.
Lombard et Denneulin [?] proposent une version de NFS dédiée aux grappes de PCs, nfsp, dans laquelle les données sont distribuées de façon statique sur les stations de travail alors que les métadonnées restent centralisées sur un ou plusieurs serveurs afin de maintenir une certaine compatibilité avec NFS. Bien qu'un tel système soit beaucoup moins gourmand en bande passante sur le réseau, il ne résout pas le problème du placement des données et ne pourra pas profiter des localités éventuelles.

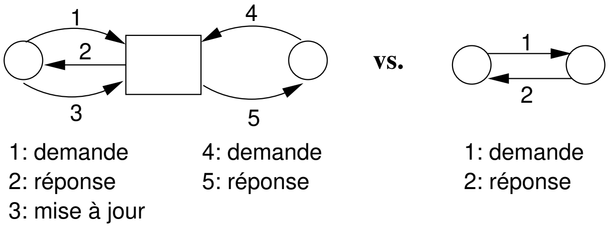


FIG. 1 – CacheFS permet d'assurer une meilleure localité et une réduction des communications. Dans le présent exemple, un premier nœud accède en lecture puis en écriture à une donnée puis un second noœud accède en lecture à cette même donnée. Les communications pour un système de fichiers distribué classique sont figurées à gauche, celles nécessaires dans le cas d'un système de fichiers distribué de type all-cache à droite.

Les environnements de déploiement ciblés par un système de fichiers distribué sont aussi nombreux. Un des critères fondamentaux caractérisant un environnement est le rapport entre le débit du réseau et le débit du disque dur local. Dans un réseau de PCs classiques d'aujourd'hui, les débits des disques durs (interface ATA100/ATA133) sont de l'ordre de 10 à 30 Mo/s en fonction du modèle avec des latences d'accès qui varient de 8 à 12 ms [**?**]. Les débits du réseau (réseau FastEthernet par exemple) vont de 3 Mo/s à 5 Mo/s avec des latences d'accès sous la milliseconde pour un réseau complètement disponible et jusqu'à la dizaine de millisecondes pour un réseau chargé. Un système de fichiers distribué tel CacheFS est donc adapté pour un environnement dans lequel les accès réseau sont moins rapides que les accès disque dur, pourvu que l'on transfère plusieurs kilo-octets à la fois.

2. Le principe de la mémoire all-cache

La performance de tout système de stockage de données peut être améliorée pour peu que l'on dispose d'un autre système de stockage de données qui soit plus rapide et qu'on va utiliser pour « cacher » le premier. Dans le cas des systèmes de fichiers distribués c'est le plus souvent la mémoire centrale de l'ordinateur qui est utilisée pour cette tâche. Bien que les systèmes de fichiers distribués de dernière génération puissent également utiliser le disque dur local pour ce faire [**?**], il s'agit bien souvent d'améliorer la tolérance aux pannes et non d'une utilisation de type cache.
L'idée même d'un cache suppose qu'il duplique les données déjà présentes sur un support différent. Dans le cas de plusieurs clients les mêmes données peuvent se retrouver avec plusieurs copies. Si la totalité des caches est d'une capacité de stockage comparable à la capacité du système central, alors une grande partie de l'information va se retrouver dupliquée dans les caches. À condition qu'on assure que toute unité de données ait au moins une copie dans au moins un cache, on pourra éliminer complètement le système central. Une telle organisation d'un système de stockage est appelé all-cache. En son temps, la société Kendall Square Research avait conçu et réalisé une machine parallèle autour d'une mémoire gérée sur ce principe [1].

3. CacheFS, un système de fichiers distribué all-cache

CacheFS est un système de fichiers distribué construit sur le principe de la mémoire all-cache. Tout nœud dans CacheFS est traité de la même façon et il fournit une certaine capacité de stockage au système. Cette capacité de stockage est gérée comme un cache, l'unité de base étant le bloc. La taille du bloc est constante à travers les nœuds d'un système donné, mais pourra varier entre deux installations à condition que le système d'exploitation le supporte.

4. Architecture et structures de données

Adressage des données

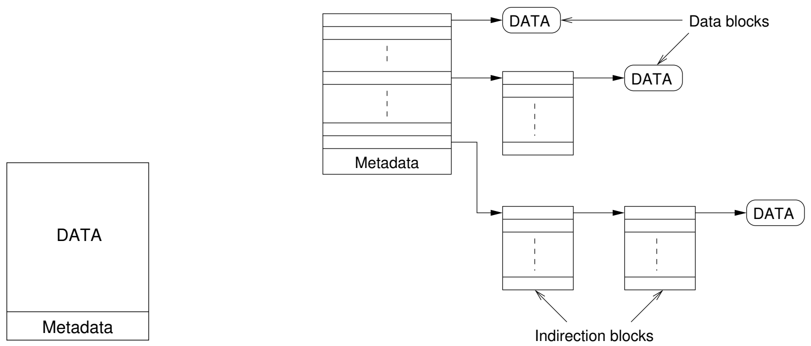
Les données sont adressées au sein des blocs. Un bloc est identifié de façon globalement unique par deux nombres : le numéro d'i-node et le numéro de bloc dans l'i-node.
Une adresse de bloc dans CacheFS est structurée de la façon suivante :

Bit 0	Bit 1	Bits 2..31	Bits 32..63
Bit de présence	Bit de propriété	Adresse physique si présent	Masque des copies si propriétaire

Les adresses physiques étant codées sur 30 bits, la taille des partitions physiques est limitée à 2^{30} blocs (4 To avec le bloc standard de 4 Ko). Le masque des copies indique les nœuds dans le réseau possédant une copie du bloc (la taille du champ limite le réseau à 32 nœuds). Le masque de copie est garanti à jour uniquement si le bit de propriété est à 1.
Il est intéressant de noter que la taille totale du système de fichiers n'est pas directement limitée, les blocs logiques n'étant jamais adressés. Cependant elle est indirectement limitée par le nombre maximal d'i-nodes (2^{32} sous UNIX/Linux sur une architecture 32 bits) et la taille maximale d'un fichier (toujours 2^{32} sous UNIX/Linux sur une architecture 32 bits).

Stockage des i-nodes

La table des i-nodes, traditionnellement présente sur les systèmes de fichiers classiques, est mal adaptée à un système all-cache parce qu'une telle organisation est susceptible de provoquer des allers/retours constants de cette table entre les nœuds. Cette même remarque est aussi valable pour les i-nodes qui décrivent des répertoires avec écritures fréquentes. Dans CacheFS tout i-node occupe un bloc entier, qui contient à la fois la structure de l'i-node, aussi bien que les tables d'indirection de premier ordre.
Plus de la moitié des i-nodes sur un système UNIX classique occupent moins d'un bloc [**?**] ce qui justifie un traitement spécial de ce cas. GFS [**?**] introduit la notion d'un i-node « stuffed », dans lequel le bloc qui contient la structure de l'i-node contient aussi l'information présente dans le fichier. Dans CacheFS les fichiers sont stockés dans des i-nodes « stuffed » quand leur taille ne dépasse pas la taille d'un bloc moins 128 octets (la taille de la structure de l'i-node). L'accès à ces fichiers ce fait avec la lecture d'un seul bloc. L'implémentation Linux de `mmap()` exige que les données dans les fichiers soient alignées sur le début d'un bloc, donc la structure des i-nodes est placée à la fin du bloc. Les i-nodes « stuffed » sont automatiquement transformés en i-nodes « unstuffed » quand leur taille dépasse le seuil. La figure 2 illustre ces deux types d'i-nodes.



(a) stuffed (b) unstuffed

FIG. 2 – Un i-node « stuffed » et un i-node normal.

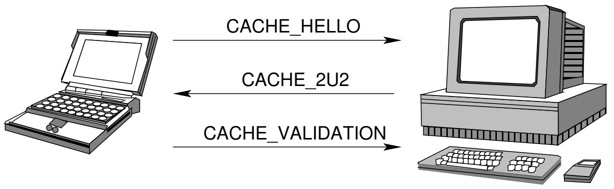


FIG. 3 – Communications lors d'un montage.

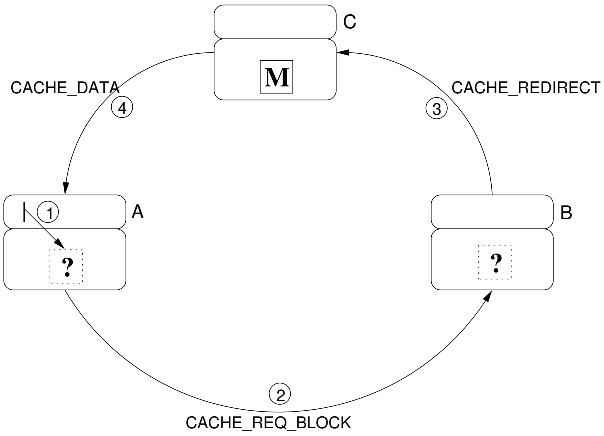


FIG. 4 – Communications lors d'une lecture.

5. Protocole réseau

Montage et démontage

Le montage est la seule opération dans CacheFS gérée de façon centralisée. Une gestion distribuée serait coûteuse à implémenter et étant donné le caractère exceptionnelle de l'opération, il y a peu de risque que ce mécanisme devienne un goulot d'étranglement. Un nœud unique est désigné maître des montages et il répond aux demandes de montages qui sont faites à l'aide de diffusions sur le réseau.

Deux formes de démontage sont prévues : le démontage faible et le démontage fort. Après un démontage faible, les blocs dont les seules copies se trouvent sur les nœuds déconnectés deviennent inaccessibles. Ce type de démontage permet de débrancher temporairement une machine du réseau. Lors d'un démontage fort tous les blocs avec copie unique sur le nœud sont diffusés vers les autres nœuds. Pour que le démontage fort soit possible, il faut que l'espace disponible sur les autres nœuds soit suffisant.

Lecture et écriture

La sémantique de la lecture et l'écriture suit les règles habituelles de gestion d'un cache : lors de la lecture les blocs à lire sont recopiés sur le disque dur local ; lors d'une écriture les copies existantes sur les autres nœuds sont invalidées.

Comme il peut arriver qu'un nœud ne connaisse pas l'emplacement d'un bloc (par exemple le bloc vient de changer de propriétaire suite à une écriture), un message de redirection est aussi prévu. Ce protocole est illustré à la figure 4. En cas de manque complète d'information sur le nœud, par exemple à un changement de propriétaire survenu alors que le nœud été déconnecté du réseau, la requête est envoyée à un nœud aléatoire. La requête est ensuite redirigée tant qu'elle n'arrive pas sur un nœud possédant la copie maître. Cette situation reste assez rare, parce que les i-nodes des fichiers peuvent hériter l'information de localisation du répertoire dans lequel il se trouvent. Ce nœud met également à

Accès à 80% dans 10 fichiers différents par chacun des clients

nb clients	accès locaux
2	80,15%
5	68,76%
10	65%
25	62,54%
50	61,18%
100	60,29%

Accès à 80% à 10 mêmes fichiers par tous les clients

nb clients	accès locaux
2	77,45%
5	60,84%
10	49,24%
25	34,78%
50	21,75%
100	13,6%

Accès aléatoire parmi tous les fichiers

nb clients	accès locaux
2	51,7%
5	24,04%
10	13,84%
25	8,41%
50	6,12%
100	4,85%

FIG. 5 – Pourcentages d'accès locaux pour différentes simulations de CacheFS. Les simulations considèrent toutes un ensemble de 500 fichiers. Chacun des fichiers contient 20 blocs.

jour le masque des copies, qui permet l'invalidation.
Le choix du bloc à remplacer dans le cache en cas de cache plein est aléatoire, avec une préférence pour les blocs en mode copie, leur suppression ne nécessitant pas une transfert vers un autre nœud.

Création d'un i-node
Une table des descriptions des i-nodes, présente sur tout nœud, référence l'état des numéros d'i-nodes qui sont disponibles, occupés, réservés par le nœud ou réservés par un autre nœud. La réservation de blocs de numéros permet la création rapide de fichiers, sans besoin de synchronisation à travers le réseau.

6. Une simulation de CacheFS

Une simulation de CacheFS a été menée. Nous avons cherché à mesurer le rapport entre les accès locaux et distants.
Trois types de clients synthétiques ont été utilisés :
- des clients réalisant 95% de lectures et 5% d'écritures avec 80% des accès concentrés sur 10 fichiers, ces 10 fichiers étant différents sur chaque nœud ;
- des clients réalisant 95% de lectures et 5% d'écritures avec 80% des accès concentrés sur les mêmes 10 fichiers ;
- des clients réalisant 95% de lectures et 5% d'écritures avec des accès aléatoires.

Les tables de la figure 5 synthétisent les résultats de ces simulations. Il apparaît que CacheFS se montre moins gourmand en bande passante sur le réseau qu'un système centralisé. Cette économie de bande passante permet une meilleure montée en charge.

7. CacheFS sous GNU/Linux

Une première implémentation des principes de CacheFS a été faite sous GNU/Linux [**?**].
L'implémentation, qui est complètement en espace noyau, est faite sous la forme d'un module Linux. Elle est construite suivant une architecture client/serveur. Le client est implémenté en librairie d'appels système. Le serveur est implanté sous la forme de threads noyau indépendants (figure 6). C'est le serveur qui effectue le nettoyage du cache quand le taux d'occupation atteint un seuil prédéterminé.
Un outil de création de partitions « vierges », `mkcachefs`, a été implémenté. Une fois les structures de données d'une partition ont été créée, elle est « peuplée » au moment du montage à l'aide d'une lecture fictive du répertoire racine. De cette façon une copie du répertoire racine est toujours présent sur chaque nœud.
La table des i-nodes est également transférée en entier pendant le montage, les états des numéros d'i-nodes étant susceptibles d'avoir changé depuis le dernier démontage.
Des retours sur le fonctionnement de CacheFS ont été exportés vers l'espace utilisateur à travers `/proc`.

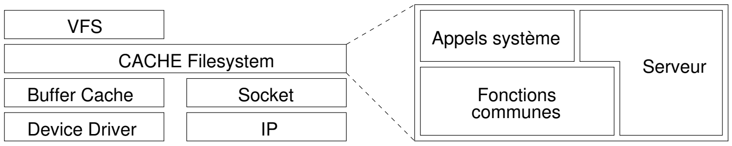


FIG. 6 – Architecture de l'implémentation Linux de CacheFS.

8. Conclusion et travaux futurs

Nous avons présenté le principe d'un système de fichiers distribué all-cache. Les systèmes de fichiers distribués all-cache s'avèrent plus performants que les systèmes de fichiers distribués traditionnels dans les environnements dans lesquels les liens entre les nœuds sont moins rapides que les accès au disques locaux, ce qui est notamment le cas dans les réseaux de PCs.
Une première version a été prototypée sous GNU/Linux. Nos travaux futurs vont mesurer les performances effectives de CacheFS dans différents cas de figure.
Un travail d'optimisation du code du serveur afin que la lecture des données aussi bien que leur émission se fasse sans recopie de tampons et en parallèle en cas de plusieurs lectures est envisagée. La mise en œuvre de « read-ahead » s'avère également cruciale pour une meilleure performance.
Une implémentation plus complète pourra également supporter la tolérance aux pannes, les systèmes all-cache étant particulièrement adaptés à cette tâche [**?**].
Une façon plus intelligente de choisir le nœud lors d'une lecture est aussi envisageable, par exemple en fonction du propriétaire du nœud qui contient le répertoire contenant le i-node lu.

Remerciements

Les auteurs tiennent à remercier Patrice Boulanger pour sa participation à la conception et au développement de la première version de CacheFS.

Bibliographie

1. Michel Banâtre, Alian Gefflaut, Anne-Marie Kermarrec, and Christine Morin. COMA : an opportunity for building fault-tolerant scalable shared memory multiprocessors. Research Report 964, IRISA, Rennes, France, décembre 1995.
2. Kendall Square Research, Waltham, Massachusetts. *Kendall Square Research Technical Summary*, 1992.
3. J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *Communications of the ACM*, 10(1) :3–25, février 1992.
4. Pierre Lombard and Yves Denneulin. nfsp : A distributed NFS server for clusters of workstations. In *Int'l Parallel and Distributed Processing Symposium, IPDPS 2002*, Fort Lauderdale, FL, avril 2002.
5. Momtchil Momtchev and Patrice Boulanger. CacheFS : Consistent All-CacHE File System. Mémoire de maîtrise, Université des sciences et technologies de Lille, juin 2000.
6. Kenneth Preslan, Andrew Barry, Jonathan Brassow, Grant Erickson, Erling Nygaard, Christopher Sabol, Steven Soltis, David Teigland, and Mathew O'Keefe. A 64-bit, shared disk file system for Linux. In *Sixteenth IEEE Mass Storage Systems Symposium held jointly with the Seventh NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, CA, mars 1999.
7. Eugene Ra. Storagereview.com's testbed3. Technical report, StorageReview.com's, 2001.
8. Andrew Tanenbaum. *Distributed Operating Systems*. Prentice-Hall, 1992.