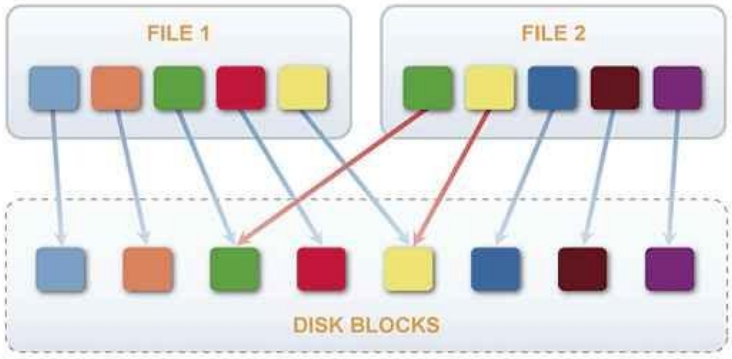
Chapitre 1

Déduplication

La déduplication de données est une technique qui permet de minimiser de l'espace de stockage. Elle consiste à ne pas répliquer les données déja existantes sur le disque. Un fichier est décomposé sous forme de blocs de données car des fichiers peuvent avoir des blocs en commum. Le mécanisme de déduplication crée une table avec les index de tous les blocs de données des fichiers présents sur le disque. La taille des blocs peut varier selon les mécanismes utilisés mais plus les blocs sont petits, plus il y aura de chance qu'un autre bloc soit identique et donc, plus la déduplication sera efficace. En général, cette taille ne dépasse pas les 128ko.

Quand un utilisateur dépose un fichier, le mécanisme crée ses index et regarde s'il n'y a pas des blocs déjà existants. Si des blocs sont similaires alors une simple référence aux blocs déjà existants sera crée. Le schéma ci-dessous montre comment la déduplication fonctionne. Les blocs étant de la même couleur sont considérés identiques.



Il existe deux types de dépuplication : la déduplication à la volée (à la source) et la déduplication hors ligne (à la destination). La déduplication à la volée analyse les fichiers avant de les stocker pour savoir s'ils n'existent pas déjà sur le disque. Cette technique utilise une forte consommation CPU et mémoire. L'autre technique consiste à copier dans un premier temps le fichier sur le disque avant de tester s'il existe déjà. Cela nécessite de prévoir un espace de stockage tampon plus important.

Dans un contexte de serveur de messagerie et de fichiers centralisés, la déduplication de données peut très rapidement économiser de nombreux gigaoctets d'espace disque ainsi que la diminution de la bande passante qui aurait été utilisée pour la sauvegarde. En effet, dans le cas où un même mail de 1Mo est envoyé à cinquante destinataires alors l'économie du disque sera de 50-1 megaoctets (stockage d'un seul mail). La déduplication est faite pour des fichiers tels que des documents bureautiques ou des machines virtuelles qui ont souvent de nombreux blocs en commun.
Le terme inverse de la déduplication est la réhydratation. Elle fait appel à la table des index afin de renvoyer tous les blocs de données référencés pour un fichier demandé.

Certain outils comme LessFS permettent de dédupliquer et de compresser les blocs de données. Cela permet de gagner encore plus d'octets sur le disque mais nécessite une consommation mémoire et CPU plus importante.

Chapitre 2

ZFS

2.1 Introduction

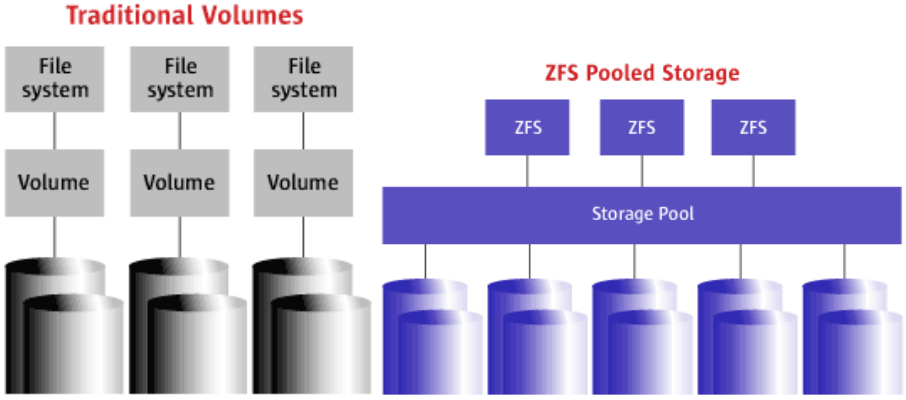
Le système de fichier ZFS (Zettabyte File System) a été conçu par Sun en 2005 et est sous licence CDDL. Il n'était disponible que sous Solaris mais est devenu récemment disponible sous linux. Il est l'un des systèmes de fichiers les plus intéressants du marché. En effet, ZFS intègre de nombreux avantages que d'autres n'ont pas. Voici une liste de ses principaux avantages :

- Pas de limites pratiques (taille des disques, fichiers, ...)
- Garantir la sécurité des données (intégrité, disponibilité)
- Administration simplifiée
- Gestionnaire de volume intégré
- Compression
- Snapshot
- Duplication
- Quotas et réservation d'espace
- Performances élevées
- Indépendant de l'architecture matérielle

ZFS est un système de fichier 128 bits contrairement aux autres systèmes qui sont de 64 bits. Ainsi ses limites sont de 16 milliards de milliards fois plus autant dire qu'il n'a quasi pas de limite. Afin d'optimiser ses performances, ZFS utilise tout l'espace disponible de la RAM pour créer un énorme cache. Ce procédé s'appelle ARC (Adaptive replacement cache). Il peut poser problème aux autres processus qui testent la mémoire inutilisée avant de ce lancer mais cette mémoire est souvent inutilisée. Il peut être partagé via le réseau avec d'autres systèmes de fichiers comme nfs ou samba. Ainsi même depuis des systèmes qui ne le supporte pas, il sera accessible.

2.2 Stockage

ZFS fonctionne avec un pool. C'est un ensemble de périphériques qui fournissent de l'espace pour le stockage et la duplication des données comme le raid logiciel ou matériel. Traditionnellement, les systèmes de fichiers classiques étaient restreint à un périphérique par système. Avec la gestion des volumes, il est possible de créer plusieurs systèmes de fichiers sur un périphérique.



Voici les différentes unités de base de stockage de données :

- Disques : entiers ou juste une partition
- Fichiers dans un autre système de fichiers
- Miroirs : 2 (ou plus) disques, partitions ou fichiers
- Raid-z : plusieurs disques, variante de RAID-5

ZMirror est un miroir classique. Il utilise les mécanismes de checksum pour valider les lectures sur un composant et bascule sur le second s'il détecte une erreur puis corrige le composant défaillant (si possible). Le système Raid-z est similaire au procédé Raid 5. Il utilise les checksums (SHA-256 + fletcher) et repose sur le copy on write : supprime le "write-hole".

2.3 Garantir la sécurité des données (intégrité, disponibilité)

Avec ZFS, toutes les données et métadonnées sont vérifiées selon un algorithme de somme de contrôle. Lorsque qu'un bloc de données endommagé est détecté, ZFS recupère les données correctes à partir d'une autre copie redondante et répare les données endommagées en les remplaçant par celles de la copie.

2.4 Snapshots

Un snapshot (ou instantané) est une copie en lecture seul d'un système de fichier ou d'un volume. ZFS permet donc de pouvoir sauvegarder et restaurer l'image du volume désiré. La création d'un instantané est quasi immédiate. Les instantanés utilisent l'espace de stockage du pool. Une seul opération, dite atomique, permet de créer des instantanés récursifs au système désiré. Ils ne sont pas directement accéssible mais peuvent etre clonés, sauvegardés ou restaurés. D'une manière simple et rapide, un instantané peut etre créé, restauré ou supprimé :

```
#creation du snapshot nommé "nomSnapshot" du système de fichier systèmeZFS au sein du pool "pool"
zfs snapshot pool/systemeZFS@nomSnapshot
#Opération atomique (récursif)
zfs snapshot pool/systemeZFS@nomSnapshot
#Instantané "nomSnapshot" supprimé
zfs destroy pool/systemeZFS@nomSnapshot
#Restauration de l'instantané "nomSnapshot"
zfs rollback pool/systemeZFS@nomSnapshot
#Permet de voir les différences entre les deux instantanés
zfs diff pool/systemeZFS@nomSnapshot pool/systemeZFS@nomSnapshot2
```

2.5 Clones

Un clone est un volume ou un système de fichier accessible en écriture dont le contenu initiale est celui de l'instantané qu'il la créer. En effet, les clones ne sont créer que par des instantanés et une dependance se créer entre les deux. Néanmoins, les clones n'héritent pas les propriétés de leur instantané mais peuvent etre modifié via les commandes zfs set et "zfs get. Les commandes ci-dessous permettent de créer un snapshot et de l'utiliser pour créer un clone.

```
# creation du snapshot
zfs snapshot pool/systemeZFS@nomSnapshot
# creation du système de fichier /home à l'aide du snapshot
zfs clone pool/systemeZFS@nomSnapshot pool/home
```

2.6 Compression

La compression est une option de ZFS. Elle peut etre activer pour chaque systèmes de fichiers et snapshots via l'option compression. Ses valeurs sont soient on, off,lzjb (algorithme tiré de Lempel ziv), gzip et gzip-n. Son taux de compression sera d'environ de 2 suivant le type de données à compresser et l'option choisit. Exemples :

```
# pour le système de fichier
zfs set compresison=on pool/systemeZFS
# pour le snapshot
zfs set compression=on pool/systemeZFS@nomSnapshot
```

2.7 Quotas et reservation d'espace

ZFS permet de limiter la taille de stockage à un système de fichier via l'option "quota" et permet de réserver un espace à un système de fichier via l'option "reservation". Ces propriétés sont très intéréssantes quand il s'agit de limiter de l'espace disques à des utilisateurs. ZFS préconise de créer un système de fichier par utilisateur qui serait monté au sein du même pool. Il est donc possible de créer des cotas par utilisateurs et par groupes via les options zfs userquotas et zfs groupquotas. Il sera possible de lister l'espace utilisé par utilisateur ou par groupe.Voici quelques exemple d'utilisations :

```
#attribution d'un quota de 10 gigaoctets au système de fichier mat
zfs set quota=10G pool/home/mat
#attribution d'un quota de 10 gigaoctets à l'utilisateur mat
zfs set userquota@mat=10G pool/home
#attribution de 100 gigaoctets au groupe etudiant
zfs set groupquota@etudiant=100G pool/etudiant
```

Chapitre 3

Compression

Tout comme la déduplication, la compression est une technique qui permet d'économiser de l'espace de stockage. Chaque fichier est constitué d'une succession de millions de bits 0 ou 1. La compression permet de diminuer le nombre de bits que constitue un fichier en changeant la succession de bits de départ. Suivant l'algorithme de codage utilisé, le taux de compression peut différer. Les algorithmes d'encodage sont plus ou moins efficaces selon le type de fichier compressé.
Il existe deux types de compression : la compression avec perte et sans perte. La compression sans perte signifie qu'après la décompression, le fichier sera identique au fichier compressé. C'est le plus souvent utilisé sur des documents, des fichiers exécutables ou des archives. Ces données étant principalement des caractères texte, ils ne peuvent pas être modifiés. Les formats de documentation tels que txt, doc ou pdf sont donc compressés sans perte. Tant qu'à la compression avec perte, les fichiers décompressés ne seront pas exactement identiques au fichier original mais les informations seront sensiblement les mêmes. Les types de fichiers utilisés par cette compression sont les images, les sons et les vidéos. Cett technique se repose sur la limitation des sens de l'homme comme la vision et l'audition. L'homme ne pourra donc pas identifier les différences entre le fichier original et le fichier après décompressage. Les formats de fichiers jpeg, avi ou mp3 sont donc compressés avec pertes.
Pour chaque technique de compression, il existe plusieurs algorithmes de codage.

3.1 Compression sans perte

Parmi les algorithmes sans perte, il y a les algorithmes tels que Lempei-Ziv ou le codage RLE (Run-Length Encoding) qui consistent à remplacer des suites de bits utilisées plusieurs fois dans un même fichier. D'autres algorithmes comme l'algorithme de codage Huffman détermine les suites de bits et plus une suite est utilisée souvent, plus la suite qui la remplacera sera courte.

3.1.1 L'algorithme Lempel-Ziv

Cet algorithme se divise en deux versions distinctes : LZ77 et LZ78. Ces algorithmes utilisent un dictionnaire où ils référencent les motifs récurrents. A la rencontre d'un motif du dictionnaire, une simple référence au motif est faite (fenêtre glissante). La déduplication utilise globalement le même procédé.

LZ77

La compression LZ77 encode avec un taux de compression inférieur à d'autres algorithmes comme PPM et CM (voir ci-dessous) mais a le double avantage d'être rapide et asymétrique. Cela lui permet d'utiliser un algorithme de décompression différent de celui de la compression. Ainsi, la compression pourra être rapide et la décompression performante. Les variantes LZSS et LZMA sont basées sur la compression LZ77 et supprime quelques inconvénients de celle-ci tels que le taux de compression assez faible (pour LZMA) ou le problème si aucun motif récurrent n'est rencontré (pour LZSS). Ce problème aura pour conséquence d'augmenter la taille du fichier. La compression LZ77 est la base des algorithmes comme Deflate (ZIP, gzip) et donc LZMA (7-zip).

LZ78

La compression LZ78 ou Lempel-Ziv-Welch utilise aussi un dictionnaire mais au lieu de le remplir au fur et à mesure des motifs rencontrés, il crée un dictionnaire initial de tous les symboles possibles. Cela permet d'améliorer la compression car les données du dictionnaire ne devant plus être envoyées au décompresseur,

l'espace utilisé est réduit. L'utilisation de cette technique a été réduite jusque 2003 car elle a été brevetée par UNISYS qui n'avait pas laissé la licence libre.

LZO

Lempel-Ziv-Oberhumer (LZO) est un algorithme de compression en temps réel se basant sur les dictionnaires. Ces avantages sont une compression et décompression rapide. L'un des logiciels l'utilisant est lzop.

3.1.2 L'algorithme RLE

Le run-length encoding (codage par plages) est une technique de compression qui s'applique uniquement sur des documents scannés en noir et blanc tels que des fax. Elle consiste à factoriser les termes d'une même couleur. Ainsi la chaîne : NNNNNNNBBBBNNNNNNNNNNBB (N étant le nombre de points noirs et B étant le nombre de points blancs) sera encodée par RLE en : 7N4B10N2B . Les formats d'images utilisent cette compression en considérant que toutes les lignes de pixels sont jointes pour former une unique séquence de couleur. Les images BMP utilisent cette compression en 1,4 et 8 bits/pixel (noir et blanc, 16 couleurs et 256 couleurs). Le format PCX utilise aussi cette compression pour les images de 8 et 24 bits/pixels. Celles de 24 bits étant découpées en trois parties de 8 bits chacune.

3.1.3 Codage par modélisation de contexte

Prédiction par reconnaissance partielle (PPM)

La prédiction par reconnaissance partielle se base sur une modélisation de contexte pour évaluer la probabilité des différents symboles. Le contexte est un ensemble de symboles déjà rencontrés dans la source de données. Elle utilise les données déjà analysées pour en déduire les données à analyser. Ainsi plus le contexte est long, meilleur sera la prédiction et donc la compression. La prédiction obtenue servira d'entrée à un codage entropique comme le codage Huffman. Elle a l'avantage d'être l'une des plus performantes sur la compression de fichiers texte mais a l'inconvénient de consommer énormément de mémoire si le contexte est très grand. La PPM est un algorithme symétrique contrairement à Lempel-Ziv ce qui signifie qu'il utilise le même pour la compression que pour la décompression. Cela implique un temps d'exécution identique et assez lent.

Pondération de contextes (CM)

La pondération de contextes consiste à utiliser plusieurs prédicteurs (par exemple des PPM) pour obtenir l'estimation la plus fiable possible du symbole à venir. A l'image de la prédiction par reconnaissance partielle, les taux de compressions sont très élevés mais proportionnellemnt aussi lents que la taille du contexte.

3.1.4 L'algorithme de codage Huffman

Cette compression s'apparente à la compression du code morse. Elle consiste donc à coder les séquences fréquentes sur peu de place et ce qui revient rarement sur des séquences plus longues. L'inconvénient avec ce procédé c'est qu'il faut avoir analysé tout le fichier pour créer une table avec les redondances avant de pouvoir le compresser. Il faut donc envoyer la table pour pouvoir le décompresser ce qui peut être problématique quand le fichier à compresser est petit. Le codage Huffman adaptatif corrige ce problème car il remplit au fur et à mesure la table et démarre la compression avec une table de base.
Ce codage est utilisé en seconde compression après que le premier algorithme (tel que LZ77) est mis en évidence la redondance d'information. Ce codage peut être utilisé pour la compression tels que JPEG, MPEG ou MP3 où les données imperceptibles par l'homme sont supprimées mais on parle donc de compression avec pertes.

3.2 Compression avec pertes

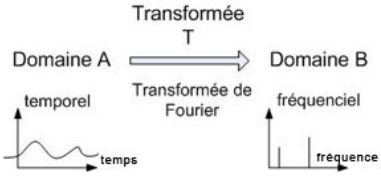
La compression avec pertes s'utilisent donc sur des données perceptibles par l'homme comme les sons, les images ou les vidéos. Elles suppriment les données que l'homme ne perçoit pas ou quasiment pas. Ainsi pour le format JPEG 2000, la compression est de 1 bit/pixels au lieu de 24 bits/pixels. La compression avec pertes est une technique irréversible c'est à dire qu'il ne sera pas possible de retrouver le fichier original. Il existe trois grandes familles de compression avec pertes : la compression par prédiction, par transformation et la compression basée sur les récurrences fractales de motif.

3.2.1 Compression par prédiction

Cet algorithme repose sur un schéma de prédiction et un codage des erreurs entre la prédiction et le signal original. La prédiction consiste à prédire les données à venir en fonction des données analysées. Les erreurs étant souvent de faibles magnitudes, une compression intéressante est possible grâce à la diminution des bits nécessaires à l'opération. Certain formats de codecs de microsoft et d'Apple utilisent cette compression.

3.2.2 Compression par transformation

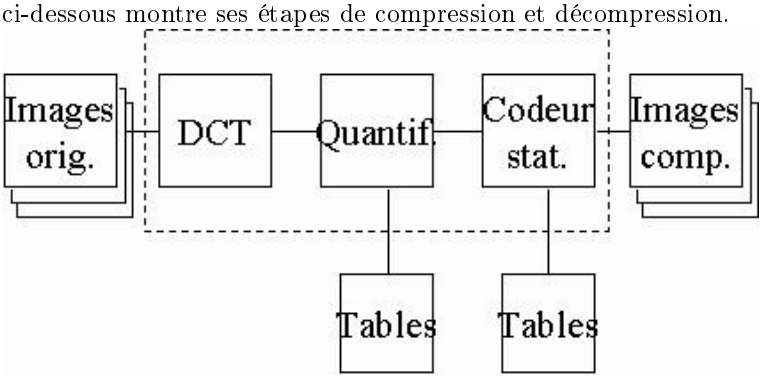
Cet algorithme transforme le signal en atténuant les fréquences non décelables par l'homme. Cette technique transforme donc le signal du domaine temporel au domaine fréquentiel afin de déterminer et de supprimer les pixels redondants. Le schéma ci-dessous montre cette transformation.



La compression JPEG,JPEG 2000 ou encore MPEG utilise cette compression. Cette méthode de compression est la plus répandue au vue de ces performances.

La norme JPEG

La norme JPEG (Joint Photographic Experts Group) est une norme qui définit le format d'enregistrement et l'algorithme de décodage pour une représentation numérique compressée d'une image fixe. La norme JPEG peut être compressée sans perte mais son taux de compression n'est que de 2 au lieu de 3 à 100. Le shéma ci-dessous montre ses étapes de compression et décompression.



Tout d'abord, le format JPEG commence par découper l'image en blocs de données comme beaucoup d'autres formats compressés avec pertes. Puis JPEG transforme les couleurs de chaque bloc à l'aide de la transformée DCT (transformée en cosinus discrète) assimilable à la transformée de Fourier qui transforme le signal temporel en signal fréquentiel (DCT). La valeur des fréquences résultantes détermineront leurs importances dans l'image. Une matrice de ces résultats sera générée. La quantification est l'étape qui permet de réduire considérablement la taille de l'image. En effet, elle utilise la DCT pour atténuer les fréquences non perceptibles par l'homme. La matrice résultante par la quantification sera ensuite codée par un algorithme RLE puis par un algorithme d'Huffman afin d'être compressée. Lors de la quantification et du codage, des tables sont créées et envoyées avec le fichier compressé pour la décompression.
Lors de la compression du format JPEG sans perte, l'étape de la quantification n'est pas présente.

Compression par ondelette

La compression (ou transformée) par ondelette s'utilise globalement comme la norme JPEG mais génère une image de meilleure qualité avec un taux de compression supérieure (de 15 à 50). Contrairement à la transformée DCT, l'image est analysée plus finement et a un résultat plus proche de la perception humaine. Les codeurs JPEG 2000 et SPIHT utilisent tous deux une transformée en ondelettes dans leur schéma de compression. Les domaines d'utilisation de cette compression est l'imagerie médicale, les empreintes digitales ou encore dans le cinéma.

3.2.3 Compression basée sur les récurrences fractales de motif

La compression basée sur les récurrences fractales de motif aussi appelée compression fractale est utilisée pour la compression d'image. Son principe est de détecter les récurrences de motifs et de supprimer les informations

redondantes de l'image. Plusieurs méthodes existent mais la plus connue est la méthode Jacquin. Deux étapes composent cette méthode. Dans un premier temps, deux segmentations sont réalisées : une segmentation de figure source et destination. Ensuite pour chaque figure Source, une figure destination est cherchée afin de créer un couple pour minimiser une erreur. Cette erreur est le résultat de leur soustraction après avoir dimensionné le couple de manière identique. A ce stade, des transformations comme la rotation peuvent être réalisées.

Chapitre 4

Mise en place d'un serveur de fichiers ZFS + NFS

Nous allons mettre en place un serveur de fichiers sur un système de fichiers ZFS pour les nombreux avantages cité ci-dessus. Il ne sera pas possible d'y ajouter des programmes de déduplication comme lessFS et openDedup car lessFS intègre un système de fichier et Opendedup fonctionne avec SDFS. Néanmoins, le système de fichier intègre au même titre que la compression une propriété permettant la déduplication (présente uniquement sur Solaris). Il sera donc question de tester les principales fonctionnalités de ZFS comme la compression, l'intégrité des données (via raid-z), les sauvegardes (via les snapshots) et leurs restaurations. Je créerai ainsi 50 espaces utilisateurs de 1Go chacun où je stockerai divers types de fichiers comme des documents (.odt, .txt, .pdf), des images (jpeg, gif, png) et une base de données mysql. Afin de partager les systèmes de fichiers dédiés aux utilisateurs, ZFS permet d'activer le partage en nfs avec l'option sharenf à on.

4.1 Création des systèmes de fichiers

Nous allons créer un pool de stockage de 60Go en raidz2 qui est équivalent au raid 6. Pour ce faire, on utilisera sept fichiers de 10Go que nous allons créer par la commande ci-dessous :

```
for i in `seq 0 6`;
do dd if=/dev/zero of=./disque_$i bs=1M count=10000;
done;
```

Parmi les septs fichiers utilisés, un sera utilisé pour les redondances d'informations du au raidz2. Ensuite, nous allons créer un pool de stockage nommé poolAsrall à l'aide de ces fichiers où seront installer les systèmes de fichiers. Le pool sera installé dans /media/. Voici la commande :

```
zpool create poolAsrall raidz2 /media/Windows7/disque_{0,1,2,3,4,5,6} -m /media/pool
```

Puis on va créer un système de fichier /home/ au sein de ce pool et les cinquantes systèmes de fichiers dédiés aux utilisateurs.

```
zfs create poolAsrall/home
for i in `seq 0 49`;
do zfs create poolAsrall/home/util_$i;
done;
```

Afin de tester la compression de ses systèmes de fichiers, nous allons activer l'option qui sera récursive à tout ses sous systèmes de fichiers. Nous allons tester l'option on, off, lzjb, gzip, gzip-1, gzip-9 pour tout les espaces utilisateurs.

```
zfs set compression=on poolAsrall/home
```

Puis on va les remplir de 1,42Go de fichiers divers et variés. Pour cela, nous avons copié le contenu de notre dossier Documents qui fait cette taille et qui contient :

- 40 pourcents de fichiers textes (sh, php, html, txt,..)
- 30 pourcents d'images (png, gif, jpeg)
- 15 pourcents de pdf
- 15 pourcents d'autres types de fichiers

Par le script suivant, nous allons remplir ces espaces utilisateurs.

```
for i in 'seq 0 24';
do cp -R  /home/mathieu/Documents poolAsrall/home/util_$i;
done;
```

Concernant la base de données mysql, il faut changer le repertoire de stockage des données dans le fichier /etc/mysql/my.conf puis copier le dossier /var/lib/mysql et le coller dans le dossier où nous voulons stocker les données. L'essentiel est de garder les droits du dossier original.
Dans notre test, nous allons créer un système de fichier bin puis mysql où allons stocker les données de la base.

```
zfs create poolAsrall/bin
zfs create poolAsrall/bin/mysql
cp -a /var/lib/mysql /media/poolAsrall/bin/mysql
cd /media/poolAsrall/bin/mysql && tar -xvf mysql.tgz
```

4.2 Etat du système

Nous allons visualiser l'etat du système mise en place. Pour commencer, nous allons regarder l'etat du pool.

```
zpool status
```

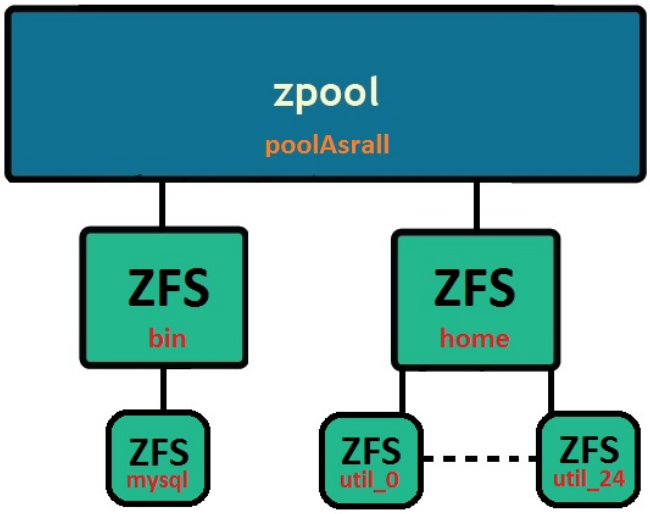
```
pool: poolAsrall
 state: ONLINE
 scrub: none requested
config:

  NAME                          STATE     READ WRITE CKSUM
  poolAsrall                    ONLINE       0     0     0
    raidz2                      ONLINE       0     0     0
      /media/Windows7/disque_0  ONLINE       0     0     0
      /media/Windows7/disque_1  ONLINE       0     0     0
      /media/Windows7/disque_2  ONLINE       0     0     0
      /media/Windows7/disque_3  ONLINE       0     0     0
      /media/Windows7/disque_4  ONLINE       0     0     0
      /media/Windows7/disque_5  ONLINE       0     0     0
      /media/Windows7/disque_6  ONLINE       0     0     0

errors: No known data errors
```

Les disques créés étant situés dans ma partition dédiée à Windows. Par la commande ci-dessous, on peut visualiser l'espace utilisé et libre du pool. Comme aucun quota ou reservation n'a été précisé, les systèmes de fichiers créés à l'interieur du pool utiliserons tous son espace jusqu'à saturation.

```
zpool list
```

```
NAME          SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
poolAsrall     68G  48,8G  19,2G    71%  ONLINE  -
```

4.2.1 Schéma de l'installation



Nous allons visualiser l'espace utilisé par chaque système de fichiers.

```
df -h
```

```
Sys de fichiers        Tail.  Occ. Disp. Occ. Monte sur
/dev/sda7               54G   34G   18G  66% /
none                   1,5G  332K  1,5G   1% /dev
none                   1,5G  116K  1,5G   1% /dev/shm
none                   1,5G  244K  1,5G   1% /var/run
none                   1,5G     0  1,5G   0% /var/lock
none                   1,5G     0  1,5G   0% /lib/init/rw
none                    54G   34G   18G  66% /var/lib/ureadahead/debugfs
/dev/sda2              187G  149G   39G  80% /media/Windows7
/dev/sda5              1,1G  149M  892M  15% /media/b07f0352-66e0-4503-ad13-9c0524b40
poolAsrall              13G   49K   13G   1% /media/poolAsrall
poolAsrall/home         13G  102K   13G   1% /media/poolAsrall/home
poolAsrall/bin          13G   22M   13G   1% /media/poolAsrall/bin
poolAsrall/bin/mysql    13G   22M   13G   1% /media/poolAsrall/bin/mysql
poolAsrall/home/util_0
              15G  1,4G   13G  10% /media/poolAsrall/home/util_0
poolAsrall/home/util_1
              15G  1,4G   13G  10% /media/poolAsrall/home/util_1
poolAsrall/home/util_2
              15G  1,4G   13G  10% /media/poolAsrall/home/util_2
poolAsrall/home/util_3
              15G  1,4G   13G  10% /media/poolAsrall/home/util_3
poolAsrall/home/util_4
              15G  1,5G   13G  10% /media/poolAsrall/home/util_4
poolAsrall/home/util_5
              15G  1,5G   13G  10% /media/poolAsrall/home/util_5
poolAsrall/home/util_6
              15G  1,5G   13G  10% /media/poolAsrall/home/util_6
poolAsrall/home/util_7
              15G  1,5G   13G  10% /media/poolAsrall/home/util_7
poolAsrall/home/util_8
              15G  1,5G   13G  10% /media/poolAsrall/home/util_8
poolAsrall/home/util_9
              15G  1,4G   13G  10% /media/poolAsrall/home/util_9
poolAsrall/home/util_10
              15G  1,4G   13G  10% /media/poolAsrall/home/util_10
poolAsrall/home/util_11
              15G  1,4G   13G  10% /media/poolAsrall/home/util_11
```

```
poolAsrall/home/util_12
              15G  1,4G   13G  10% /media/poolAsrall/home/util_12
poolAsrall/home/util_13
              15G  1,4G   13G  10% /media/poolAsrall/home/util_13
poolAsrall/home/util_14
              15G  1,4G   13G  10% /media/poolAsrall/home/util_14
poolAsrall/home/util_15
              15G  1,4G   13G  10% /media/poolAsrall/home/util_15
poolAsrall/home/util_16
              15G  1,4G   13G  10% /media/poolAsrall/home/util_16
poolAsrall/home/util_17
              15G  1,4G   13G  10% /media/poolAsrall/home/util_17
poolAsrall/home/util_18
              15G  1,4G   13G  10% /media/poolAsrall/home/util_18
poolAsrall/home/util_19
              15G  1,4G   13G  10% /media/poolAsrall/home/util_19
poolAsrall/home/util_20
              15G  1,4G   13G  10% /media/poolAsrall/home/util_20
poolAsrall/home/util_21
              15G  1,4G   13G  10% /media/poolAsrall/home/util_21
poolAsrall/home/util_22
              15G  1,4G   13G  10% /media/poolAsrall/home/util_22
poolAsrall/home/util_23
              15G  1,4G   13G  10% /media/poolAsrall/home/util_23
poolAsrall/home/util_24
              15G  1,4G   13G  10% /media/poolAsrall/home/util_24
/dev/sda3            125G  116G  8,5G  94% /media/Data
```

4.2.2 Compression

Les tests sur les différentes compressions ont été fait de cette manière :
- poolAsrall/home/util_0 - 3 -> compression On : 1,39Go
- poolAsrall/home/util_4 - 8 -> compression Off : 1,42Go
- poolAsrall/home/util_9 - 13 -> compression lzjb : 1,40Go
- poolAsrall/home/util_14 - 18 -> compression gzip : 1,36Go
- poolAsrall/home/util_19 - 21 -> compression gzip-1 : 1,37Go
- poolAsrall/home/util_22 - 24 -> compression gzip-9 : 1,36Go

Nous pouvons constater que la compression à la volée du système de fichiers ZFS permet de gagner environ quatre pourcents grâce à l'algorithme de compression gzip. En effet, contrairement à l'algorithme lzjb qui est tiré de Lempel ziv, gzip compresse grâce à Lempel ziv et Deflate. Le taux de compression peut paraitre assez faible mais ce gain de place sera beaucoup plus élevé sur du contenu textes. De plus, la compression est juste une option parmi les nombreuses que compte ZFS. Malheureusement, la consommation mémoire qui est très élevé pose des problèmes lors du rédémarrage qui parfois, ne réussi à monter tout les systèmes fautes de places mémoires.

4.2.3 Snapshots et restaurations de ZFS

Sous ZFS, il est possible de faire des snapshots sur tous les systèmes de fichiers d'un pool. Ainsi, il sera possible de sauvegarder séparément les systèmes de fichiers récursivement avec leurs fils ou non. Leurs restaurations se feront elles aussi de manière individuelle. Après avoir réaliser le snapshot désirer, il sera judicieux d'envoyer la sauvegarde sur un support sécurisé. Afin de restaurer la sauvegarde réalisée, il faudra supprimer ce système de fichiers puis le restaurer par la commande "zfs receive". Nous allons réaliser un snapshot du système de fichier mysql, le supprimer et le restaurer par les commandes suivantes :

```
# Création du snapshot
zfs snapshot poolAsrall/bin/mysql@snapMysql.snap
# Sauvegarde du snapshot
zfs send poolAsrall/bin/mysql@snapMysql.snap > /home/mathieu/Documents/snapMysql.sna
# Suppression du système de fichier poolAsrall/bin/mysql
zfs destroy poolAsrall/bin/mysql
```

```
# Restauration de la sauvegarde
zfs receive poolAsrall/bin/mysql < /home/mathieu/Documents/snapMysql.snap
# Changement du point de montage car celui par défaut sera son nom
zfs mountpoint=/media/poolAsrall/bin/mysql poolAsrall/bin/mysql
```

Après m'être connecter à une base de données mysql nommé wiki, j'ai créé et sauvegardé un snapshot du système poolAsrall/bin/mysql qui contenait les données de mysql. La sauvegarde ayant pour nom snapMysql.snap, et aura la même taille que le système de fichier sauvegardé. Après avoir supprimé ce système, je l'ai donc restaurer via la sauvegarde. La connexion à cette base se refait parfaitement, ainsi que l'accés à ces données.
Nous allons tester la sauvegarde incrémentale de zfs en insérant une ligne dans la base wiki et en comparant avec l'ancien snapshot :

```
# Ajout d'une ligne dans la base mysql "wiki"
# Création du second snapshot
zfs snapshot poolAsrall/bin/mysql@snapMysql2.snap
# Enregistrement de la difference entre les deux snapshots
zfs send -i poolAsrall/bin/mysql@snapMysql.snap poolAsrall/bin/mysql@snapMysql2.snap
# Taille du snapShot snapMysql.snap soit 42Mo
ls -ali snapMysql.snap
```

```
1591678 -rw-r--r-- 1 root root 44766176 2012-03-11 16:53 ../snapMysql.snap
```

```
# Taille de l'incrément diff.snap soit 1.1Mo
ls -ali diff.snap
```

```
1582651 -rw-r--r-- 1 root root 1137672 2012-03-12 22:23 diff.snap
```

Comme précédemment, nous allons supprimer le système de fichier poolAsrall/bin/mysql avant de pouvoir le restaurer via le snapShot et l'incrément. Avant de pouvoir restaurer un incrément, il faut d'abord réaliser un rollback depuis le snapshot. La commande zfs rollback permet, comme son nom l'indique, de retourner aux données d'un snapshot donné.

```
# Destruction du système de fichier
zfs destroy poolAsrall/bin/mysql
# Restauration du snapshot
zfs receive poolAsrall/bin/mysql < /home/mathieu/Documents/snapMysql.snap
# Retour au snapshot
zfs rollback poolAsrall/bin/mysql@snapMysql.snap
# Insertion de l'increment au système
zfs receive poolAsrall/bin/mysql < diff.snap
```

Concernant les sauvegardes à chauds, la restauration posera des problèmes si une modification de la base est en cours. Afin de corriger ce problème, il faudra utiliser les commandes de mysql lock et unlock avant et après le snapshot.

4.2.4 Rsnapshot

Introduction

Rsnapshot est un logiciel libre qui permet de réaliser des sauvegardes et restaurations. Ces sauvegardes sont incrémentales : si elles sont réalisées tout les jours et qu'elles contiennent des fichiers identiques alors des liens en durs seront fait de cette sauvegarde à la sauvegarde précédente. Le nom des sauvegardes auront le noms de l'option de l'intervalle (hourly.x, daily.x, monthly.x et yearly.x) avec x le numéro de la sauvegarde. La sauveagrde la plus récente portant le numéro 0, et dès qu'une nouvelle sauvegarde sera réalisée alors la sauvegarde 0 deviendra 1 et ainsi de suite.

Installation et configuration

On peut l'installer via le gestionnaire de paquet :

```
sudo apt-get install rsnapshot
```

Rsnapshot n'a qu'un fichier de configuration /etc/rsnapshot.conf où est précisé le répertoire de destination des sauvegardes, les emplacements à sauvegarder, les scripts à éxécuter ou la périodicité des sauvegardes à éxécuter.

Rsnapshot n'a pas prévu un traitement spécifique pour la sauvegarde des bases de données. Ainsi, il utilise les programmes propres aux sauvegardes de bases de données comme pg_dumpall pour postgresql et mysqldump pour mysql. Des scripts pour l'utilisation de ces programmes sont mis à disposition dans un des dossiers sources de rsnapshot. Ils contiennent deux lignes : l'une pour utiliser le programme propre de sauvegarde de la base de données et l'autres pour changer les droits du fichier résultant. Dans mon cas, je vais donc sauvegarder ma base mysql via ce script. Voici les principales configurations du fichier rsnapshot.conf .

```
# Spécification du répertoire de destination des sauvegardes
snapshot_root     /home/mathieu/.snapshots/
#Pour utiliser les sauvegardes via ssh
cmd_ssh           /path/to/ssh
# Spécification de l'intervalles des sauvegardes
interval          hourly  6
interval          daily   7
# Spécification du repertoire à sauvegarder et de l'emplacement dans le repertoire de destination
backup   /media/poolAsrall/home/util_0            localhost/home/
# Spécification du script à executer et de l'emplacement dans le repertoire de destination
backup_script  /usr/local/bin/backup_mysql.sh  localhost/mysql/
```

Sauvegardes

Afin d'automatiser la sauvegarde, il faudra ajouter le lancement de rsnapshot dans le cron. Dans l'exemple suivant, une sauvegarde est réalisée toutes les 4h et tout les jours à 23h.

```
crontab -e
```

```
0 */4 * * * /usr/local/bin/rsnapshot hourly
30 23 * * * /usr/local/bin/rsnapshot daily
```

Pour tester rsnapshot, nous avons éxécuter la commande "/usr/bin/rsnapshot hourly" trois fois de suite. La première fois, la sauvegarde mettra quelques dizaines de secondes à se terminer alors que les sauvegardes suivantes ne mettrons que deux à trois secondes. Pourtant, la taille des dossiers créés sera la même. En effet, les dossiers sauvegardés n'étant pas modifier entre les sauvegardes, juste des liens en durs seront éffectuer de la première sauvegarde aux suivantes.

```
# on test l'espace utilisé par le dossier de sauvegarde .snapshots/
du -hs .snapshots/
```

```
1,5G    .snapshots/
```

La taille du dossier de l'espace utilisateur util_0 de ZFS fait toujours 1,42Go. On peux constater que l'espace du dossier .snapshots/ est lègerement plus volumineux car il contient aussi la sauvegarde des bases mysql. Voyons le contenu de ce dossier et l'espace utilisé pour chacun des sous-dossiers hourly.

```
du -hs .snapshots/*
```

```
1,5G    ./hourly.0
868K    ./hourly.1
868K    ./hourly.2
```

Comme on peux le constater, l'espace de la sauvegarde la plus récente contient les données et les anciennes sauvegardes font justes des liens vers ces fichiers. A chaque modification d'un de ses fichiers, la réécriture du fichier devrait etre effectué. Nous allons copié un fichier iso de 417Mo dans le repertoire util_0, réalisé une sauvegarde, supprimer ce fichier et réaliser une nouvelle sauvegarde.

```
du -ms .snapshots/*
```

```
1481    .snapshots/hourly.0
417     .snapshots/hourly.1
1       .snapshots/hourly.2
1       .snapshots/hourly.3
```

On peux constater que l'or de la modification du repertoire à sauvegarder, Rsnapshot rajoute le fichier entier au snapshot. Puis si le fichier est toujours présent alors il réalisera juste un lien vers le snapshot qui contient ce fichier. Maintenant, nous allons tester si Rsnapshot analyse juste la derniere sauvegarde ou toutes les sauvegardes

pour voir si des fichiers existent déja afin de réaliser les liens. Pour ce faire on va réaliser une sauvegarde, le rajout du même fichier de 417Mo puis une nouvelle sauvegarde. En testant la taille des dossiers, on verra si le fichier apparait une ou deux fois.

```
du -ms .snapshots/*
```

```
1897    .snapshots/hourly.0
1   .snapshots/hourly.1
417 .snapshots/hourly.2
1   .snapshots/hourly.3
1   .snapshots/hourly.4
```

Comme on peut le voir ci-dessus, le fichier entier de 417Mo apparait deux fois dans les snapshots. On peut donc en conclure que Rsnapshot compare juste le dossier à sauvegarder au dernier snapshot réalisé.

Restaurations

Avec rsnapshot, il n'y a pas de façon automatique incorporé pour restaurer des fichiers ou des bases de données. Il faudra donc effectuer les restauration manuellement en se déplaçant dans l'arborescence des sauvegardes et puis, en choisissant le ou les fichiers à restaurer. Afin de partager les sauvegardes avec NFS, il faudra créer un repertoire de sauvegarde propres à chaque utilisateur avec des droits de lecture où ils pourront choisir leurs fichiers sans les modifier,et les rapatrier à l'emplacement désiré. Concernant les bases de données, on pourra les restaurer en utilisant, de la meme manière que pour leurs sauvegardes, les programmes propres aux SGBD.

Chapitre 5

Déduplication avec LessFS

5.1 Introduction

LessFS est un système de fichier permettant la déduplication.Lessfs s'utilise avec le logiciel FUSE (Filesystem in Userspace) et Tokyocabinet (jeu de librairie permettant d'utiliser une base de données). Contrairement au système de fichier normal comme ext3, lessfs stock uniquement les blocks de données qui n'existent pas déja. Quand un block de données existent déja, il fait une simple référence à ce fichier et stock cette référence dans les tables de TokyoCabinet.

5.2 Tests

Afin de tester lessfs sur un cas concret, nous avons choisit de mettre en place ce logiciel sur les données générées par un serveur mail. Ces données étant des fichiers textes, la déduplication devrait atteindre son maximum. La taille des données non dédupliqué du serveur mail est de 20Go. Dans un premier temps, tout les mails sont stockés dans des fichiers différents. Par défaut la taille des blocks de données générer par lessfs est de 128ko (le maximum). Nous allons donc tester avec cette taille de blocks puis avec une taille de blocks de 64ko et pour finir une taille de 4ko (le minimum). Plus la taille des blocks est faible, plus la déduplication devrait etre élevé car il y aura plus de chances de trouver un block identique mais cela générera énormement d'entrées/sorties et donc de consommation mémoire et CPU. Lessfs a un fichier de configuration où on précise le repertoire pour stocker les blocks de données, la base de données tokyocabinet, la taille du cache ou la taille des blocks (voir en annexe). Aussi, lessfs permet de compresser ces blocks de données. Les algorithmes possibles sont qlz (par défaut), qlz15, snappy, Deflate, bzip et lzo. Nous allons tester l'algorithme qlz, bzip, lzo et Deflate. Après avoir copié les données du serveur mail dans le système de fichier lessfs monté sur /media/lessfs.

```
du -hs /media/lessfs/

20G /media/lessfs/
```

On constate que le repertoire fait bien une taille de 20 Go. Regardons maintenant la taille des repertoires stockant les blocks de données et la base de données. Celle ci devrait etre très largement inférieur. J'ai choisit de stocker ces données dans /data/, les blocks de données dans /data/dta/ et la base de données dans /data/mta.

5.2.1 Taille des blocks=4ko et Compression=qlz

```
du -hs /data/*

11G /data/dta
694M /data/mta
```

On peux constater que la déduplication a pas très bien fonctionner. Le nombre de blocks stockés dans la base est très important du aux très grand nombres de blocks a référencé. Le taux de déduplication est inférieur à deux et cela a mis 4h10 pour copier tout le contenu.

5.2.2 Taille des blocks=4ko et Compression=bzip

```
du -hs /data/*
```

```
8,4G    /data/dta
509M    /data/mta
```

La déduplication fonctionne légerement mieux que le test avec la compression qlz mais le temps reste très important. Cela a mis 3h30.

5.2.3 Taille des blocks=4ko et Compression=lzo

```
du -hs /data/*
```

```
9,3G    /data/dta
559M    /data/mta
```

Comme dans le test précédent, la déduplication est faible mais le temps imparti est très largement inférieur. Cela a mis 1h49.

5.2.4 Taille des blocks=4ko et Compression=Deflate

```
du -hs /data/*
```

```
7,8G    /data/dta
559M    /data/mta
```

La déduplication est légerement supérieur et cela mis un temps a peu près identique, c'est à dire 1h51.

5.2.5 Taille des blocks=64ko et Compression=qlz

```
du -hs /data/*
```

```
2,4G    /data/dta
99M /data/mta
```

Avec la taille des blocks de 64ko, la déduplication est très largement supérieur et dans un temps lui aussi largement inférieur, c'est à dire 18 minutes. On constate aussi que la taille de la base de données est lui aussi inférieur car le nombre de blocks a référencé est moins important.

5.2.6 Taille des blocks=64ko et Compression=bzip

```
du -hs /data/*
```

```
2,5G    /data/dta
99M /data/mta
```

Cette fois, la taux de deduplication est quasiment identique avec 8x mais cela a mis plus de deux fois plus de temps, c'est à dire 35 minutes. La compression bzip est donc bien moins rapide que qlz.

5.2.7 Taille des blocks=64ko et Compression=lzo

```
du -hs /data/*
```

```
5,6G    /data/dta
99M /data/mta
```

La déduplication avec lzo est moins performante que celle avec bzip ou qlz mais elle se termine dans un temps largement inférieur, c'est à dire 19 minutes.

5.2.8 Taille des blocks=64ko et Compression=Deflate

```
du -hs /data/*
```

```
1,7G    /data/dta
99M /data/mta
```

Comme dans le test avec les blocks a 4ko, la compression la plus performante est Deflate. Dans ce cas, la compression a très bien fonctionné car elle a mis que 18 minutes avec un taux de 11,7.

5.2.9 Taille des blocks=128ko et Compression=qlz

```
du -hs /data/*
```

```
1,7G    /data/dta
84M     /data/mta
```

La taille de blocks deux fois supérieur, la déduplication et la compression avec qlz ont atteint les performances de la compression Deflate du test précédent. Malgré le taux de déduplication égale, le temps quand a lui, a mis 32 minutes.

5.2.10 Taille des blocks=128ko et Compression=bzip

```
du -hs /data/*
```

```
1,6G    /data/dta
84M     /data/mta
```

Cette fois, la déduplication a bien fonctionné meme mieux que le premier test. La compression avec bzip fonctionne donc mieux que la compression qlz. Le taux de déduplication est de 12,5x en 36 minutes.

5.2.11 Taille des blocks=128ko et Compression=lzo

```
du -hs /data/*
```

```
5,7G    /data/dta
84M     /data/mta
```

Dans ce test le taux de déduplication n'est que 3.5x mais la compression lzo se rattrape dans la rapidité car le temps n'a été que de 19 minutes. Ce temps a été plus rapide que sur le système de fichier ext3 non dédupliqué.

5.2.12 Taille des blocks=128ko et Compression=Deflate

```
du -hs /data/*
```

```
1,3G    /data/dta
84M     /data/mta
```

On constate que la déduplication par Deflate est la plus performante car elle atteind un taux de 15.3x. De plus, la copie a mis un temps de 17 minutes.

5.2.13 Tableau récapitulatif

Taille des blocks / Compression	qlz	bzip	lzo	Deflate
4Ko	1.8x/4h10	2,4x/3h30	2.15x/1h49	2.56x/1h51
64Ko	8.5x/18m	8x/35m	3.5x/19m	11.7x/18m
128Ko	11.7x/32m	12.5x/36m	3.5x/19m	15.3x/17m

La déduplication avec lessFs est donc très performante mais cette performance est très liées aux performances de la machine l'utilisant. En théorie, on pouvait pensé que plus la taille des blocks était faible, plus lessFs pourrait trouvé des blocks similaires. Mais on a pu constater que dans le cas d'un serveur mail et d'une machine assez peu performante, la déduplication était la meilleure quand la taille des blocks était la plus élevé. Un test a été éffectué pour savoir le temps que mettait la copie du meme contenu sur un système de fichier ext3 non dédupliqué et le résultat a été de 23 minutes. LessFs est donc plus rapide de 6 minutes tout en diminiant la taille de plus de quinze fois. Parmi les différents tests, la compression avec Deflate a été la plus élevé car cette compression utilise l'algorithme LZ77 et le codage Huffman qui, sur du contenu qui revient souvent comme sur des mails est très performant. Si on désire sauvegarder ces donneés, c'est possible de "geler" la base tokyocabinet via une connexion telnet afin de réaliser une sauvegarde et une autre option permet de la "dégeler". Après l'avoir tester avec Rsnapshot, la restauration et l'accés aux données se réalise sans problème quand aucune écriture n'est en cours. Dans le cas contraire, la restauration s'effectuera sans problème mais aucun fichier ne sera accéssible. Les données seront donc corrompues.

Testons maintenant une architecture du serveur mail différente. Cette fois, les mails de chaque utilisateur sont stockées dans un fichier. Il y aura donc un fichier par utilisateur mais les données seront identiques.

5.2.14 Taille des blocks=128ko et Compression=Deflate

```
du -hs /data/*
```

```
2,0G    /data/dta
83M /data/mta
```

5.2.15 Taille des blocks=4ko et Compression=Deflate

```
du -hs /data/*
```

```
14G /data/dta
695M    /data/mta
```

Les performances avec lessFs peuvent donc varier suivant le nombre de fichier a traité malgré des données identiques mais misent sous une forme différentes. La déduplication a moins bien fonctionnée. Il a dédupliqué environ deux fois moins mais dans un temps lui aussi deux fois moins pour des blocks d'une taille de 128ko. Pour l'autre test, le temps a mit exactement la meme durée et la déduplication a été aussi deux fois moins performante. LessFS crée un fichier de stat dans le lequel il écrit le ratio de compression qu'il a réalisé sur chaque fichier (voir annexes). Pour le dernier test effectué, on constate que sont taux de compression est d'environ 1,5x pour chaque fichier.

Pour conclure, la déduplication allié à la compression à la volée comme le fait lessFS permet d'économiser énormement d'espace. Le gain d'espace variera selon le nombre et le type de fichiers mais il aura pour la plupart du temps, des meilleurs performances que sur un système non dédupliqué. L'autres avantage est l'accés aux données qui sera transparent pour les utilisateurs contrairement à une archive tar qui faudrait décompresser avant de pouvoir l'exploiter.

Chapitre 6

Annexes

6.1 Rsnapshot

```
#############################################################
# rsnapshot.conf - rsnapshot configuration file             #
#############################################################
#                                                           #
# PLEASE BE AWARE OF THE FOLLOWING RULES:                   #
#                                                           #
# This file requires tabs between elements                  #
#                                                           #
# Directories require a trailing slash:                     #
#   right: /home/                                           #
#   wrong: /home                                            #
#                                                           #
#############################################################

##########################
# CONFIG FILE VERSION    #
##########################

config_version  1.2

##############################
# SNAPSHOT ROOT DIRECTORY    #
##############################

# All snapshots will be stored under this root directory.
#
snapshot_root /home/coach/.snapshots/

# If no_create_root is enabled, rsnapshot will not automatically create the
# snapshot_root directory. This is particularly useful if you are backing
# up to removable media, such as a FireWire or USB drive.
#
#no_create_root 1

#####################################
# EXTERNAL PROGRAM DEPENDENCIES     #
#####################################

# LINUX USERS:   Be sure to uncomment "cmd_cp". This gives you extra features.
# EVERYONE ELSE: Leave "cmd_cp" commented out for compatibility.
#
# See the README file or the man page for more details.
#
```

```
#cmd_cp    /bin/cp

# uncomment this to use the rm program instead of the built-in perl routine.
#
cmd_rm     /bin/rm

# rsync must be enabled for anything to work. This is the only command that
# must be enabled.
#
cmd_rsync /usr/bin/rsync

# Uncomment this to enable remote ssh backups over rsync.
#
#cmd_ssh  /path/to/ssh

# Comment this out to disable syslog support.
#
cmd_logger  /usr/bin/logger

# Uncomment this to specify the path to "du" for disk usage checks.
# If you have an older version of "du", you may also want to check the
# "du_args" parameter below.
#
#cmd_du    /usr/bin/du

# Uncomment this to specify the path to rsnapshot-diff.
#
#cmd_rsnapshot_diff /usr/local/bin/rsnapshot-diff

# Specify the path to a script (and any optional arguments) to run right
# before rsnapshot syncs files
#
#cmd_preexec  /path/to/preexec/script

# Specify the path to a script (and any optional arguments) to run right
# after rsnapshot syncs files
#
#cmd_postexec /path/to/postexec/script

#########################################
#           BACKUP INTERVALS            #
# Must be unique and in ascending order #
# i.e. hourly, daily, weekly, etc.      #
#########################################

interval  hourly  6
interval  daily 7
interval  weekly  4
#interval monthly 3

#################################################
#               GLOBAL OPTIONS                  #
# All are optional, with sensible defaults      #
#################################################

# Verbose level, 1 through 5.
# 1     Quiet           Print fatal errors only
# 2     Default         Print errors and warnings only
# 3     Verbose         Show equivalent shell commands being executed
# 4     Extra Verbose   Show extra verbose information
# 5     Debug mode      Everything
```

```
#
verbose  2

# Same as "verbose" above, but controls the amount of data sent to the
# logfile, if one is being used. The default is 3.
#
loglevel  3

# If you enable this, data will be written to the file you specify. The
# amount of data written is controlled by the "loglevel" parameter.
#
#logfile  /var/log/rsnapshot

# If enabled, rsnapshot will write a lockfile to prevent two instances
# from running simultaneously (and messing up the snapshot_root).
# If you enable this, make sure the lockfile directory is not world
# writable. Otherwise anyone can prevent the program from running.
#
lockfile  /var/run/rsnapshot.pid

# Default rsync args. All rsync commands have at least these options set.
#
#rsync_short_args -a
#rsync_long_args  --delete --numeric-ids --relative --delete-excluded

# ssh has no args passed by default, but you can specify some here.
#
#ssh_args -p 22

# Default arguments for the "du" program (for disk space reporting).
# The GNU version of "du" is preferred. See the man page for more details.
# If your version of "du" doesn't support the -h flag, try -k flag instead.
#
#du_args  -csh

# If this is enabled, rsync won't span filesystem partitions within a
# backup point. This essentially passes the -x option to rsync.
# The default is 0 (off).
#
#one_fs   0

# The include and exclude parameters, if enabled, simply get passed directly
# to rsync. If you have multiple include/exclude patterns, put each one on a
# separate line. Please look up the --include and --exclude options in the
# rsync man page for more details on how to specify file name patterns.
#
#include  ???
#include  ???
#exclude  ???
#exclude  ???

# The include_file and exclude_file parameters, if enabled, simply get
# passed directly to rsync. Please look up the --include-from and
# --exclude-from options in the rsync man page for more details.
#
#include_file /path/to/include/file
#exclude_file /path/to/exclude/file

# If your version of rsync supports --link-dest, consider enable this.
# This is the best way to support special files (FIFOs, etc) cross-platform.
# The default is 0 (off).
```