ZFS et Btrfs, systèmes de fichiers ultimes à l'heure des big data ?

CNAM - ENG 221

Stéphane Legrand < stephleg@free.fr >

Juin 2014

Résumé

ZFS et Btrfs sont des systèmes de fichiers nouvelle génération. Leur développement a pour but d'apporter des solutions aux limites des systèmes de fichiers classiques. Et d'apporter des fonctionnalités capables de répondre aux besoins actuels. A savoir assurer la gestion de volumes de données de plus en plus gigantesques tout en maintenant l'intégrité des données et sans rendre l'administration des systèmes inutilement complexe.

Nous verrons que ZFS et Btrfs apportent tous les deux des fonctionnalités similaires qui remplissent ces objectifs. Néanmoins ZFS conserve une longueur d'avance. Ses possibilités avancées, son architecture, sa fiabilité éprouvée et sa simplicité d'utilisation apparaissent supérieures à ce que peut offrir Btrfs à l'heure actuelle.

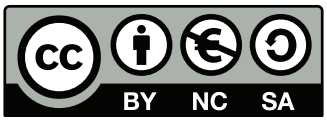

Table des matières

1 Introduction **9**

2 ZFS **11**
2.1 Historique 11
2.2 Fonctionnalités 12
Capacités 12
Support natif du RAID 12
Intégrité des données 13
« Auto-réparation » 14
Réplication explicite 14
Contrôle préventif des périphériques de stockage 15
Répartition dynamique 15
Compression 15
Quotas et réservation d'espace 16
Administration du système de fichiers 16
Instantanés et clones 17
Envoi/Réception 17
Portabilité 18
Déduplication 18
2.3 Architecture 18
« Copie-sur-écriture » 18
Composition des systèmes de fichiers 19
Composants fonctionnels 20
SPA 21
DMU 21
ZPL 22

3 Btrfs **25**
3.1 Historique 25
3.2 Fonctionnalités 25
Capacités 25
Support natif du RAID 26

Intégrité des données 27
« Auto-réparation » 27
Contrôle préventif des périphériques de stockage 27
Compression 28
Quotas et réservation d'espace 28
Administration du système de fichiers 28
Instantanés et clones 28
Envoi/Réception 29
Déduplication 29
Défragmentation 29
Clone et Périphérique « graine » 30
3.3 Architecture 30
« Copie-sur-écriture » 30
Composition des systèmes de fichiers 30
Composants fonctionnels 31
B-tree 31

4 Comparatif **33**
4.1 Performances 33
4.2 Supports 33
4.3 Licences 34
4.4 Maturité 34
4.5 Fonctionnalités 34
4.6 Mise en pratique 35

5 Conclusion **37**

Références **39**

A Arbre de Merkle **41**

B « Copie-sur-écriture » **43**

C Solutions commerciales **45**
C.1 ZFS 45
EraStor 45
iXsystems 45
Nexenta 45
Oracle 45
C.2 Btrfs 46
Netgear 46

D Tests pratiques **47**

D.1 ZFS 47
Configuration 47
Création du pool 47
Création d'un système de fichiers (dataset) 48
Ecriture et test de corruption de données 49
Instantané 51
Remplacement d'un disque 53
D.2 Btrfs 54
Configuration 54
Création du volume 54
Ecriture et test de corruption de données 56

Glossaire

ACL Access Control List (Liste de Contrôle d'Accès). Système de gestion des droits d'accès aux fichiers. 22

B-tree B-tree (Arbre B). Structure de données en arbre équilibré. Les données sont stockées sous une forme triée ce qui permet de limiter le nombre de comparaisons nécessaires lors de la recherche d'un élément. 25

Btrfs B-tree file system (système de fichiers B-tree). 10, 25, *voir* B-tree

CDDL Common Development and Distribution License (Licence Commune de Développement et de Distribution) : `http://opensource.org/licenses/CDDL-1.0`. 11, 34

DMU Data Management Unit (Unité de Gestion des Données). 20, 21

FUSE Filesystem in Userspace (Système de fichiers en espace utilisateur) : `http://fuse.sourceforge.net/`. 11

GNU GNU's Not UNIX (GNU n'est pas UNIX). Projet informatique initié par Richard Stallman dont le but initial est de créer un système d'exploitation complet et entièrement libre. Pour parvenir à cet objectif, ce projet a donné naissance à toute une myriade de logiciels libres. *voir* UNIX

GPL GNU General Public License (Licence Publique Générale GNU). Licence qui fixe les conditions légales de distribution de logiciels libres et en particulier ceux du projet GNU. 34, *voir* GNU

inode Index node (noeud d'index). Un inode est une structure utilisée par un système de fichiers pour stocker les méta-données associées à un fichier, comme son propriétaire par exemple. 16, 22, 26, 31

iSCSI internet Small Computer System Interface (Petite Interface de Système Informatique sur internet). Protocole de stockage en réseau destiné à relier les installations de stockage de données. 20

LZ4 Lempel Ziv 4. Algorithme de compression sans perte de données mis au point par Yann Collet. Cette méthode met l'accent sur ses performances en vitesse de compression/décompression. 15, 18

LZJB Lempel Ziv Jeff Bonwick. Algorithme de compression sans perte de données mis au point par Jeff Bonwick (un des concepteurs de ZFS). 15

LZO Lempel-Ziv-Oberhumer. Algorithme de compression sans perte de données qui se focalise sur la vitesse de décompression. 28

NVRAM Non Volatile Random Access Memory (Mémoire à Accès Aléatoire Non Volatile). Mémoire vive qui ne perd pas ses informations lorsque son alimentation électrique est interrompue. 23

POSIX Portable Operating System Interface UNIX (Interface Portable de Système d'Exploitation UNIX). Famille de standards destinés aux logiciels fonctionnant sur les systèmes d'exploitation de type UNIX. 20, *voir* UNIX

RAID Redundant Array of Independent (or inexpensive) Disks (Ensemble redondant de disques indépendants/bon marchés). 10, 12, 26

SPA Storage Pool Allocator (Allocateur de Ressource de Stockage). 20, 21

SSD Solid-State Drive (Disque à Etat Solide). Disque dur où les données ne sont pas sauvegardées sur un support magnétique mais sur de la mémoire à semi-conducteurs à l'état solide (mémoire dite flash). 18

VFS Virtual File System (Système de Fichiers Virtuel). Interface générique entre le système d'exploitation et le système de fichiers. 20

ZFS Z File System (Système de fichiers Z). 10, 11

ZIL ZFS Intent Log (Journal d'Intentions ZFS). 23, 47

ZLIB Librairie open-source de compression/décompression de données : `http://zlib.net/`). 28

ZPL ZFS POSIX Layer (Couche POSIX de ZFS). 20, 21, *voir* POSIX

Chapitre 1

Introduction

Un système de fichiers est un composant logiciel qui fait l'interface (voir figure 1.1) entre l'organisation logique et l'organisation physique des fichiers et répertoires d'un système informatique. L'organisation logique correspond à la vision utilisateur. Pour celui-ci, les fichiers et répertoires sont organisés sous la forme d'une arborescence. Un fichier ou un répertoire est localisé via un chemin dans cette arborescence. L'organisation physique correspond à la façon dont sont stockées les données sur les supports physiques de stockage (disque dur, CD-ROM, clef USB, disquette ...).

Un système de fichiers assure les opérations de base nécessaires à la gestion des données (création, modification et suppression). Il doit également assurer la sécurité et l'intégrité des données. Y compris dans la mesure du possible en cas de panne matériel ou logiciel.

Un système de fichiers assure par ailleurs une gestion des méta-données. Ce sont des données supplémentaires rattachées aux ...données. Il s'agit par exemple des droits d'accès, de la date de dernière modification, de l'indication de l'utilisateur et du groupe propriétaires du fichier, de la taille du fichier, de son type (lien symbolique par exemple) ...

Les systèmes de fichiers classiques ont aujourd'hui plusieurs inconvénients :

— Leurs limites théoriques de capacités (taille maximale d'un fichier ou d'une partition par exemple) deviennent problématiques avec la capacité des périphériques de stockage actuels et à l'heure où les volumes de données à gérer deviennent gargantuesques.

— Ils n'assurent aucune défense contre les corruptions de données silencieuses. Un problème au niveau du disque, du contrôleur, des câbles, du gestionnaire de périphérique ...peut conduire à une corruption de données.

— Leur administration peut être compliquée. Il faut gérer des partitions,

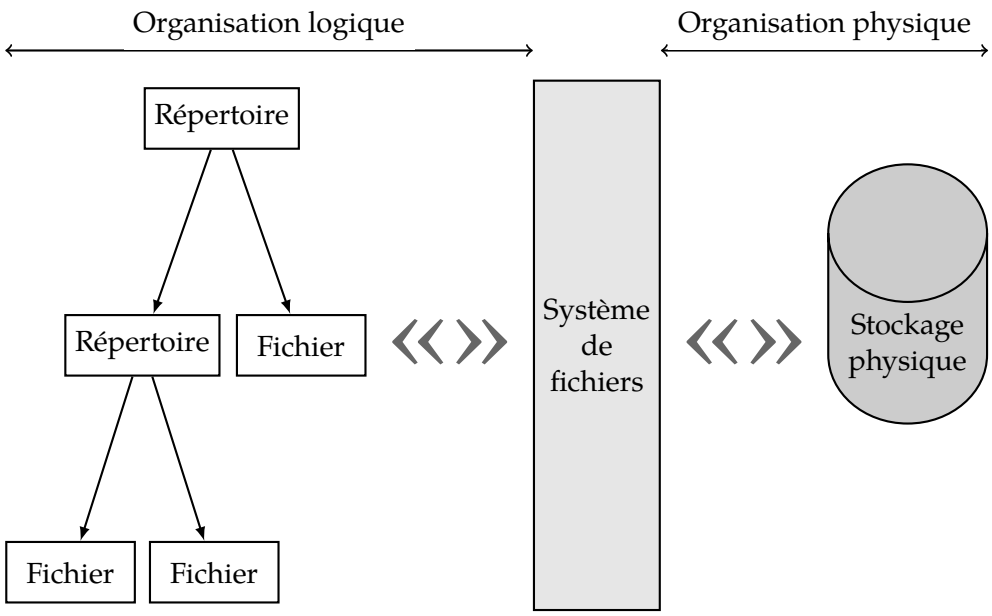

FIGURE 1.1 – Un système de fichiers est une interface.

des labels, procéder à des réservations d'espace, il peut être délicat de réajuster les tailles des partitions, il est nécessaire de gérer des fichiers de configuration (pour les points de montage notamment), les volumes doivent être administrés avec des outils distincts …

— Avec l'explosion des quantités de données à gérer, les durées nécessaires à certaines opérations deviennent prohibitives : temps de reconstruction RAID (Redundant Array of Independent (or inexpensive) Disks), temps nécessaire pour les sauvegardes, vérification des systèmes de fichiers après un arrêt inopiné ….

Les systèmes de fichiers nouvelle génération comme Z File System (ZFS) et B-tree file system (Btrfs) se proposent d'apporter des solutions pérennes à ces problèmes ainsi que des fonctionnalités adaptées aux besoins actuels.

Nous allons maintenant découvrir plus en détails les moyens mis en oeuvre par ces deux systèmes de fichiers pour parvenir à ces objectifs.

Chapitre 2

ZFS

Nous allons tout d'abord voir un historique du développement de ZFS pour ensuite décrire ses principales fonctionnalités. Puis nous détaillerons une partie des composants de son architecture technique. On trouvera également un test effectué sous système FreeBSD en annexe D.1. Et quelques exemples de solutions commerciales en annexe C.1.

2.1 Historique

ZFS a aujourd'hui 13 ans d'existence [1] :

- 2001 Le développement débute chez la société Sun Microsystems avec 2 ingénieurs (Jeff Bonwick et Matthew Ahrens).
- 2005 Le code source est rendu disponible sous licence libre CDDL (Common Development and Distribution License).
- 2006 Début du développement de ZFS sur FUSE (Filesystem in Userspace) pour Linux.
- 2008 ZFS est inclus avec le système d'exploitation FreeBSD version 7.0 [1].
- 2008 Début du développement de la version native pour Linux (en mode noyau et non plus en mode utilisateur).
- 2008 Disponibilité de la série 7000 du produit conçu par Sun Microsystems dédié au stockage ZFS.
- 2009 Rachat de Sun par la société Oracle.
- 2010 Oracle cesse sa contribution au code open-source de ZFS. A partir de cette date, on a d'un côté une version de ZFS développée par Oracle sous licence propriétaire et sans diffusion du code source. Cette version fermée est intégrée dans certains produits Oracle. Et d'un autre côté une version open-source basée sur la dernière version libre disponible.

1. `http://www.freebsd.org/releases/7.0R/relnotes.html#FS`

2013 La version en mode noyau de ZFS pour Linux est considérée comme stable et utilisable en production.

2013 Création du projet OpenZFS [2] dont l'objectif est de centraliser et de coordonner les développements sur la version open-source de ZFS.

2.2 Fonctionnalités

Capacités

Comme l'utilisation d'une implémentation d'un système de fichiers peut s'étendre sur plusieurs décennies, et prenant en compte l'évolution des capacités de stockage, les concepteurs de ZFS ont pris le parti de coder les éléments d'adressage sur 128 bits [2]. Ce qui autorise des capacités gigantesques [3] (voir figure 2.1) !

Nombre d'entrées dans un répertoire	2^{48}
Taille d'un fichier	2^{64} octets ($\approx 16\,000\,000$ To)
Taille d'une ressource de stockage (taille d'un pool)	2^{78} octets ($\approx 256\,000\,000\,000$ To)
Nombre de périphériques dans un pool	2^{64}
Nombre de pools dans un système	2^{64}
Nombre de systèmes de fichiers dans un pool	2^{64}
Nombre d'instantanés (« snapshots »)	2^{48}

FIGURE 2.1 – Limites théoriques de ZFS

Support natif du RAID

ZFS intègre nativement le support de plusieurs niveaux de RAID :

— RAID-0 qui correspond à une agrégation des périphériques de stockage avec une répartition des données entre ces périphériques. Ce niveau de RAID n'assure aucune sécurité puisqu'il suffit qu'un seul des périphérique tombe en panne pour perdre l'intégralité des données.

2. `http://open-zfs.org/`
3. `http://en.wikipedia.org/wiki/ZFS`

— RAID-1 qui correspond à une copie intégrale des mêmes données sur chaque périphérique de stockage (mode « miroir »). On a ainsi une très grande sécurité puisqu'il suffit qu'un seul périphérique reste en fonction pour préserver l'intégralité des données. Inconvénient, la capacité totale de stockage n'augmente jamais avec l'ajout d'un périphérique puisque celui-ci servira à stocker une autre copie des données. ZFS permet l'utilisation de plusieurs périphériques miroirs afin d'accroître d'autant la sécurité des données.
— RAID-Z qui est l'équivalent d'un mode RAID-5 amélioré. Tout comme le RAID-5 traditionnel, les parités calculées à partir des données sont sauvegardées de manière répartie sur les différents disques. Ce qui permet de reconstituer les données si l'un des périphériques tombe en panne. RAID-Z offre l'avantage de ne pas connaître le problème du « trou d'écriture » (« write hole ») inhérent au RAID-5 traditionnel. Ce problème peut survenir si une coupure d'alimentation survient pendant l'écriture [4]. Dans ce cas, il est impossible de déterminer quels blocs de données ou quels blocs de parités ont effectivement été sauvegardés ou non. Et il est impossible de déterminer de manière certaine quelles informations sont incorrectes - les parités ou bien les données. Par ailleurs, RAID-Z offre la possibilité d'avoir des parités doubles ou triples. Ce qui permet aux données de survivre à la défaillance de deux ou trois périphériques.

Intégrité des données

Toutes les données et les méta-données ont une somme de contrôle calculée via un des algorithmes disponibles (Fletcher2, Fletcher4, SHA256) [3]. Par défaut, c'est un algorithme Fletcher qui est utilisé pour le calcul de la somme de contrôle des données. Pour les méta-données seul l'algorithme SHA256 est utilisé. Ces sommes de contrôle sont vérifiées à chaque lecture de bloc et sont mises à jour à chaque écriture. Ce mécanisme permet de détecter avec une très grande probabilité les corruptions de données silencieuses qui seraient provoquées par des défauts d'un disque, du contrôleur, d'un câble, du gestionnaire de périphérique ou du firmware.

Les sommes de contrôle sont stockées dans l'arborescence des blocs au niveau des noeuds d'adresses [4] (voir figure 2.2). La racine de cet arbre est appelée überblock [2]. Ce bloc racine est le seul à stocker lui-même sa somme de contrôle (puisqu'il n'a pas de parent). Le stockage des sommes de contrôle dans les blocs parents permet une auto-validation puisque chaque somme de

4. `http://www.raid-recovery-guide.com/raid5-write-hole.aspx`

contrôle est elle-même incluse dans la somme de contrôle du bloc parent. Cette structure hiérarchique forme un arbre de Merkle auto-validant (voir annexe A).

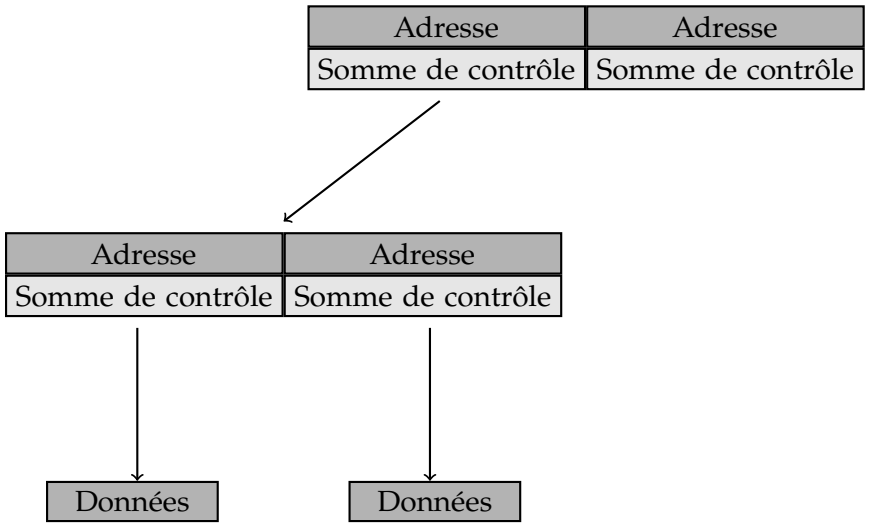

FIGURE 2.2 – Sommes de contrôle dans l'arbre des blocs

« Auto-réparation »

Les sommes de contrôle décrites précédemment permettent dans certaines circonstances une « auto-réparation » des données [2]. En effet, si le système de fichiers détecte une corruption de données, c'est à dire une absence de correspondance entre la somme de contrôle calculée à la lecture et la somme de contrôle sauvegardée, il peut tenter de lire une autre copie des données. A condition que cette copie existe par ailleurs (par exemple sur un disque miroir) et qu'elle soit elle-même valide. Le système de fichiers peut ensuite réparer automatiquement la version endommagée des données grâce à la copie valide.

Réplication explicite

En parallèle de la sécurité des données assurée par la mise en place d'un mode RAID-Z ou de disques miroirs, il est possible de configurer un système de fichiers ZFS pour qu'il conserve jusqu'à 3 copies physiques des blocs de données. Ces copies seront autant que possible placées sur des périphériques différents ou bien, dans le cas d'une configuration mono-disque, à des emplacements différents du périphérique. Les méta-données sont par ailleurs toujours copiées au minimum en deux exemplaires.

Ce mécanisme de réplication est avant-tout une protection supplémentaire contre d'éventuels secteurs disques défectueux et non pas contre une panne complète de disque.

Contrôle préventif des périphériques de stockage

Il est possible (et recommandé) de lancer périodiquement l'exécution d'un contrôle (« scrubbing ») des disques. Cette opération permet de découvrir les erreurs latentes pendant qu'elles peuvent être encore corrigées. La commande vérifie l'intégrité de toutes les données. Elle parcourt les méta-données du pool et lit chaque copie de chaque bloc (copies miroir, parités RAID-Z, blocs dupliqués explicitement). Elle vérifie également chaque copie avec sa somme de contrôle et répare les données si besoin et dans la mesure du possible.

Cette procédure peut s'effectuer avec les systèmes de fichiers en ligne. De manière à diminuer l'impact sur les performances, ses entrées/sorties sont affectées d'une faible priorité. Par ailleurs, la périodicité de cette vérification est définie par l'administrateur. Il est ainsi possible d'effectuer un contrôle graduel du pool sur un mois ou sur un trimestre par exemple.

Répartition dynamique

ZFS distribue automatiquement les écritures à travers tous les périphériques disponibles dans le pool. Lorsqu'un nouveau périphérique est ajouté, il n'est pas nécessaire de migrer les données existantes. Les anciennes données restent sur les anciens périphériques, les nouvelles données seront automatiquement réparties entre les anciens périphériques et le nouveau. Le mécanisme de « copie-sur-écriture » (« copy-on-write ») se chargera de migrer les anciennes données au fur et à mesure des écritures.

La répartition dynamique permet également à ZFS de gérer les périphériques hétérogènes [5]. Il peut en effet allouer les blocs de données sur les disques de façon proportionnelle à leurs performances respectives.

Compression

Il est possible de configurer un système de fichiers pour que les blocs de données soient compressés/décompressés à la volée de manière transparente. Plusieurs méthodes de compression sont disponibles : LZJB (Lempel Ziv Jeff Bonwick), Gzip et LZ4 (Lempel Ziv 4). Il faut noter que l'utilisation de cette option permet généralement d'améliorer les performances du système de fichiers. En effet, cela entraîne une réduction des entrées/sorties disques. Ce qui permet un gain de temps global puisque ces opérations restent relativement

lentes par rapport aux traitements supplémentaires effectués par le processeur pour compresser/décompresser les données.

Quotas et réservation d'espace

Par défaut, un système de fichiers ZFS croît et décroît au fur et à mesure que les utilisateurs ajoutent ou suppriment des données (dans la limite des ressources de stockage) [2]. Si nécessaire, l'administrateur peut imposer des quotas et des réservations d'espace disque sur les systèmes de fichiers de manière à éviter une utilisation abusive.

Les quotas limitent la quantité d'espace qu'un dataset et éventuellement ses descendants (instantanés par exemple) peuvent utiliser. Ils permettent également de limiter l'espace pour un utilisateur ou un groupe en particulier.

Il existe 4 types de quotas :
- Le « Refquota » qui limite l'espace qu'un dataset peut utiliser. Toutefois cette limite n'inclue pas l'espace utilisé par les descendants.
- Le quota global (« general quota ») qui est l'équivalent de « refquota » mais qui inclu les descendants.
- Le quota utilisateur qui limite l'espace pour un utilisateur donné.
- Le quota groupe qui limite l'espace utilisable pour un groupe d'utilisateurs donné.

Et il existe 2 types de réservation d'espace :
- La propriété « reservation » qui permet de réserver un espace minimum garanti pour un dataset et ses descendants.
- La propriété « refreservation » qui est l'équivalent de « reservation » mais sans inclure les descendants.

Comme les systèmes de fichiers n'ont pas de taille fixe, il n'est plus possible d'allouer statiquement les méta-données lors de la création du système de fichiers. Les structures comme les inodes (index nodes) sont par conséquent allouées et libérées dynamiquement au fur et à mesure de la création et de la suppression des fichiers.

Administration du système de fichiers

Deux commandes suffisent à l'administration de ZFS :
- « zpool » destinée à la gestion des pools (création, destruction, import/export, ajout de stockage, visualisation de l'état et des performances).
- « zfs » dédiée à la gestion des systèmes de fichiers (création, destruction, montage, gestion des options comme l'activation de la compression, instantané, clone, sauvegarde).

L'administrateur peut définir une politique de stockage pour chaque dataset (utilisation d'instantanés, activation ou non de la compression, réalisation des sauvegardes, application de quotas ...). Comme la création de dataset est très peu coûteuse en terme de ressources, il est possible d'en créer autant que nécessaire dans la limite (confortable) des 2^{48} possibles par pool. Les dataset deviennent ainsi des points de contrôle dans l'administration du système informatique.

Par ailleurs l'administrateur peut configurer pour certains dataset des délégations de droits d'administration auprès d'une sélection d'utilisateurs. Il peut ainsi donner l'autorisation à un utilisateur de créer des instantanés ou de créer ses propres sous-dataset. L'administrateur peut également définir un ensemble de droits sous une appellation et appliquer des délégations sur cet ensemble. Si les droits contenus dans cet ensemble sont par la suite modifiés, ce changement sera appliqué à l'ensemble des utilisateurs bénéficiant de cette délégation.

Instantanés et clones

Les instantanés (« snapshots ») sont une copie en lecture seule d'un système de fichiers à un instant T. La création d'un instantané est immédiate (quelle que soit la taille du système de fichiers) et il est possible d'en créer un nombre quasi illimité. Aucun espace de stockage supplémentaire n'est utilisé lors de la création. Les blocs sont en effet recopiés uniquement lorsqu'ils sont modifiés. Le contenu des instantanés est accessible dans un répertoire dédié appelé « .zfs/snapshot » situé à la racine de chaque système de fichiers. Ce qui permet aux utilisateurs de récupérer leurs fichiers sans intervention de l'administrateur.

Il est possible d'effectuer un retour arrière (un « rollback ») d'un instantané afin de revenir sur toutes les modifications introduites après sa création.

Un clone est une copie en écriture d'un instantané. Un clone est toujours créé à partir d'un instantané. Il peut par exemple être utilisé pour stocker plusieurs copies privées de données principalement partagées (installations logiciels, dépôts de code source, clients sans disque, machines virtuelles).

Envoi/Réception

Les instantanés peuvent être envoyés/reçus sous la forme d'un flux de données grâce aux commandes d'envoi (« zfs send ») et de réception (« zfs receive ») de ZFS [6]. Il est possible soit d'envoyer la totalité des données de l'instantané (copie complète) soit d'envoyer uniquement les différences entre deux instantanés (copie incrémentale). Cette fonctionnalité peut être utilisée pour

effectuer des sauvegardes ou assurer une réplication des données en local ou à distance.

Portabilité

ZFS utilise un format neutre, indépendant du matériel utilisé. Il est donc possible de passer d'un processeur x86 à un processeur SPARC sans aucune difficulté.

Ce système de fichiers est également disponible sur plusieurs familles de systèmes d'exploitation : Solaris, FreeBSD [5], Linux, Mac OS X.

Déduplication

Il est possible de configurer un dataset pour qu'il utilise la fonction de déduplication des données. Lorsque ZFS détectera des blocs de données redondants, ce bloc ne sera stocké qu'une seule fois ce qui permettra ainsi d'économiser de l'espace de stockage. Il s'agit donc d'une forme de compression. Si cette fonction paraît séduisante sur le papier, il faut garder à l'esprit qu'elle consomme énormément de mémoire vive. Il est en effet dans ce cas recommandé de disposer de 5 gigaoctets de mémoire vive par téraoctet de stockage [7]. Dans la pratique, il sera donc souvent préférable de ne pas activer cette option et d'opter pour l'utilisation d'une vraie compression via LZ4 par exemple.

2.3 Architecture

« Copie-sur-écriture »

ZFS utilise la technique de « copie-sur-écriture » [8] [4]. A chaque fois qu'un bloc de données doit être modifié, il est tout d'abord copié et n'est jamais directement modifié. La copie du bloc contient la nouvelle version des données et les différents pointeurs de blocs sont mis à jour de manière à adresser cette nouvelle version. L'ancien bloc n'est pas supprimé, il est simplement marqué comme libre à condition qu'il ne soit plus du tout référencé (ce qui peut malgré tout être le cas avec les instantanés par exemple). La figure B.1 illustre ce mécanisme lors d'une mise à jour de donnée.

Le procédé de « copie-sur-écriture » a l'inconvénient de provoquer une forte fragmentation des disques ce qui peut avoir un impact négatif non négligeable sur les performances. Il faut toutefois noter que cet inconvénient devient apparemment un avantage avec les disques SSD (Solid-State Drive) [6]. ZFS utilise

5. Le support de ZFS sur NetBSD et OpenBSD semble pour le moment expérimental.

6. `http://storagegaga.wordpress.com/2011/08/22/copy-on-write-and-ssds-a-better-match-than-other-file-systems/`

une technique basée sur un partitionnement en différentes « régions » des périphériques virtuels (« metaslab »[7]) afin de réduire cet effet de fragmentation.

Composition des systèmes de fichiers

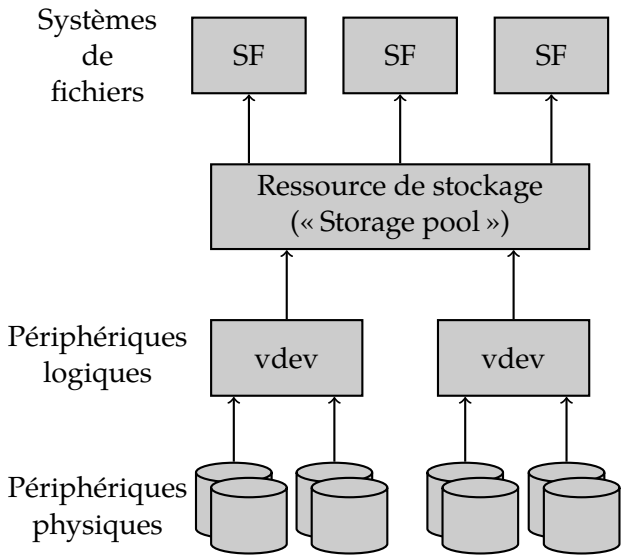

FIGURE 2.3 – Les éléments qui composent un système de fichiers

Un système de fichiers ZFS s'appuie sur plusieurs niveaux de différents éléments [2] (voir figure 2.3) :

— Niveau 1 : Les périphériques physiques qui peuvent être des fichiers[8], des partitions ou des disques durs.
— Niveau 2 : Le périphérique virtuel (« vdev ») est l'unité de base de stockage des données [9]. Il peut correspondre à un périphérique physique unique ou bien à un ensemble de périphériques physiques. C'est le cas pour les vdevs en mode miroir ou en mode RAID-Z.
— Niveau 3 : Une ressource de stockage (« storage pool ») est un regroupement de périphériques virtuels. La capacité totale de stockage sera égale à la somme des capacités des vdevs.
— Niveau 4 : Les systèmes de fichiers sont de deux types. Les dataset qui sont une sorte de partition du pool, par défaut sans taille définie. Dans la pratique, ce sera principalement ce type de système de fichiers qui sera utilisé. Et les volumes qui sont également une sorte de partition du pool mais de taille fixe. Un volume sera vu par le système d'exploitation comme un périphérique de stockage « brut » (« raw storage device »). Ce

7. `https://blogs.oracle.com/bonwick/en/entry/space_maps`
8. Uniquement à des fins de tests. Ne pas utiliser en production …

type de système de fichiers peut servir dans le cadre de l'utilisation d'un disque iSCSI (internet Small Computer System Interface) par exemple.

Le concept de ressource de stockage est central dans la conception de ZFS. A l'origine, chaque système de fichiers gérait un seul disque physique. Avec les besoins croissants d'espace, de débit et de fiabilité la solution adoptée a été d'ajouter la notion de volume afin d'agréger des disques physiques. Mais le système de fichiers et le gestionnaire de volumes étaient toujours deux éléments séparés. Les problèmes inhérents à l'interface entre ces deux composants ne pouvaient pas être corrigés.

Dès l'origine de ZFS il a donc été décidé d'intégrer les fonctions d'un gestionnaire de volumes grâce à ce concept de ressource de stockage. L'idée est de permettre ainsi une gestion similaire à celle de la mémoire vive [2]. En effet, lorsque l'on souhaite augmenter cette mémoire, il suffit d'ajouter des barrettes pour que la nouvelle mémoire soit immédiatement disponible pour le système. Avec ZFS il suffit d'ajouter un disque pour qu'il soit quasi-immédiatement disponible pour le système.

Une ressource de stockage est donc une collection de périphériques dans laquelle les systèmes de fichiers peuvent venir puiser lorsque nécessaire. Il n'y a pas de partitions à gérer, l'augmentation et la réduction de la taille se fait automatiquement, le débit maximum d'entrée/sortie est toujours disponible et tous les périphériques sont par défaut partagés.

Composants fonctionnels

La figure 2.4 montre les principaux composants fonctionnels de ZFS [2]. Le gestionnaire de périphérique exporte un périphérique de blocs auprès de l'allocateur de ressource de stockage (SPA). Le SPA gère l'allocation des blocs et les entrées/sorties et exporte les adresses virtuelles des blocs alloués et des blocs libres auprès de l'unité de gestion des données (DMU). La DMU transforme les blocs adressés virtuellement du SPA en une interface objet transactionnel auprès de la couche POSIX (Portable Operating System Interface UNIX) de ZFS appelée ZPL. Enfin, la ZPL implémente un système de fichiers POSIX au dessus des objets DMU et exporte les opérations vnode [9] auprès de la couche des appels système.

9. Les opérations vnode font partie de l'interface de système de fichiers virtuel (VFS (Virtual File System)). Par exemple une des opérations vnode est vop_create() qui crée un nouveau fichier.

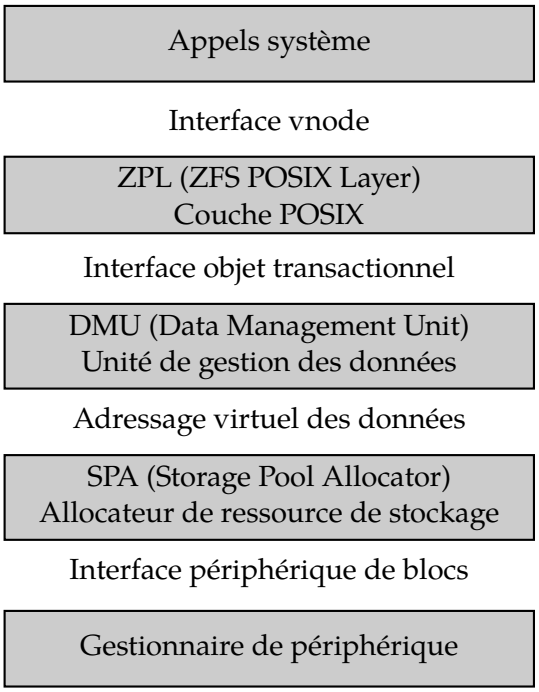

FIGURE 2.4 – Composants fonctionnels

SPA

Le SPA alloue des blocs depuis l'ensemble des périphériques d'une ressource de stockage. Un système peut avoir plusieurs ressources de stockage. Contrairement à un gestionnaire de volume, le SPA ne se présente pas en tant que tel comme un périphérique de blocs logique. Au contraire, il se présente comme une interface pour allouer et libérer des blocs adressés virtuellement. Ce mécanisme de blocs virtuels permet l'ajout et la suppression de périphériques au sein de la ressource de stockage sans interruption de service. Il permet également de faciliter le travail de l'administrateur. Il n'est en effet plus nécessaire de créer des périphériques logiques ou des partitions, il suffit d'indiquer au SPA quels périphériques utiliser.

DMU

La DMU consomme les blocs provenant du SPA et exporte les objets. Ces objets vivent dans le contexte d'un dataset particulier. Un dataset fournit un espace de nommage privé pour les objets contenus dans le dataset. Ces objets sont identifiés par un nombre codé sur 64 bits, contiennent jusqu'à 2^{64} octets de données et peuvent être créés, détruits, lus et écrits. Chaque écriture/création/destruction d'un objet de la DMU est assignée à une transaction

spécifique.

La DMU conserve les données du disque dans un état cohérent à tout instant en traitant tous les blocs en « copie-sur-écriture ». Toutes les données de la ressource de stockage font partie d'une arborescence de blocs indirects, avec les blocs de données placés dans les feuilles. Le bloc situé à la racine de cet arbre est appelé überblock. A chaque fois qu'une partie d'un bloc est modifiée, un nouveau bloc est alloué et l'intégralité du bloc modifié est copiée dans ce nouveau bloc. Comme le bloc indirect doit être modifié afin de sauvegarder le nouvel emplacement du bloc de données, il doit également être copié dans un nouveau bloc. Lorsque la DMU doit modifier le überblock à la racine de l'arbre, il le modifie sur place. Dans le cas où cette modification ne se termine pas correctement, le überblock contient une somme de contrôle qui permet de détecter ce type d'incident. Dans ce cas, la DMU peut lire une copie de sauvegarde du überblock située à un autre emplacement.

Les transactions sont implémentées en écrivant tous les blocs concernés et en modifiant le überblock une seule fois pour la transaction entière. Pour des raisons de performance, la DMU regroupe plusieurs transactions ensemble, ce qui permet au überblock et aux autres blocs indirects d'être modifiés une seule fois pour plusieurs écritures/modifications de blocs de données.

L'interface objet générique de la DMU facilite également l'allocation dynamique des méta-données. Lorsque le système de fichiers a besoin d'un nouvel inode, il alloue simplement un nouvel objet et écrit les données du inode dans l'objet. Lorsque le système de fichiers n'a plus besoin de cet inode, il détruit l'objet le contenant. Le même principe s'applique aux données utilisateur. La DMU alloue et libère les blocs depuis le SPA autant que nécessaire pour stocker les données contenues dans n'importe quel objet.

ZPL

La ZPL permet aux objets de la DMU d'être vus comme un système de fichiers POSIX avec notamment les permissions et autres propriétés. Elle implémente également un ensemble de fonctionnalités aujourd'hui standards pour les systèmes de fichiers du type POSIX avec la fonction mmap(), les permissions ACL (Access Control List)), les attributs étendus …

La ZPL utilise l'interface objet transactionnel de la DMU pour stocker toutes ses données et méta-données. Pour la création des systèmes de fichiers, la ZPL crée simplement quelques objets DMU avec les informations nécessaires (par exemple le inode du répertoire racine). C'est une opération à temps constant, sans aucun lien avec la taille du système de fichiers. Au final, la création d'un

nouveau système de fichiers ne demande pas plus de ressources que la création d'un nouveau répertoire.

L'implémentation des transactions peut perdre les dernières secondes d'écriture si le système plante inopinément. La ZPL inclue un journal appelé ZIL (ZFS Intent Log) pour enregistrer l'activité depuis le dernier groupe de transactions effectivement effectué avec succès. A noter que ce ZIL n'est pas nécessaire pour maintenir la cohérence du système de fichiers, il est uniquement utile pour récupérer les écritures non encore réalisées. Le ZIL peut être stocké soit sur disque soit en mémoire NVRAM (Non Volatile Random Access Memory).

Chapitre 3

Btrfs

Nous allons tout d'abord voir un historique du développement de Btrfs pour ensuite décrire ses principales fonctionnalités. Puis nous détaillerons une partie des composants de son architecture technique. On trouvera également un test effectué sous système Linux en annexe D.2. Et un exemple de solution commerciale en annexe C.2.

3.1 Historique

Btrfs a aujourd'hui 7 ans d'existence [1] :

- 2007 Le B-tree « copie-sur-écriture », structure de donnée fondamentale dans la conception de Btrfs, est proposée par Ohad Rodeh (chercheur chez IBM).
- 2007 Chris Mason, qui travaille à l'époque pour la société SUSE sur un autre système de fichiers appelé ReiserFS, rejoint Oracle et commence le développement de Btrfs.
- 2009 La version 1.0 de Btrfs est intégrée au noyau Linux.
- 2012 Quelques distributions Linux intègrent Btrfs en tant que système de fichiers apte à être utilisé en production.
- 2013 Chris Mason rejoint Facebook où il poursuit le développement de Btrfs.

3.2 Fonctionnalités

Capacités

Btrfs utilise un codage sur 64 bits à chaque fois que possible dans la gestion des différents identifiants stockés au sein de ses arbres B-tree. Btrfs peut ainsi

1. `http://en.wikipedia.org/wiki/Btrfs#History`

gérer jusqu'à 2^{64} inodes (moins quelques centaines qui sont réservés) par volume/sous-volume.

Nombre maximum de fichiers par volume/sous-volume	2^{64}
Taille d'un fichier	2^{64} octets ($\approx 16\,000\,000$ To)
Taille d'un volume	2^{64} octets ($\approx 16\,000\,000$ To)

FIGURE 3.1 – Limites théoriques de Btrfs

Support natif du RAID

Btrfs intègre nativement [10] le support de plusieurs niveaux de RAID :

- RAID-0 qui correspond à une agrégation des périphériques de stockage avec une répartition des données entre ces périphériques. Ce niveau de RAID n'assure aucune sécurité puisqu'il suffit qu'un seul des périphérique tombe en panne pour perdre l'intégralité des données. A noter que Btrfs supporte un mode d'agrégation où les différents disques peuvent être de capacités différentes.
- RAID-1 qui permet d'avoir une copie des données sur un autre disque. Btrfs supporte un mode RAID-1 avec plus de deux disques et qui peuvent être de capacités différentes [2]. Une seule copie des données est toutefois disponible [3]. Par conséquent, les données peuvent survivre à la panne d'un seul disque mais pas au-delà. L'avantage est que la capacité totale de stockage correspond à la moitié de la capacité physique totale des disques.
- RAID-10 qui est une combinaison des deux premiers. Les données sont ainsi distribuées (RAID-0) entre des groupes de disques miroirs (RAID-1).

Le support des niveaux RAID-5 et RAID-6 est à l'heure actuelle au stade expérimental [4]. Le support du RAID-Z n'est à priori pas prévu.

Btrfs permet par ailleurs de configurer des niveaux différents de RAID pour les données d'une part et les méta-données d'autre part. Il faut également noter

2. http://jrs-s.net/2014/02/13/btrfs-raid-awesomeness/
3. https://btrfs.wiki.kernel.org/index.php/UseCases#How_do_I_create_a_RAID1_mirror_in_Btrfs.3F
4. https://btrfs.wiki.kernel.org/index.php/Changelog#By_feature

que les méta-données sont toujours au minimum dupliquées, qu'un système de fichiers soit configuré en RAID ou non.

Intégrité des données

Tous les blocs de données et de méta-données ont une somme de contrôle de taille variable calculée via l'algorithme crc32c. Les sommes de contrôle sont vérifiées à chaque lecture de bloc et sont mises à jour à chaque écriture. Ce mécanisme permet de détecter les éventuelles corruptions de données. Ces sommes de contrôle sont stockées dans un arbre B-tree spécifique.

Le calcul des sommes de contrôle sur les données s'effectuent via plusieurs tâches (« threads ») de manière à partager ce travail entre les processeurs disponibles. Ce qui permet de diminuer fortement l'impact sur les performances [11].

« Auto-réparation »

Les sommes de contrôle décrites précédemment permettent dans certaines circonstances une « auto-réparation » des données [12]. En effet, si le système de fichiers détecte une corruption de données, c'est à dire une absence de correspondance entre la somme de contrôle calculée à la lecture et la somme de contrôle sauvegardée, il peut tenter de lire une autre copie des données. A condition que cette copie existe par ailleurs (par exemple sur un disque miroir) et qu'elle soit elle-même valide. Le système de fichiers peut ensuite réparer automatiquement la version endommagée des données grâce à la copie valide. L'erreur sera toutefois signalée dans le fichier de logs de manière à permettre à l'administrateur d'agir en conséquence (remplacement d'un disque par exemple). Si aucune copie correcte n'est disponible, Btrfs renvoie une erreur à l'application [11].

Contrôle préventif des périphériques de stockage

Il est possible (et recommandé) de lancer périodiquement l'exécution d'un contrôle (« scrubbing ») des disques [13]. Cette opération permet de découvrir les erreurs latentes pendant qu'elles peuvent être encore corrigées. La commande vérifie l'intégrité de toutes les données et les répare automatiquement si besoin et dans la mesure du possible (si une copie « saine » de la donnée est disponible).

Cette procédure s'effectue en arrière-plan avec les systèmes de fichiers en ligne. De manière à diminuer l'impact sur les performances, il est par conséquent préférable de lancer l'exécution de la commande à des périodes où le système est moins sollicité (la nuit ou le week-end par exemple).

Compression

Il est possible de configurer un système de fichiers pour que les blocs de données soient compressés/décompressés à la volée de manière transparente [13]. Deux méthodes de compression sont disponibles : ZLIB et LZO (Lempel-Ziv-Oberhumer). C'est la méthode LZO qui est utilisée par défaut puisqu'elle favorise la vitesse par rapport au ratio de compression [14].

Pour les écritures de petites tailles, il peut arriver que la compression augmente la taille au lieu de la diminuer. Btrfs détecte ce genre de situation et désactive dans ce cas automatiquement la compression. A noter qu'il est possible d'utiliser une option lors du montage du système de fichiers pour forcer dans tous les cas la compression [11].

Quotas et réservation d'espace

L'administrateur peut fixer une limite de taille pour chaque sous-volume [10]. Indépendamment du propriétaire des fichiers. Il est possible de configurer une hiérarchie de quotas. C'est à dire de fixer un quota global pour un sous-volume et tous ses descendants. Et de fixer des quotas différents pour certains de ces descendants.

Il semble qu'actuellement il existe une limitation technique lorsque le quota a atteint son maximum [10]. En effet, du fait de l'utilisation du mécanisme de « copie-sur-écriture », même la suppression d'un fichier recquiert de l'espace disque. Par conséquent, si l'utilisateur parvenu à saturation de son quota souhaite libérer de l'espace en supprimant des fichiers, il ne pourra pas le faire. Car le système de fichiers ne pourra pas procéder à cette suppression du fait qu'il ne peut plus effectuer d'écriture …

Administration du système de fichiers

Deux commandes suffisent à l'administration de Btrfs :

- « mkfs.brfs » destinée à l'initialisation d'un système de fichiers. C'est avec cette commande qu'il est possible d'indiquer le niveau RAID choisi, pour les méta-données et les données.
- « btrfs » dédiée à l'administration d'un système de fichiers existant (création des sous-volumes, instantanés, ajout d'un périphérique, lancement d'un contrôle du système de fichiers, définition des quotas …).

Instantanés et clones

Les instantanés (« snapshots ») sont une copie d'un sous-volume à un instant T. La création d'un instantané est immédiate (quelle que soit la taille du sous-

volume) et il est possible d'en créer un nombre quasi illimité. Aucun espace de stockage supplémentaire n'est utilisé lors de la création. Les blocs sont en effet recopiés uniquement lorsqu'ils sont modifiés.

Un instantané se présente sous la forme d'un sous-répertoire du répertoire qui a fait l'objet de l'instantané [11]. Il est possible de parcourir les fichiers contenus dans ce sous-répertoire tout comme dans un répertoire normal. Par défaut, les instantanés peuvent être modifiés (ils sont en écriture). Mais il est également possible de les créer en lecture seule. Ce qui peut être utile dans le cas de backups par exemple. Il est recommandé de donner des noms explicites aux instantanés afin de les distinguer facilement des répertoires.

Il est possible d'effectuer un retour arrière (un « rollback ») d'un instantané afin de revenir sur toutes les modifications introduites après sa création.

Envoi/Réception

Les instantanés peuvent être envoyés/reçus sous la forme d'un flux de données grâce aux commandes d'envoi (« btrfs send ») et de réception (« btrfs receive ») [10]. Il est possible soit d'envoyer la totalité des données de l'instantané (copie complète) soit d'envoyer uniquement les différences entre deux instantanés (copie incrémentale). Cette fonctionnalité peut être utilisée pour effectuer des sauvegardes ou assurer une réplication des données en local ou à distance.

Seul les instantanés en lecture seule peuvent être envoyés par cette méthode. Ceci n'est pas forcément limitatif puisqu'il est possible de créer un instantané (en lecture seule) d'un autre instantané (en lecture/écriture).

Déduplication

Il existe un projet tiers appelé « bedup » [5] qui permet de rechercher les blocs de données dupliqués dans un système de fichiers afin de les dédupliquer. Ce procédé ne s'effectue donc pas à la volée lors de l'écriture des données. L'avantage de cette méthode est de nécessiter moins de mémoire vive.

Un mécanisme de déduplication lors des écritures est disponible sous forme de patch [6] expérimental. Btrfs pourrait donc à l'avenir intégrer cette fonctionnalité.

Défragmentation

Même si Btrfs utilise des mécanismes efficaces d'allocation de manière à prévenir la fragmentation, ce phénomène se produit au fil du temps et il peut avoir un impact non négligeable sur les performances. Pour corriger ce problème,

5. `https://github.com/g2p/bedup`
6. `http://www.mail-archive.com/linux-btrfs%40vger.kernel.org/msg32862.html`

Btrfs dispose d'un outil de défragmentation en ligne qui peut être exécuté y compris lorsque le système de fichiers est en production [14].

Clone et Périphérique « graine »

Il est possible de créer des clones d'un fichier [12]. Un clone est une copie en lecture/écriture du fichier qui ne nécessite aucun espace disque lors de sa création. Seules les éventuelles modifications apportées au clone seront effectivement stockées.

Un périphérique « graine » (« seed device ») est un périphérique en lecture seule vers lequel plusieurs systèmes de fichiers en lecture/écriture peuvent pointer [12]. Toutes les modifications locales sont stockées au niveau de ces systèmes de fichiers. Si la plus grande partie des données du périphérique « graine » reste inchangée, cela permet un gain de place conséquent. Ce mécanisme est ainsi une forme de clonage.

3.3 Architecture

« Copie-sur-écriture »

Tout comme ZFS, Btrfs utilise le mécanisme de « copie-sur-écriture » [15]. On se reportera à la section 2.3 pour plus de détails sur ce procédé. Là encore, la modification d'une donnée est effectuée de manière complète ou n'est pas effectuée du tout (c'est une opération atomique). Le système de fichiers est donc toujours dans un état cohérent.

A noter que pour réduire l'effet de fragmentation, Btrfs utilise de son côté un B-tree de pré-allocation d'espace de stockage [8].

Composition des systèmes de fichiers

Un système de fichiers Btrfs s'appuie sur plusieurs niveaux de différents éléments (voir figure 3.2) :

- Niveau 1 : Les périphériques physiques qui peuvent être des partitions ou des disques durs.
- Niveau 2 : Un volume qui est un regroupement de périphériques physiques. La capacité totale de stockage sera égale à la somme des capacités de ces périphériques.
- Niveau 3 : Le volume constitue lui-même déjà un système de fichiers. Il est ensuite possible de créer des sous-volumes qui pourront être montés comme des systèmes de fichiers indépendants. L'administrateur peut appliquer des quotas sur ces sous-volumes, effectuer des instantanés …

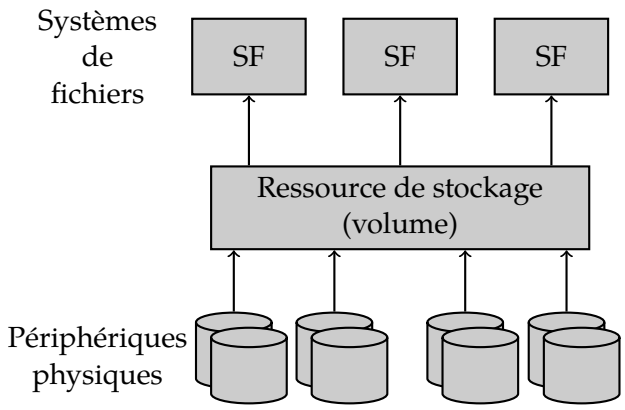

FIGURE 3.2 – Les éléments qui composent un système de fichiers

Composants fonctionnels

B-tree

Un système de fichiers Btrfs utilise plusieurs B-tree [11] [14] [15] :

- Arbre racine : stocke l'emplacement des racines de tous les autres B-tree.
- Arbre des « chunks » (« morceaux ») : conserve la liste des « parties » de périphériques qui sont allouées et pour quels types.
- Arbre des extensions : conserve la liste de toutes les extensions allouées pour le système ainsi que leur compteur de référence.
- Arbre des sommes de contrôle : stocke les sommes de contrôle de toutes les extensions utilisées pour les données dans le système de fichiers.
- Arbre du système de fichiers : stocke les informations sur le système de fichier, sur les fichiers et les répertoires.

Les informations contenues dans ces B-tree y sont organisées par clefs [11] et ce sont les feuilles qui contiennent les données de l'arbre. Chaque clef possède 3 valeurs : un identifiant d'objet, un type et un offset. Pour les fichiers, l'identifiant d'objet correspond au numéro inode. Et chacune des informations spécifique à un inode peut avoir ce même numéro inode comme identifiant d'objet mais avec un type et un offset différent. Ainsi chaque méta-donnée en lien avec un inode peut être stockée à proximité du inode lui-même. Les objets feuilles contiennent la clef, la taille et l'offset de la donnée au sein de la feuille. Ces objets sont rangés dans le sens gauche-droite de la feuille alors que les données sont rangés dans l'ordre inverse. Les données peuvent être d'une taille arbitraire. On peut donc avoir une feuille composée d'un seul objet, le reste de l'espace disponible dans la feuille étant la donnée. C'est ce mécanisme qui permet aux plus petits fichiers d'être stockés directement dans les feuilles du B-tree.

Chapitre 4

Comparatif

4.1 Performances

De récents tests de performance effectués par Phoronix montre un Btrfs globalement meilleur que ZFS pour une même configuration Linux [1]. Il est cela dit assez difficile d'effectuer une comparaison fiable. De nombreux paramètres de configuration peuvent en effet entrer en jeu et les tests ne reflètent pas forcément des cas réels d'applications en production. Par ailleurs, du fait que Btrfs a été dès son origine développé dans le cadre d'un environnement Linux, il se trouve certainement davantage optimisé que ne l'est la version en mode noyau pour Linux de ZFS.

4.2 Supports

Btrfs est développé conjointement par plusieurs acteurs majeurs du secteur informatique (Oracle, Redhat, Fujitsu, Intel, SuSE, aujourd'hui Facebook ...) [13]. Si la version open-source de ZFS a bénéficié à ses débuts du soutien de Sun Microsystems puis pendant une courte période de celui d'Oracle, elle est aujourd'hui soutenue par quelques organismes disposant de moyens nettement moins importants. On peut citer en particulier la société iXsystems [2] pour la version FreeBSD et l'institut Lawrence Livermore National Laboratory pour le port Linux en mode noyau [3].

1. http://www.phoronix.com/scan.php?page=article&item=zfs_linux_062&num=3 et http://www.phoronix.com/scan.php?page=article&item=linux_310_10fs&num=3
2. http://www.ixsystems.com/
3. http://computation.llnl.gov/

4.3 Licences

Btrfs est distribué sous licence GPL (GNU General Public License). Son code source est intégré au dépôt officiel du noyau Linux depuis sa version 2.6.29-rc1. ZFS est quant à lui distribué sous licence CDDL. Si cette licence est sans problème compatible avec par exemple celle utilisée par le projet FreeBSD, elle ne l'est pas avec la licence GPL [4] utilisée pour le noyau Linux. ZFS ne sera donc jamais directement intégré dans le dépôt officiel des sources de Linux. Même si cela n'interdit pas dans la pratique d'utiliser le module noyau de ZFS sous Linux [1], il doit malgré tout être distribué à part.

4.4 Maturité

Le développement de ZFS a débuté en 2001, celui de Btrfs en 2007. ZFS a par conséquent eu davantage de temps pour devenir fiable. Il a aujourd'hui fait ses preuves dans des systèmes en production. Btrfs ne peut pas encore être considéré comme étant totalement stable ni aussi intensivement testé que ZFS. Mais cette situation va indubitablement s'améliorer à court terme. D'autant que quelques distributions Linux proposent dorénavant d'utiliser Btrfs lors de l'installation.

4.5 Fonctionnalités

On peut notamment citer les différences suivantes :

- Absence du RAID-Z sous Btrfs.
- Les systèmes de fichiers Btrfs n'ont pas un format indépendant de l'ordre des octets utilisé par le processeur (« endianness »).
- Btrfs ne propose pas la possibilité de définir un sous-volume qui serait vu par le système d'exploitation comme un périphérique de stockage « brut » (voir section 2.3 page 19).
- La déduplication est nativement intégrée dans ZFS.
- Btrfs utilise un codage 64 bits alors que ZFS utilise un codage 128 bits. Les limites de capacités théoriques sont par conséquent plus élevées sous ZFS.
- Btrfs ne permet pas la délégation de droits d'administration.
- ZFS ne propose pas d'instantané avec une granularité aussi fine que le niveau fichier.
- Btrfs permet de convertir les systèmes de fichiers ext3/ext4 afin de faciliter la migration vers son format.

4. https://www.gnu.org/licenses/license-list.html#CDDL et http://lwn.net/Articles/114839/

Pour d'autres éléments de comparaison technique, on pourra consulter une page pro-ZFS [5] et une page pro-Btrfs [6]. Pour les équivalences entre les commandes ZFS et Btrfs, des pierres de Rosette sont disponibles [7].

4.6 Mise en pratique

Les quelques tests effectués dans le cadre de ce travail (voir annexes D.1 et D.2 m'ont à titre personnel donné le sentiment que ZFS est actuellement plus robuste et plus simple d'accès. La même simulation de corruption d'un disque n'a en effet entraîné aucune perte de données ni aucune interruption de service avec ZFS. Alors qu'avec Btrfs le fichier de test s'est avéré irrécupérable. Il est possible que cette perte de données soit due à mon inexpérience dans l'usage de ce système de fichiers. Il n'empêche que cela montre une certaine fragilité dans Btrfs. Par ailleurs, j'ai trouvé les commandes ZFS plus faciles à appréhender et davantage cohérentes. Avec Btrfs j'ai eu l'impression d'une conception plus « fouillis ». Les messages en sortie des commandes m'ont parus beaucoup moins compréhensibles par exemple. Autres détails qui ne simplifient pas l'utilisation : les sous-volumes sont identifiés par un numéro et non pas un nom, il n'y a pas d'équivalent à un « zpool status » …

5. `http://rudd-o.com/linux-and-free-software/ways-in-which-zfs-is-better-than-btrfs`
6. `http://drdabbles.us/journal/2014/2/15/my-case-for-btrfs-over-zfs.html`
7. `http://rkeene.org/projects/info/wiki/btrfs` et `http://www.seedsofgenius.net/solaris/zfs-vs-btrfs-a-reference`

Chapitre 5

Conclusion

Comme nous l'avons vu, ZFS et Btrfs apportent des améliorations majeures par rapport aux systèmes de fichiers classiques. Leur conception prend en compte la capacité croissante des supports de stockage ainsi que celle exponentielle des volumes de données. L'administration est simplifiée, les configurations sont plus aisément adaptables. La fiabilité de stockage est de par la structure même du système de fichiers nettement meilleure. Tous ces atouts font qu'ils sont théoriquement parfaitement adaptés aux environnements exigeants la manipulation d'énormes quantités de données et/ou une grande souplesse dans leurs configurations. Leur utilisation dans le cadre des big data et du cloud serait donc parfaitement valide.

Ces systèmes de fichiers nouvelle génération sont conçus pour fonctionner sur des environnements disposant au minimum de ressources en mémoire et en puissance processeur que l'on trouve typiquement sur une configuration PC standard actuelle. Il serait par exemple impossible d'utiliser ZFS sur un système avec moins de un gigaoctet de mémoire vive. ZFS et Btrfs ne sont donc pas forcément adaptés à une utilisation sur des systèmes embarqués.

Au niveau des fonctionnalités offertes, ZFS et Btrfs restent proches. Si un choix doit être fait entre ces deux systèmes de fichiers, ce ne sera donc pas obligatoirement sur des critères purement techniques. ZFS a pour l'instant l'avantage d'une maturité plus grande. Il est aussi multiplateforme, ses capacités de stockage sont pour ainsi dire illimitées et il existe des solutions commerciales basées sur ZFS. Btrfs ne peut pas pour le moment se targuer d'être aussi fiable que ZFS[1]. Mais il a l'avantage d'être supporté par des sociétés importantes du monde informatique et d'être officiellement intégré au code source du noyau Linux. Cet écart de maturité encore présent entre Btrfs et ZFS sera très certainement comblé à court ou moyen terme.

1. Les quelques tests décrits en annexe le prouve !

Un scénario possible est que ZFS deviennent un standard sur les serveurs utilisant un système d'exploitation compatible et nécessitant de fortes capacités de stockage. Et qu'il devienne le système de fichiers par défaut pour les systèmes FreeBSD et dérivés. Tandis que Btrfs deviendra celui par défaut sur les distributions Linux. Et sera utilisé sur les serveurs Linux de moyenne gamme.

Malgré leurs apports, ZFS et Btrfs ne répondent pas à toutes les problématiques liées aux systèmes de fichiers. En particulier leur usage se borne à une utilisation locale. Ils n'apportent pour le moment aucune fonctionnalité spécifique quant à l'intégration de ressources de stockage distantes accessibles via une connexion réseau. Dans le cas de ZFS, on pourrait ainsi imaginer à l'avenir la possibilité de créer des périphériques virtuels (vdev) de type distant.

Références

[1] Matt Ahrens and Brian Behlendorf. OpenZFS. http://events.linuxfoundation.org/sites/events/files/slides/OpenZFS%20-%20LinuxCon_0.pdf.

[2] Jeff Bonwick, Matt Ahrens, Val Henson, Mark Maybee, and Mark Shellenbaum. The Zettabyte File System. http://users.soe.ucsc.edu/~scott/courses/Fall04/221/zfs_overview.pdf.

[3] Pawel Jakub Dawidek. Porting the ZFS file system to the FreeBSD operating system. http://2007.asiabsdcon.org/papers/P16-paper.pdf.

[4] Jeff Bonwick and Bill Moore. ZFS - The Last Word In File Systems. http://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf.

[5] Henson, Ahrens, and Bonwick. Automatic Performance Tuning in the Zettabyte File System. http://3c2controller.net/project/truetrue/solaris10/henson-self-tune.pdf.

[6] Documentation/ZfsSend. http://open-zfs.org/wiki/Documentation/ZfsSend.

[7] Manuel FreeBSD - The Z File System (ZFS). http://www.freebsd.org/doc/handbook/filesystems-zfs.html.

[8] Aaron Toponce. ZFS Administration, Part IX- Copy-on-write, 2012. https://pthree.org/2012/12/14/zfs-administration-part-ix-copy-on-write/.

[9] Matthieu Herrb. Introduction à ZFS, 2009. http://homepages.laas.fr/matthieu/talks/capitoul-zfs.pdf.

[10] Jonathan Corbet. Série d'articles sur Btrfs, 2013. http://lwn.net/Articles/576276/.

[11] Josef Bacik. Btrfs - The Swiss Army Knife of Storage, 2012. https://c59951.ssl.cf2.rackcdn.com/4376-bacik_0.pdf.

[12] Margaret Bierman and Lenz Grimmer. How I Got Started with the Btrfs File System for Oracle Linux, 2012. http://www.oracle.com/technetwork/articles/servers-storage-admin/gettingstarted-btrfs-1695246.html.

[13] Chris Mason. The Btrfs Filesystem, 2011. http://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_mason.pdf.

[14] Meaza Taye Kebede. Performance Comparison of Btrfs and Ext4 Filesystems, 2012. https://www.duo.uio.no/bitstream/handle/10852/34150/Thesis.pdf?sequence=1.

[15] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs : The linux b-tree filesystem. *Trans. Storage*, 9(3), 2013. http://doi.acm.org/10.1145/2501620.2501623.

[16] Chris Mason. The Btrfs Filesystem, 2007. https://oss.oracle.com/projects/btrfs/dist/documentation/btrfs-ukuug.pdf.

[17] Heger Dominique A. Workload Dependent Performance Evaluation of the Btrfs and ZFS Filesystems, 2007. http://www.dhtusa.com/media/IOPerfCMG09.pdf.

Annexe A

Arbre de Merkle

Un arbre de Merkle [1] est une structure où les feuilles contiennent la somme de contrôle (le hachage) de blocs de données. Les noeuds parents contiennent la somme de contrôle de leurs enfants. Par exemple, sur la figure A.1, « Hachage 0 » contient la somme de contrôle de la concaténation de « Hachage 0-0 » et de « Hachage 0-1 ». Autrement dit, « Hachage 0 » = somme de contrôle (« Hachage 0-0 » + « Hachage 0-1 »). A noter qu'un arbre de Merkle n'est pas obligatoirement binaire. Un noeud parent peut comporter plus de deux enfants.

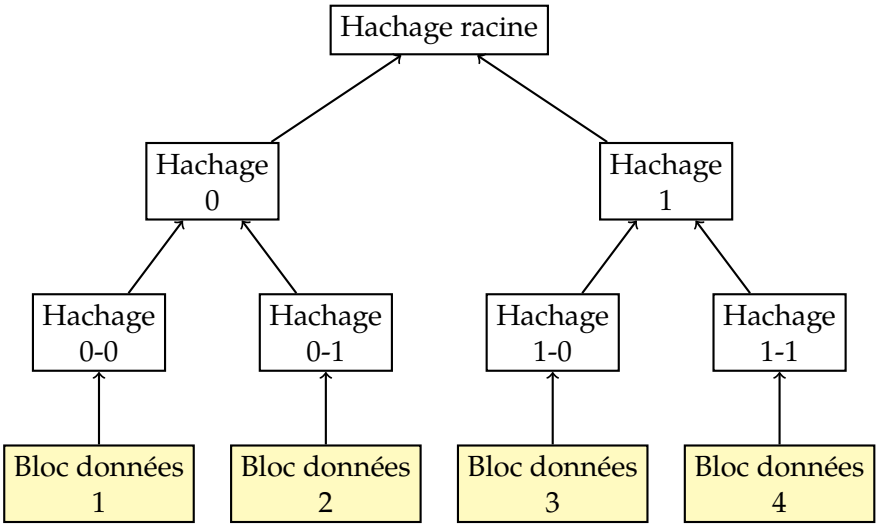

FIGURE A.1 – Structure d'un arbre de Merkle

L'intégrité d'une branche de l'arbre peut être vérifiée même si l'arbre complet n'est pas disponible. Par exemple, sur la figure A.1, la validité du bloc de

1. http://en.wikipedia.org/wiki/Merkle_tree

données numéro 2 peut être vérifiée immédiatement à partir du moment où « Hachage 0-0 » et « Hachage 1 » sont disponibles. Il suffit de calculer la somme de contrôle du bloc de données et de combiner ce résultat avec « Hachage 0-0 » et « Hachage 1 » de manière à pouvoir comparer le résultat final avec la somme de contrôle racine.

Annexe B

« Copie-sur-écriture »

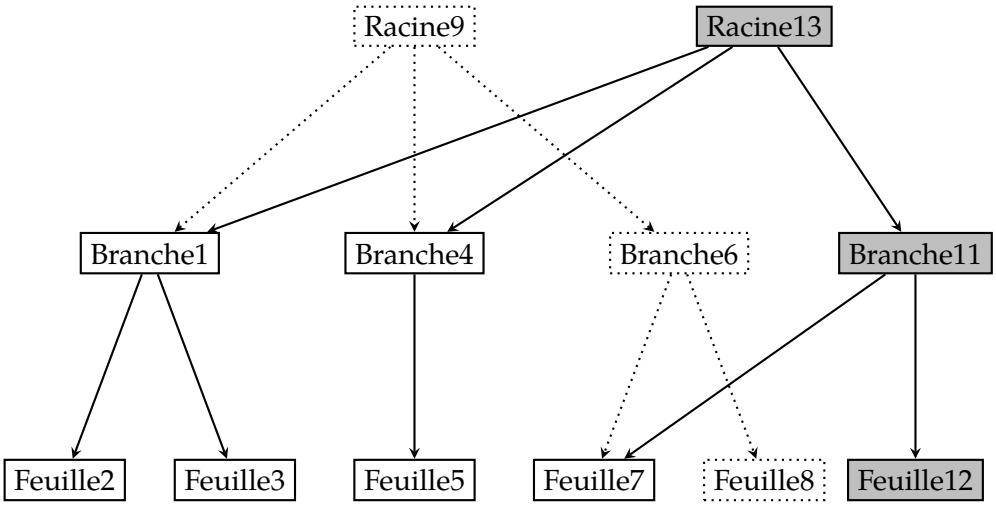

FIGURE B.1 – Opération de mise à jour avec le mécanisme de « copie-sur-écriture »

La figure B.1 illustre un exemple d'une mise à jour avec le mécanisme de copie-sur-écriture [1].

Lorsque les données du bloc de la feuille numéro 8 doivent être modifiées, elles ne le sont pas directement dans ce bloc. Une copie du bloc est modifiée et sauvegardée dans une nouvelle feuille (ici la feuille numéro 12). Les références à la feuille 8 sont ensuite si nécessaire mises à jour de manière récursive jusqu'à la racine de l'arbre.

1. `http://www.bzero.se/ldapd/btree.html`

Les feuilles et les références illustrées en pointillés restent stockées, elles ne sont pas supprimées. Elles deviennent simplement obsolètes.

Annexe C

Solutions commerciales

C.1 ZFS

Le site du projet OpenZFS possède une page avec une liste de sociétés qui proposent des produits basés sur ZFS : `http://open-zfs.org/wiki/Companies`.

EraStor

`https://www.erastor.eu/products/`

iXsystems

`http://www.ixsystems.com/storage/truenas/`
`http://www.ixsystems.com/static/downloads/pdf/truenas_datasheet.pdf`
`http://www.ixsystems.com/static/downloads/pdf/UCSD_CaseStudy_05122014.pdf`
`http://www.ixsystems.com/static/downloads/pdf/DeepDyve_Case_Study-20140514b.pdf`

Nexenta

`https://www.nexenta.com/products`

Oracle

`http://www.oracle.com/us/products/servers-storage/storage/nas/overview/index.html`
`http://www.oracle.com/us/products/servers-storage/storage/nas/zs3-ds-2008069.pdf`

C.2 Btrfs

Le site du projet Btrfs possède une page avec une liste de sociétés qui proposent des produits basés sur ce système de fichiers : `https://btrfs.wiki.kernel.org/index.php/Production_Users`

Netgear

`http://www.netgear.com/images/BTRFS%20on%20ReadyNAS%20OS%206_9May1318-76105.pdf`

Annexe D

Tests pratiques

D.1 ZFS

Configuration

Machine virtuelle sous VirtualBox avec 2 gigaoctets de mémoire vive et 6 disques virtuels de 512 mégaoctets chacun. Système d'exploitation FreeBSD 10.0-RELEASE (révision du 16 janvier 2014) en version 64 bits.

Il peut être nécessaire de charger le module noyau ZFS :

```
# kldload zfs
```

Et d'ajouter la ligne suivante dans le fichier /etc/rc.conf pour monter automatiquement au démarrage les systèmes de fichiers ZFS :

```
zfs_enable="YES"
```

Création du pool

On crée un pool appelé pool1 composé des éléments suivants :

- Un volume raidz en parité 1 composé de 3 disques. On a donc approximativement une capacité utile totale $= (3 - 1) * 512$ Mo $= 1$ Go.
- Un volume dédié pour le ZIL. Ce volume est en mode miroir.

```
# zpool create pool1 raidz /dev/ada1 /dev/ada2 /dev/ada3 \\
  log mirror /dev/ada4 /dev/ada5

# zpool list
```

```
NAME    SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT
pool1  1.48G   251K  1.48G     0%  1.00x  ONLINE  -

# zpool status
  pool: pool1
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        pool1       ONLINE       0     0     0
          raidz1-0  ONLINE       0     0     0
            ada1    ONLINE       0     0     0
            ada2    ONLINE       0     0     0
            ada3    ONLINE       0     0     0
        logs
          mirror-1  ONLINE       0     0     0
            ada4    ONLINE       0     0     0
            ada5    ONLINE       0     0     0

errors: No known data errors

# df -h
Filesystem     Size    Used   Avail Capacity  Mounted on
/dev/ada0p2    1.8G    736M    997M    42%    /
devfs          1.0K    1.0K      0B   100%    /dev
pool1          980M     40K    980M     0%    /pool1
```

On peut constater que la capacité disponible indiqué par la commande « df » est égale à la capacité théorique calculée précédemment.

Création d'un système de fichiers (dataset)

Maintenant que le pool est disponible, on peut créer un premier dataset :

```
# zfs create pool1/fs1
# zfs list
NAME        USED  AVAIL  REFER  MOUNTPOINT
pool1       195K   980M  41.3K  /pool1
pool1/fs1  40.0K   980M  40.0K  /pool1/fs1

# df -h
```

```
Filesystem     Size    Used   Avail Capacity  Mounted on
/dev/ada0p2    1.8G    736M    997M    42%    /
devfs          1.0K    1.0K      0B   100%    /dev
pool1          980M     42K    980M     0%    /pool1
pool1/fs1      980M     40K    980M     0%    /pool1/fs1
```

Ecriture et test de corruption de données

On crée dans le dataset un fichier de contenu aléatoire d'une taille de 128 Mo :

```
# cd /pool1/fs1/
# dd if=/dev/random of=fichier1 bs=1M count=128
128+0 records in
128+0 records out
134217728 bytes transferred in 6.259539 secs (21442110 bytes/sec)
# ls -lh
total 131043
-rw-r--r--  1 root  wheel   128M May 25 00:42 fichier1
```

On calcule une somme de contrôle pour ce fichier :

```
# md5 fichier1
MD5 (fichier1) = 3a162fd727586d02e1a7c52552f5e0c7
```

Pour l'instant, tout va bien :

```
# zpool status
  pool: pool1
 state: ONLINE
  scan: none requested
```

On simule une corruption de données sur un des disques :

```
# cd /
# umount /pool1/fs1
# umount /pool1
# kldunload zfs
# dd if=/dev/random of=/dev/ada2 bs=1M count=64
```

On remonte le système de fichiers :

```
# kldload zfs
# zfs mount pool1
# zfs mount pool1/fs1
# df -h
Filesystem     Size    Used   Avail Capacity  Mounted on
/dev/ada0p2    1.8G    736M    997M    42%    /
devfs          1.0K    1.0K      0B   100%    /dev
pool1          852M     42K    852M     0%    /pool1
pool1/fs1      980M    128M    852M    13%    /pool1/fs1
```

On vérifie le statut du pool :

```
# zpool status
  pool: pool1
 state: ONLINE
status: One or more devices has experienced an unrecoverable error. An
 attempt was made to correct the error. Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the
 errors using 'zpool clear' or replace the device with 'zpool replace'.
see: http://illumos.org/msg/ZFS-8000-9P
scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        pool1       ONLINE       0     0     0
          raidz1-0  ONLINE       0     0     0
            ada1    ONLINE       0     0     0
            ada2    ONLINE       0     0    18
            ada3    ONLINE       0     0     0
        logs
          mirror-1  ONLINE       0     0     0
            ada4    ONLINE       0     0     0
            ada5    ONLINE       0     0     0

errors: No known data errors
```

On lance un scrub pour réparer :

```
# zpool scrub pool1
# zpool status
  pool: pool1
```

```
 state: ONLINE
status: One or more devices has experienced an unrecoverable error.
 An attempt was made to correct the error. Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the
 errors using 'zpool clear' or replace the device with 'zpool replace'.
see: http://illumos.org/msg/ZFS-8000-9P
scan: scrub repaired 44.2M in 0h0m with 0 errors on Sun May 25 01:07:59 2014
config:

        NAME        STATE     READ WRITE CKSUM
        pool1       ONLINE       0     0     0
          raidz1-0  ONLINE       0     0     0
            ada1    ONLINE       0     0     0
            ada2    ONLINE       0     0   778
            ada3    ONLINE       0     0     0
        logs
          mirror-1  ONLINE       0     0     0
            ada4    ONLINE       0     0     0
            ada5    ONLINE       0     0     0

errors: No known data errors
```

On vérifie l'intégrité du fichier, la somme de contrôle est bien identique à celle calculée précédemment :

```
# md5 fichier1
MD5 (fichier1) = 3a162fd727586d02e1a7c52552f5e0c7
```

Instantané

On prend un instantané nommé snapfs1 du dataset pool1/fs1 :

```
# zfs snapshot pool1/fs1@snapfs1
```

On crée ensuite un nouveau fichier dans le dataset :

```
# echo "test" > fichier2
```

On vérifie que l'instantané a bien été créé :

```
# zfs list -t snapshot
NAME                USED  AVAIL  REFER  MOUNTPOINT
pool1/fs1@snapfs1  25.3K      -   128M  -
```

Par défaut, le répertoire .zfs où sont stockés les instantanés est invisible. Nous allons le rendre visible :

```
# zfs set snapdir=visible pool1/fs1

# ls -laig /pool1/fs1/
total 131048
4 drwxr-xr-x  3 root  wheel          4 Jun  9 13:03 .
4 drwxr-xr-x  3 root  wheel          3 May 24 23:11 ..
1 dr-xr-xr-x  4 root  wheel          4 May 24 23:11 .zfs
9 -rw-r--r--  1 root  wheel  134217728 May 25 00:42 fichier1
8 -rw-r--r--  1 root  wheel          5 Jun  9 13:03 fichier2
```

On peut ensuite vérifier que le contenu de l'instantané correspond bien à l'état du dataset à l'instant où l'instantané a été pris (le fichier2 n'est pas présent) :

```
# ls -laig /pool1/fs1/.zfs/snapshot/snapfs1/
total 131045
4 drwxr-xr-x  2 root  wheel          3 May 25 00:40 .
2 dr-xr-xr-x  3 root  wheel          3 Jun  9 13:00 ..
9 -rw-r--r--  1 root  wheel  134217728 May 25 00:42 fichier1
```

Si on modifie fichier1, la modification ne sera pas répercutée sur l'instantané :

```
# echo "toto" > /pool1/fs1/fichier1
# ls -laig /pool1/fs1/
total 139
4 drwxr-xr-x  3 root  wheel  4 Jun  9 13:03 .
4 drwxr-xr-x  3 root  wheel  3 May 24 23:11 ..
1 dr-xr-xr-x  4 root  wheel  4 May 24 23:11 .zfs
9 -rw-r--r--  1 root  wheel  5 Jun  9 13:10 fichier1
8 -rw-r--r--  1 root  wheel  5 Jun  9 13:03 fichier2
# ls -laig /pool1/fs1/.zfs/snapshot/snapfs1/
total 131045
4 drwxr-xr-x  2 root  wheel          3 May 25 00:40 .
2 dr-xr-xr-x  3 root  wheel          3 Jun  9 13:00 ..
9 -rw-r--r--  1 root  wheel  134217728 May 25 00:42 fichier1
```

Et on peut ensuite revenir à l'état correspondant à l'instantané :

```
# zfs rollback pool1/fs1@snapfs1
# ls -laig /pool1/fs1/
total 131047
4 drwxr-xr-x  3 root  wheel          3 May 25 00:40 .
4 drwxr-xr-x  3 root  wheel          3 May 24 23:11 ..
1 dr-xr-xr-x  4 root  wheel          4 May 24 23:11 .zfs
9 -rw-r--r--  1 root  wheel  134217728 May 25 00:42 fichier1
# ls -laig /pool1/fs1/.zfs/snapshot/snapfs1/
total 131045
4 drwxr-xr-x  2 root  wheel          3 May 25 00:40 .
2 dr-xr-xr-x  3 root  wheel          3 Jun  9 13:00 ..
9 -rw-r--r--  1 root  wheel  134217728 May 25 00:42 fichier1
```

Puis on peut détruire l'instantané :

```
# zfs destroy pool1/fs1@snapfs1
# zfs list -t snapshot
no datasets available
# ls -laig /pool1/fs1/.zfs/snapshot/snapfs1/
ls: /pool1/fs1/.zfs/snapshot/snapfs1/: No such file or directory
# ls -laig /pool1/fs1/.zfs/snapshot/
total 0
2 dr-xr-xr-x  2 root  wheel  2 Jun  9 13:13 .
1 dr-xr-xr-x  4 root  wheel  4 May 24 23:11 ..
```

Remplacement d'un disque

On peut remplacer en ligne un disque par un autre :

```
# zpool replace pool1 ada2 /dev/ada6
# zpool status
  pool: pool1
 state: ONLINE
  scan: resilvered 64.2M in 0h0m with 0 errors on Mon Jun  9 13:24:26 2014
config:

        NAME        STATE     READ WRITE CKSUM
        pool1       ONLINE       0     0     0
          raidz1-0  ONLINE       0     0     0
```

```
            ada1    ONLINE       0     0     0
            ada6    ONLINE       0     0     0
            ada3    ONLINE       0     0     0
        logs
          mirror-1  ONLINE       0     0     0
            ada4    ONLINE       0     0     0
            ada5    ONLINE       0     0     0

errors: No known data errors
```

D.2 Btrfs

Configuration

Machine virtuelle sous VirtualBox avec 2 gigaoctets de mémoire vive et 6 disques virtuels de 512 mégaoctets chacun. Système d'exploitation Fedora 20 (noyau Linux 3.11.10) version 64 bits.

Création du volume

On crée un volume appelé volume1 configuré en mode RAID-10 avec 4 périphériques :

```
# mkfs.btrfs -d raid10 -m raid10 -L volume1 /dev/sdb /dev/sdc /dev/sdd /dev/sde
SMALL VOLUME: forcing mixed metadata/data groups

WARNING! - Btrfs v0.20-rc1 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using

Turning ON incompat feature 'mixed-bg': mixed data and metadata block groups
Turning ON incompat feature 'extref': increased hardlink limit per file to 65536
Created a data/metadata chunk of size 8388608
adding device /dev/sdc id 2
adding device /dev/sdd id 3
adding device /dev/sde id 4
fs created label volume1 on /dev/sdb
        nodesize 4096 leafsize 4096 sectorsize 4096 size 2.00GiB
Btrfs v0.20-rc1
```

On peut vérifier que le nouveau volume a été créé et s'ajoute au côté de celui déjà existant pour le système :

```
# btrfs filesystem show -d
Label: 'fedora'  uuid: 01cf0437-3830-4efe-b1a8-cfc7c43aee50
        Total devices 1 FS bytes used 3.73GiB
        devid    1 size 6.71GiB used 4.74GiB path /dev/sda3

Label: 'volume1'  uuid: 5e79ffed-494b-47c2-a638-bfccb57718d7
        Total devices 4 FS bytes used 28.00KiB
        devid    1 size 512.00MiB used 12.00MiB path /dev/sdb
        devid    2 size 512.00MiB used 0.00 path /dev/sdc
        devid    3 size 512.00MiB used 0.00 path /dev/sdd
        devid    4 size 512.00MiB used 0.00 path /dev/sde

Btrfs v0.20-rc1
```

On peut maintenant monter le volume :

```
# mount /dev/sdb volume1
# df -h
Sys. de fichiers Taille Utilise Dispo Uti% Monte sur
/dev/sda3          6,8G    3,9G  2,5G  62% /
devtmpfs           995M       0  995M   0% /dev
tmpfs             1002M     92K 1002M   1% /dev/shm
tmpfs             1002M    728K 1002M   1% /run
tmpfs             1002M       0 1002M   0% /sys/fs/cgroup
tmpfs             1002M     16K 1002M   1% /tmp
/dev/sda1          477M     88M  360M  20% /boot
/dev/sda3          6,8G    3,9G  2,5G  62% /home
/dev/sdb           2,0G     28K  2,0G   1% /home/stephane/volume1
```

On peut ensuite créer dans ce volume un sous-volume nommé sousvol1 :

```
# btrfs subvolume create /home/stephane/volume1/sousvol1
Create subvolume '/home/stephane/volume1/sousvol1'
# ls -laig /home/stephane/volume1/
total 4
256 drwxr-xr-x. 1 root      16  9 juin  14:06 .
257 drwx------. 1 stephane 388  9 juin  14:24 ..
256 drwxr-xr-x. 1 root       0  9 juin  14:31 sousvol1
# btrfs subvolume list /home/stephane/volume1
ID 256 gen 5 top level 5 path sousvol1
```

On peut maintenant monter ce sous-volume :

```
# mkdir /home/stephane/sousvol1
# mount -o subvolid=256 /dev/sdb /home/stephane/sousvol1
# df -h
Sys. de fichiers Taille Utilise Dispo Uti% Monte sur
/dev/sda3          6,8G    3,9G  2,5G  62% /
devtmpfs           995M       0  995M   0% /dev
tmpfs             1002M     92K 1002M   1% /dev/shm
tmpfs             1002M    728K 1002M   1% /run
tmpfs             1002M       0 1002M   0% /sys/fs/cgroup
tmpfs             1002M     16K 1002M   1% /tmp
/dev/sda1          477M     88M  360M  20% /boot
/dev/sda3          6,8G    3,9G  2,5G  62% /home
/dev/sdb           2,0G     32K  2,0G   1% /home/stephane/volume1
/dev/sdb           2,0G     32K  2,0G   1% /home/stephane/sousvol1
```

Ecriture et test de corruption de données

On crée dans le sous-volume un fichier de contenu aléatoire d'une taille de 128 Mo :

```
# cd sousvol1/
# dd if=/dev/urandom of=fichier1 bs=1M count=128
# ls -lh
total 128M
-rw-r--r--. 1 root root 128M  9 juin  15:09 fichier1
```

On calcule une somme de contrôle pour ce fichier :

```
# md5sum fichier1
4f8e7971bff6736511b799d92a1f6945  fichier1
```

On simule une corruption de données sur un des disques :

```
# cd /
# umount /home/stephane/volume1
# umount /home/stephane/sousvol1
# dd if=/dev/urandom of=/dev/sdc bs=1M count=64
```

On tente de remonter les systèmes de fichiers :

```
# mount /dev/sdb /home/stephane/volume1
mount: mauvais type de systeme de fichiers, option erronee,
       superbloc errone sur /dev/sdb, page de code ou programme
       auxiliaire manquant, ou autre erreur
```

Avec Btrfs, c'est impossible ! Afin de pouvoir à nouveau accéder au volume il a été nécessaire de suivre les étapes suivantes.

On consulte les messages de logs pour constater qu'un problème est à priori détecté sur l'arbre des "chunks" :

```
# dmesg | tail
[ 5478.147187] device label volume1 devid 3 transid 16 /dev/sdd
[ 5478.147711] device label volume1 devid 4 transid 16 /dev/sde
[ 5503.535004] device label fedora devid 1 transid 173 /dev/sda3
[ 5503.535397] device label volume1 devid 1 transid 16 /dev/sdb
[ 5503.537238] device label volume1 devid 3 transid 16 /dev/sdd
[ 5503.537811] device label volume1 devid 4 transid 16 /dev/sde
[ 5641.606272] device label volume1 devid 1 transid 16 /dev/sdb
[ 5641.612342] btrfs: disk space caching is enabled
[ 5641.613827] btrfs: failed to read chunk tree on sde
[ 5641.616362] btrfs: open_ctree failed
```

On demande la reconstruction de cet arbre :

```
# btrfs rescue chunk-recover /dev/sdb
ERROR: device scan failed '/dev/sdc' - Invalid argument
We are going to rebuild the chunk tree on disk,
it might destroy the old metadata on the disk,
Are you sure? [y/N]: y
Recover the chunk tree successfully.
```

Ce qui permet de pouvoir à nouveau monter le volume et le sous-volume :

```
# mount /dev/sdb /home/stephane/volume1
# mount -o subvolid=256 /dev/sdb /home/stephane/sousvol1
# df -h
Sys. de fichiers Taille Utilise Dispo Uti% Monte sur
/dev/sda3          6,8G    3,9G  2,5G  62% /
devtmpfs           995M       0  995M   0% /dev
```

```
tmpfs             1002M     92K 1002M   1% /dev/shm
tmpfs             1002M    796K 1001M   1% /run
tmpfs             1002M       0 1002M   0% /sys/fs/cgroup
tmpfs             1002M     16K 1002M   1% /tmp
/dev/sda1          477M     88M  360M  20% /boot
/dev/sda3          6,8G    3,9G  2,5G  62% /home
/dev/sdb           2,0G    129M  1,9G   7% /home/stephane/volume1
/dev/sdb           2,0G    129M  1,9G   7% /home/stephane/sousvol1
```

Le fichier créé précédemment est visible :

```
# ls -laig /home/stephane/sousvol1/
total 131072
256 drwxr-xr-x. 1 root            16  9 juin  15:08 .
257 drwx------. 1 stephane       404  9 juin  15:05 ..
258 -rw-r--r--. 1 root     134217728  9 juin  15:09 fichier1
```

Mais il est illisible … :

```
# md5sum /home/stephane/sousvol1/fichier1
md5sum: /home/stephane/sousvol1/fichier1: Erreur entree/sortie
```

On lance une vérification (« scrubbing ») :

```
# btrfs scrub start /dev/sdb
scrub started on /dev/sdb, fsid 5e79ffed-494b-47c2-a638-bfccb57718d7 (pid=3652)
# btrfs scrub status /dev/sdb
scrub status for 5e79ffed-494b-47c2-a638-bfccb57718d7
        scrub started at Mon Jun  9 15:46:54 2014 and finished after 0 seconds
        total bytes scrubbed: 268.00KiB with 0 errors
```

Mais cela ne change rien :

```
# md5sum /home/stephane/sousvol1/fichier1
md5sum: /home/stephane/sousvol1/fichier1: Erreur entree/sortie
```

On lance l'équivalent d'un « fsck ». Il est au préalable nécessaire de démonter les systèmes de fichiers … :

```
# umount /home/stephane/sousvol1
# umount /home/stephane/volume1
# btrfsck /dev/sdb
Checking filesystem on /dev/sdb
UUID: 5e79ffed-494b-47c2-a638-bfccb57718d7
checking extents
checking free space cache
checking fs roots
checking csums
checking root refs
found 208899 bytes used err is 0
total csum bytes: 131072
total tree bytes: 208896
total fs tree bytes: 12288
total extent tree bytes: 12288
btree space waste bytes: 58961
file data blocks allocated: 134283264
 referenced 134283264
Btrfs v0.20-rc1
# mount /dev/sdb /home/stephane/volume1
# mount -o subvolid=256 /dev/sdb /home/stephane/sousvol1
# ls -laig /home/stephane/sousvol1/
total 131072
256 drwxr-xr-x. 1 root            16  9 juin  15:08 .
257 drwx------. 1 stephane       404  9 juin  15:05 ..
258 -rw-r--r--. 1 root     134217728  9 juin  15:09 fichier1
```

Là encore, sans succès :

```
# md5sum /home/stephane/sousvol1/fichier1
md5sum: /home/stephane/sousvol1/fichier1: Erreur entree/sortie
```

On essaye la commande « restore » de Btrfs :

```
# umount /home/stephane/sousvol1
# umount /home/stephane/volume1
# btrfs restore -l /dev/sdb
 tree key (EXTENT_TREE ROOT_ITEM 0) 4198400 level 1
 tree key (DEV_TREE ROOT_ITEM 0) 4263936 level 0
 tree key (FS_TREE ROOT_ITEM 0) 4202496 level 0
 tree key (CSUM_TREE ROOT_ITEM 0) 4308992 level 1
```

```
 tree key (256 ROOT_ITEM 0) 4210688 level 0
 tree key (DATA_RELOC_TREE ROOT_ITEM 0) 4231168 level 0
# mount /dev/sdb /home/stephane/volume1
# mount -o subvolid=256 /dev/sdb /home/stephane/sousvol1
# ls -laig /home/stephane/sousvol1/
total 131072
256 drwxr-xr-x. 1 root            16  9 juin  15:08 .
257 drwx------. 1 stephane       404  9 juin  15:05 ..
258 -rw-r--r--. 1 root     134217728  9 juin  15:09 fichier1
```

Toujours sans succès :

```
# md5sum /home/stephane/sousvol1/fichier1
md5sum: /home/stephane/sousvol1/fichier1: Erreur entree/sortie
```

Une dernière tentative :

```
# umount /home/stephane/sousvol1
# umount /home/stephane/volume1
# btrfsck --repair /dev/sdb
enabling repair mode
Checking filesystem on /dev/sdb
UUID: 5e79ffed-494b-47c2-a638-bfccb57718d7
checking extents
checking free space cache
cache and super generation don't match, space cache will be invalidated
checking fs roots
checking csums
checking root refs
found 208899 bytes used err is 0
total csum bytes: 131072
total tree bytes: 208896
total fs tree bytes: 12288
total extent tree bytes: 12288
btree space waste bytes: 58961
file data blocks allocated: 134283264
 referenced 134283264
Btrfs v0.20-rc1
# mount -o recovery /dev/sdb /home/stephane/volume1
# mount -o subvolid=256 -o recovery /dev/sdb /home/stephane/sousvol1
# ls -laig /home/stephane/sousvol1/fichier1
```

```
258 -rw-r--r--. 1 root 134217728  9 juin  15:09 \\
    /home/stephane/sousvol1/fichier1
# md5sum /home/stephane/sousvol1/fichier1
md5sum: /home/stephane/sousvol1/fichier1: Erreur entree/sortie
```

On abandonne ce test qui s'avère au final peu concluant pour Btrfs ...