N°164 OCTOBRE 2013

GNU LINUX MAGAZINE / FRANCE

SÉCURITÉ / PRIVILÈGES

Utilisez le mécanisme de capabilities pour confiner l'exécution des applications et réduire les privilèges des processus p.04

ADMINISTRATION ET DÉVELOPPEMENT SUR SYSTÈMES OPEN SOURCE ET EMBARQUÉS

SQLITE / JAVASCRIPT

Embarquez une base de données et un cache applicatif dans le navigateur côté client p.54

PHP / GOOGLE

Développez des applications web en PHP reposant sur Google App Engine p.76

DEBUG / C

Intégrez une interception de signal avec dump de la pile d'appels à vos codes C p.62

BSD / NAS

Montez votre solution NAS ZFS sous FreeBSD 9 avec interface Web et client lourd p.44

L 19275 - 164 - F: 7,50 € - RD

PHP / FORMATS

PHPExcel : la solution ultime pour échanger des données entre PHP et votre tableur p.70

C / LIBS

Point d'entrée personnalisé : rendez vos bibliothèques exécutables tout comme la glibc p.66

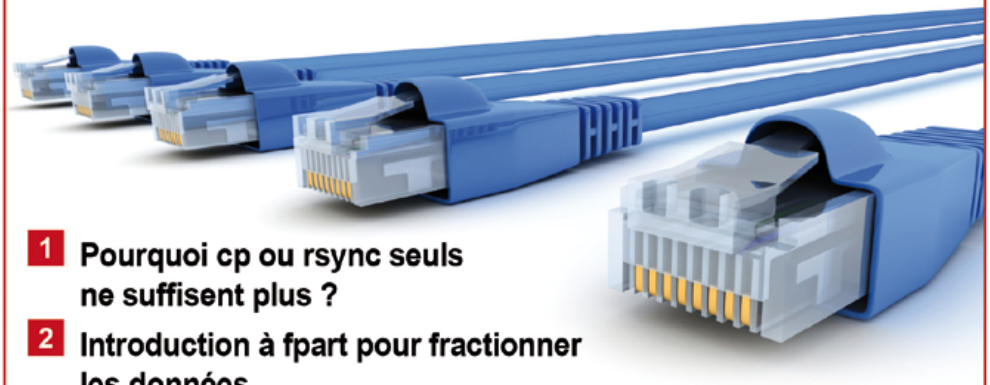
RÉSEAU / BIG DATA

Vous avez des centaines de giga à migrer entre serveurs ?

Retour d'expérience :

BIG DATA

PARALLÉLISEZ ET ACCÉLÉREZ VOS TRANSFERTS DE FICHIERS ! p.14



1. Pourquoi cp ou rsync seuls ne suffisent plus ?
2. Introduction à fpart pour fractionner les données
3. Utilisation de GNU parallel et rsync avec fpart
4. Mise en pratique : migration de données sur le terrain

MAIL / SPAM

Découvrez MIMEdefang, véritable couteau suisse de l'administrateur de messagerie p.30

HUMEUR / PHP

Second degré trollesque : «Pourquoi PHP est-il un meilleur langage que Python ?» p.26

France METRO : 7,50 € – CH : 13 CHF – BEL/PORT.CONT : 8,50 € – DOM : 8 € – CAN : 13,75 $ cad – NCAL/S : 1000 CFP – POL/S : 1100 CFP – POL/A : 1500 CFP – TUNISIE : 16,50 TND – MAR : 95 MAD

EN ROUTE POUR LES NOUVEAUX DOMAINES

Vous pouvez maintenant choisir parmi **plus de 700 nouvelles extensions de domaines** pour créer l'adresse originale et facile à retenir qui vous correspond le mieux, comme par exemple **monrestaurant.paris**, **magalerie.art** ou **legrand.shop**. Pré-réservez aussi de nouvelles extensions pour vos domaines existants. Il sera d'autant plus facile de vous trouver !

Avec environ 20 millions de domaines enregistrés, 1&1 est leader du marché de l'enregistrement de noms de domaine en Europe. Grâce à la **fonction de redirection intégrée**, les domaines enregistrés auprès de 1&1 peuvent rapidement et facilement être redirigés vers n'importe quel site Web, quel que soit votre hébergeur.

Plus d'informations sur **1and1.fr**

NOUVEAU !
PRÉ-RÉSERVEZ SANS FRAIS
ET SANS ENGAGEMENT*

1&1

MEMBRE DE united internet

DOMAINES | MAIL | HÉBERGEMENT | E-COMMERCE | SERVEURS

1and1.fr

* La phase de pré-réservation est gratuite et sans engagement. La disponibilité et le prix des domaines dépendent du registre compétent : 1&1 s'engage à communiquer ces informations dès qu'elles seront disponibles. L'enregistrement du domaine est soumis aux conditions générales de vente de 1&1 et à celles du registre compétent. 1&1 ne peut garantir l'attribution finale des domaines pré-réservés.

SOMMAIRE N°164

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

B.P. 20142 – 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – **Fax** : 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – boutique.ed-diamond.com

Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Secrétaire de rédaction : Véronique Sittler
Réalisation graphique : Jérémy Gall

Responsable publicité : Valérie Fréchard, Tél. : 03 67 10 00 27
v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 34 52 34 01
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, N° **ISSN** : 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,50 €

MIXTE Papier issu de sources responsables FSC® C015136 www.fsc.org

Membre April www.april.org

ÉDITORIAL

Big data... big, big, big data !

On dirait un buzzword, un peu comme le cloud, mais c'est un fait : la masse de données devient énorme, et ce aussi bien en milieu professionnel que sur le plan privé pour son « petit » usage personnel. Il est vrai qu'il existe une tendance naturelle chez tout bon informaticien à ne jamais rien effacer (voire ne jamais rien jeter). On garde tout pour deux principales raisons : la première tient dans la phrase « ça pourra me servir un jour », un classique à la limite du comportement compulsif. La seconde est bien plus pragmatique et justifiée : parce qu'on peut le faire, tout simplement. Un disque de 3 To représente un budget de quelques 100 euros. Attendez une seconde... 3 To ! Mais c'est énorme ! Mais, en vérité, tout a augmenté proportionnellement : les mégapixels des APN, la résolution des vidéos, le détail des informations stockées... Finalement, il s'avère qu'on « rempli » un disque de 3 To presque aussi vite qu'un 120 Go il n'y a pas si longtemps...

Tout change et finalement rien ne change. Sauf, bien sûr, lorsqu'il est question de copies, de sauvegardes, de duplications, ou de manipulations des données... Car même si la puissance de traitement à notre disposition augmente également, elle le fait selon une tout autre logique : la démultiplication. On se retrouve ainsi dans une phase où les moyens nous manquent, car la plupart des outils ne sont pas conçus pour ce type de fonctionnement. Le sujet vedette en couverture du magazine traite du transfert de données, mais nous l'avons vu dans les pages d'un précédent numéro, c'est également vrai pour les outils de compression, par exemple. C'est là le défi qui se présente à nous actuellement, et ce sur n'importe quelle plateforme : réinventer bon nombre de choses qui n'ont jamais été conçues pour le parallélisme et le fonctionnement concurrent.

C'est ça ou convaincre la plupart des utilisateurs et décideurs que non, « plus » ne signifie pas forcément « meilleur ». Mais cet objectif-là semble être bien plus difficile à atteindre, sinon totalement utopique...

Denis Bodor

LES CAPABILITIES SOUS LINUX

par Nicolas Grenèche [Ingénieur de recherche à la DSI de l'université Paris 13 / membre associé au LIPN (Laboratoire d'Informatique Paris Nord)]

La gestion des capabilities est un mécanisme de sécurité du noyau Linux concourant à assurer un confinement d'exécution des applications s'exécutant sur le système en affinant les possibilités d'appliquer le principe du moindre privilège.

1 Confinement d'exécution

Le confinement d'exécution désigne tous les mécanismes réduisant les privilèges d'un processus en activité sur le système. Cette réduction porte sur les différentes actions que peut réaliser le processus : entrées / sorties dans certaines portions du système de fichiers, envoi de signaux, passage d'une interface en mode promiscuous, etc. Le périmètre est assez large, c'est la raison pour laquelle le confinement d'exécution repose sur un ensemble de mécanismes divers. Nous allons en dresser un diaporama (non exhaustif) et essayer de situer ces mécanismes les uns par rapport aux autres en dépilant les couches une par une.

1.1 Séparation de privilèges

Les programmes historiques tels que le serveur de mail Sendmail se présentent sous la forme d'un exécutable unique. C'était quelque chose d'acceptable lorsque les actions réalisées par les programmes étaient assez simples. Au vu de la croissance des fonctionnalités au sein de ces applications, il est nécessaire de les éclater en sous-programmes travaillant de concert pour assurer le service. Ainsi, le serveur Postfix de génération postérieure à Sendmail décompose le service de transfert de mails en plusieurs petits programmes (voir Figure 1).

Dans la figure 1, nous voyons qu'un programme « master » (s'exécutant en root) exécute plein de petits sous-programmes concourant tous à la gestion du transfert de mails. Prenons le cas d'un mail traversant notre service Postfix (s'il est utilisé, par exemple, comme filtre antispam). Le service va recevoir une connexion SMTP sur le port 25 TCP. Le processus « master » va invoquer le processus `smtpd` pour gérer la socket TCP. Une fois que les données sont reçues, « master » invoque « pickup » pour mettre le message en file d'attente de traitement. Ensuite « master » appelle « qmgr » pour traiter le message. Enfin « master » invoque « smtp » qui va réaliser une connexion TCP sur le port 25 du serveur mail de destination. Au niveau développement, ce type d'architecture produit un code court pour chaque fonction, l'audit est facilité. Combiné avec le principe du moindre privilège, la séparation de privilèges augmente considérablement la sécurité des applications.

1.2 Moindre privilège

Une fois les fonctionnalités éclatées en sous-programmes, l'idée du moindre privilège est de donner à chacun d'eux uniquement les droits dont il a besoin. Le principe est simple, la réalisation autrement plus compliquée. Une grande diversité de mécanismes existe pour appliquer ce principe. Dans notre exemple basé sur Postfix, lorsque « master » invoque `smtpd` il commence par accomplir toutes les actions réalisables uniquement par root : ouvrir une socket sur un port privilégié, éventuellement se chrooter, etc. Ensuite il réalise un setuid pour emprunter une identité d'exécution moins privilégiée sur le système.

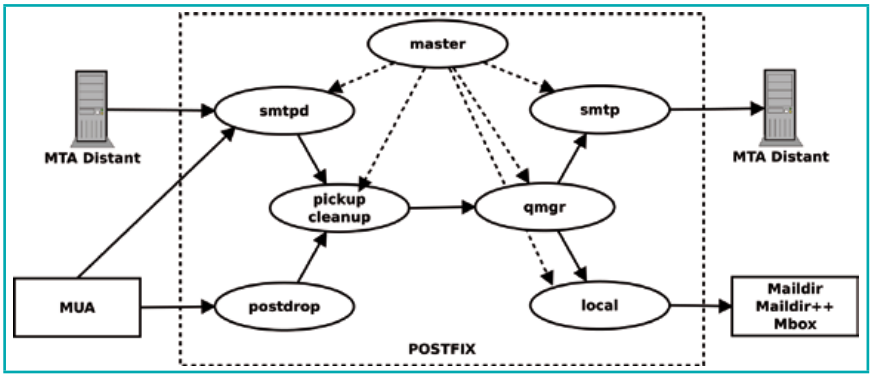


Fig. 1 : Architecture de Postfix

Ce cheminement est un schéma assez classique de l'application du moindre privilège pour une application donnée. Nous allons voir que d'autres mécanismes que le changement d'identité peuvent être utilisés.

2 Le contrôle d'accès sous Linux

Le contrôle d'accès, autrefois très simple (car basé sur un tuple <utilisateur,permission,objet>), s'est complexifié sur les systèmes actuels. Ce modèle a évolué vers des politiques plus complètes disposant de permissions beaucoup plus fines. Certains systèmes de contrôle d'accès peuvent descendre jusqu'à la granularité de l'appel système. La surface de code de ces systèmes et leur nombre croissant ont entraîné la création d'une couche modulaire dédiée à la sécurité du contrôle d'accès : les LSM (Linux Security Modules) **[LSM]**. Les LSM implémentent des hooks dans les fonctions standards du noyau. Toutes les actions résultantes des appels à ces fonctions sont donc interceptées par les modules constituant la chaîne LSM. On notera que les capabilities sont maintenant implémentées comme un module LSM.

2.1 DAC

Le contrôle d'accès sous Linux est, par défaut, de type DAC : Discretionnary Access Control, ou contrôle d'accès discrétionnaire. Cela signifie que les droits sur un objet sont laissés à la discrétion du propriétaire de l'objet. Si vous êtes propriétaire d'un fichier, vous pouvez fixer n'importe quels droits dessus. Cette approche a comme principal problème que la politique de sécurité évolue en fonction des utilisateurs. Elle est donc dans un état non déterminable par l'administrateur du système. De plus, l'implémentation du DAC sous Linux se résume essentiellement à placer des droits sur des fichiers. Enfin, comme tout est fichier, le spectre est finalement assez large. Dans ce modèle, l'utilisateur root outrepasse tout le contrôle d'accès.

2.2 MAC

Le MAC : Mandatory Access Control, ou contrôle d'accès mandataire est également disponible sous Linux. Ce type de contrôle d'accès définit une politique de sécurité pour le système à laquelle même root doit se soumettre. L'implémentation la plus connue du MAC sous Linux est SELinux (Security Enhanced Linux) **[SELinux]** basé sur l'architecture flask **[FLASK]**. Cette implémentation supporte le modèle de sécurité TE (Type Enforcement) **[TE]**. Le TE consiste à labelliser le système. Chaque entité du système se voit attribuer un label. Ensuite, il faut fournir une politique de sécurité définissant les actions autorisées entre les labels. Ce qui n'est pas autorisé explicitement par la politique est interdit. Ici, on voit que la granularité est beaucoup plus fine qu'avec le DAC. Par exemple, tous les processus peuvent écrire dans `/tmp`. Si une politique TE est fournie pour le programme et qu'elle n'autorise pas l'écriture dans `/tmp`, l'opération sera refusée par la couche SELinux et l'écriture échouera. La complexité d'administration de la politique de sécurité est proportionnelle à la finesse de la granularité de la protection.

Note **Modèles de sécurité MAC**

Le modèle de sécurité le plus populaire est le TE, mais il en existe d'autres tels que BellLapudala **[BL]** et son évolution Biba **[Biba]** qui traitent essentiellement de la confidentialité et de l'intégrité grâce à une politique très simple (sur le papier) : « *no write up and no read down* » pour Biba (intégrité) et « *no write down and no read up* » pour BellLapudala (confidentialité).

Dans le modèle MAC, même root est soumis au contrôle d'accès. Le leitmotiv de SELinux est « *confining the omnipotent root* ». Il faut bien comprendre qu'avec ce modèle, root est obligé de suivre une politique de sécurité. C'est un modèle qui impose des contraintes.

2.3 Et sinon ?

Nous avons vu que le DAC se heurte à certaines limites en termes de sécurité et que le MAC était assez compliqué à gérer. Donc niveau moindre privilège simple à mettre en œuvre, n'existe-t-il rien de mieux que le classique `chroot -> setuid` avec des permissions restreintes sur le système de fichiers ? Hé bien si, sous Linux il existe une possibilité assez méconnue consistant à manipuler les capabilities des processus. Prenons par exemple un service tournant sous l'identité root. Dans une configuration courante, il s'exécute avec des droits illimités sur le système. Dans une configuration DAC, il est tout à fait permis au service de changer les droits sur les fichiers du système, de changer d'identité, de créer des utilisateurs, etc. L'idée de SELinux (et de toutes les autres implémentations du MAC que l'on peut trouver dans le noyau Linux) est de le contraindre à suivre une politique. L'approche des capabilities est différente : on juge que le programme s'exécutant avec les droits root est responsable et qu'il abandonne les privilèges dont il n'a pas besoin pour prévenir une escalade de privilèges en cas d'exploitation. Si on va jusqu'au bout du raisonnement, on peut utiliser les capabilities pour faire tourner un programme root avec une identité banalisée en ajoutant à cette identité quelques privilèges normalement réservés à l'administrateur.

3 Les capabilities

Les capabilities sont intégrées aux LSM (Linux Security Modules). Chaque opération privilégiée est associée à une capability. Par exemple, l'appel système setuid ne peut être réalisé que par des processus disposant de la capability `cap_setuid`. Ces capabilities sont réparties en trois sets (ou ensembles) : « effective », « permitted » et « Inheritable ». Lorsqu'un processus a une capability dans les sets « effective » et « permitted » il peut l'utiliser. Par exemple, si un processus dispose de la capability

cap_setuid dans ces deux sets, il peut alors invoquer l'appel système **setuid**. Définition des trois sets :

- Permitted : contient les capabilities permises (mais pas nécessairement utilisées) pour un processus. Une capability doit ABSOLUMENT être présente dans ce set pour pouvoir être utilisée. Les capabilities de ce set peuvent aussi être ajoutées au set « Inheritable » par les processus ne disposant pas de la capability **CAP_SETCAP**. Ceux qui la possèdent ne sont pas soumis à cette restriction ;
- Effective : contient les capabilities autorisées par le noyau pour le processus ;
- Inheritable : contient l'ensemble des capabilities conservées lors d'un appel à execve (recouvrement de l'espace mémoire d'un processus par un autre). Les capabilities de ce set peuvent être passées au set « Permitted » du programme appelé.

Cependant, l'utilisation des capabilities ne se limite pas aux processus. Elles se placent également sur les fichiers exécutables. Chaque exécutable possède également les trois sets sus-nommés. Ils sont évalués avec ceux du processus l'invoquant, via un execve, pour déduire le contenu des sets du processus résultant. Les règles sont les suivantes pour calculer les trois sets du processus résultant :

- Permitted : ces capabilities sont automatiquement ajoutées dans le set « Permitted » du processus résultant. Peu importe le contenu du set « Inheritable » du processus ayant invoqué l'execve ;
- Inheritable : un ET logique est réalisé entre le contenu de ce set et le set « Inheritable » du processus à l'origine du execve. Le résultat est ajouté au set « Permitted » du processus résultant ;
- Effective : en fait, il s'agit plutôt d'un flag que l'on positionne sur chaque capability. Si on ajoute une capability dans ce set, cela veut dire que si elle est également présente dans le set « Permitted » calculé pour le processus résultant alors elle se retrouve également dans son set « Effective ».

Voila pour les règles de fonctionnement des capabilities, nous allons voir comment cela fonctionne côté utilisateur et côté noyau.

3.1 Niveau utilisateur

Au niveau utilisateur (enfin plutôt administrateur système), on va surtout manipuler les capabilities sur les fichiers. C'est-à-dire que l'on va donner quelques privilèges à un exécutable afin qu'il ne tourne pas avec l'identité root. De manière classique pour que les utilisateurs puissent exécuter un programme nécessitant des droits root, on peut lui positionner le bit SUID. Ce bit particulier exécute le programme avec les droits du possesseur de l'exécutable et non ceux du lanceur. Un exemple typique est **ping** :

```
root@capng:/testcap# ls -al /bin/ping
-rwsr-xr-x 1 root root 36136 avril 12  2011 /bin/ping
```

Le SUID est identifié par le bit « s » mis en évidence dans la sortie de **ls**. Supprimons-le :

```
root@capng:/testcap# chmod u-s /bin/ping
root@capng:/testcap# ls -al /bin/ping
-rwxr-xr-x 1 root root 36136 avril 12  2011 /bin/ping
```

Testons en tant que l'utilisateur « toto » :

```
toto@capng:/testcap$ ping www.google.fr
ping: icmp open socket: Operation not permitted
```

Nous pouvons ajouter la capacité **cap_net_raw** dans les sets « permitted » et « effective » (pe) du fichier exécutable **ping**. Cet ajout se fait avec la commande **setcap** (**getcap** pour afficher les capabilities d'un exécutable) :

```
root@capng:/testcap# setcap cap_net_raw=pe /bin/ping
root@capng:/testcap# getcap /bin/ping
/bin/ping = cap_net_raw+ep
```

Testons à nouveau en tant que l'utilisateur « toto » :

```
toto@capng:~$ ping -c 1 www.google.fr
PING www.google.fr (173.194.40.152) 56(84) bytes of data.
64 bytes from par10s10-in-f24.1e100.net (173.194.40.152): icmp_req=1
ttl=63 time=1.83 ms

--- www.google.fr ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.838/1.838/1.838/0.000 ms
```

Ça marche ! Le souci est que la capability **cap_net_raw** de **ping** est appliquée pour tous les utilisateurs. D'après les règles de transitions vues en introduction, nous avons placé la capability dans le set « Permitted » et nous l'avons activée dans le set « Effective ». Ainsi, quelles que soient les capabilities du processus appelant, **ping** s'exécutera avec cette capability dans son set « Effective ».

On pourrait vouloir restreindre cela à certains utilisateurs. C'est tout à fait possible en plaçant des capabilities à activer pour un utilisateur donné dans le set « Inheritable » du processus de son shell. Voyons ce qu'il en est par défaut pour « toto » :

```
toto@capng:~$ /sbin/getpcaps $$
Capabilities for `2630': =
```

La commande **getpcaps** suivie du PID affiche les capabilities du processus attaché au PID. La variable $$ renvoie le PID du shell en cours d'exécution. L'astuce pour donner des capabilities au shell d'un utilisateur est d'utiliser un (Pluggable Authentication Modules) PAM, le **pam_cap**. Les PAM s'exécutant en root, il est tout à fait possible de placer des capabilities sur le shell qui est invoqué à la connexion. Il faut juste ajouter une entrée au fichier **/etc/security/capability.conf** :

```
cap_net_raw             toto
none  *
```

GREAT PLACE TO WORK® Best Workplaces 2013 France

UNIVERSUM TOP 100 IDEAL EMPLOYER 2013 STUDENT SURVEY

IMAGINER ET CONCEVOIR UN MONDE SANS CONTRAINTE

Technologies de pointe, challenges complexes en systèmes embarqués, ambitions internationales. Avec Parrot, faites le pari de l'innovation dans l'univers des périphériques sans fil.

LES TECHNOLOGIES JAVA, ANDROID ET LINUX N'ONT PAS DE SECRETS POUR VOUS ?
BOUSCULEZ TOUS LES CODES SUR **WWW.RECRUTE.PARROT.COM**

UN MONDE D'INNOVATIONS AU CŒUR DE PARIS

Parrot®

OMNIMOON

Ici, on ajoute la capability **cap_net_raw** au set « Inheritable » du shell de l'utilisateur **toto**. Vérifions au login :

```
toto@capng:~$ /sbin/getpcaps $$
Capabilities for `2651': = cap_net_raw+i
```

Modifions les capabilities positionnées sur **ping** :

```
root@capng:/testcap# setcap cap_net_raw=ie /bin/ping
root@capng:/testcap# getcap /bin/ping
/bin/ping = cap_net_raw+ei
```

Testons avec « toto » :

```
toto@capng:~$ ping -c 1 www.google.fr
PING www.google.fr (173.194.40.152) 56(84) bytes of data.
64 bytes from par10s10-in-f24.1e100.net (173.194.40.152): icmp_req=1
ttl=63 time=2.28 ms

--- www.google.fr ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.280/2.280/2.280/0.000 ms
```

Et avec « titi » :

```
titi@capng:~$ /sbin/getpcaps $$
Capabilities for `2674': =
titi@capng:~$ ping -c 1 www.google.fr
ping: icmp open socket: Operation not permitted
```

L'utilisateur toto a la capability **cap_net_raw** dans le set « Inheritable » de son shell (qui est le processus réalisant le fork -> execve du ping). On l'a également positionné sur le set « Inheritable » du fichier exécutable. Le résultat du ET logique l'ajoute donc dans le set « Permitted » du processus résultant. Or, sur l'exécutable, le flag du set « Effective » est activé pour cette capability. Elle est donc ajoutée dans le set « Effective » du processus résultant. Victoire !

Avec un utilitaire comme **ping**, cela peut paraître bien accessoire. Il est utilisé depuis des années, n'a pas une surface de code monstrueuse et n'offre pas beaucoup de biais à l'utilisateur. Par contre, avec un programme comme **nmap** qui nécessite des droits root pour faire de l'OS fingerprinting (reconnaissance de système d'exploitation grâce aux spécificités de la pile réseau cible) c'est plus intéressant. Nmap dispose de tout un système de plugins avec un langage particulier. La surface de code est donc beaucoup plus conséquente **[nmap]**.

3.2 Niveau noyau

Pour le noyau, les capabilities sont attachées à deux objets : les tâches (tasks, processus) et les fichiers. Un set de capabilities est représenté de la manière suivante (**linux/capability.h**) :

```
typedef struct kernel_cap_struct {
        __u32 cap[_KERNEL_CAPABILITY_U32S];
} kernel_cap_t;
```

Avec la variable **_KERNEL_CAPABILITY_U32S** définie à 1 ou 2 selon que le nombre de capabilities disponible est inférieur ou supérieur à 32. Pour les processus, cette structure est utilisée dans la structure **cred** (**linux/cred.h**). Cette structure représente le contexte de sécurité d'un processus en cours d'exécution.

```
struct cred {
  [...]
  kernel_cap_t    cap_Inheritable; /* caps our children can inherit */
  kernel_cap_t    cap_permitted;  /* caps we're permitted */
  kernel_cap_t    cap_effective;  /* caps we can actually use */
  kernel_cap_t    cap_bset;       /* capability bounding set */
  [...]
}
```

On retrouve bien les trois vecteurs « Inheritable », « Permitted » et « Effective » plus un quatrième « Bounding » dont nous reparlerons un peu plus tard. Cette structure est bien rattachée à la structure **task_struct** qui représente une tâche (processus) en cours d'exécution (**sched.h**) :

```
struct task_struct {
  [...]
  const struct cred __rcu *real_cred; /* objective and real subjective task
                                       * credentials (COW) */
  const struct cred __rcu *cred;  /* effective (overridable) subjective task
                                   * credentials (COW) */
  [...]
}
```

Nous avons deux pointeurs sur la structure **cred**. Le premier (**real_cred**) est utilisé par le processus lorsqu'un autre processus essaye d'interagir avec lui. Le second (**cred**), lorsque le processus essaye d'agir sur un autre. Sauf cas particulier, les deux pointeurs pointent sur la même instance de la structure **cred**. En fait, c'est l'histoire du RUID (Real UID) et de l'EUID (Effective UID).

Pour les fichiers, la commande **setcap** fait appel au **syscall setxattr** :

```
root@capng:/testcap~# strace -e setxattr setcap cap_net_raw=ei /bin/ping
setxattr("/bin/ping", "security.capability", "\x01\x00\x00\x02\x00\x00\
x00\x00\x00 \x00\x00\x00\x00\x00\x00\x00\x00", 20, 0) = 0
```

Les attributs étendus (xattr, eXtended ATTRibutes) sont des métadonnées non interprétées par le système de fichiers. Ici, ils vont être utilisés pour stocker les structures **kernel_cap_t** associées aux fichiers. Le prototype de la fonction est (**fs/xattr.c**) :

```
setxattr(struct dentry *d, const char __user *name, const void __user
*value, size_t size, int flags)
```

En argument, on a un objet **dentry**. La structure **dentry** correspond à un cache au niveau de VFS (Virtual FileSystem).

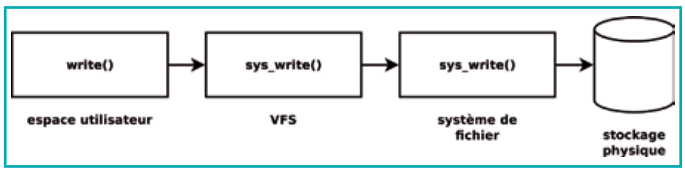


Fig. 2 : VFS

C'est une structure qui correspond à une projection en mémoire vive des différents composants du chemin sur lequel on travaille. Ici, il s'agit du fichier **/bin/ping**. Nous avons donc trois composants **dentry** : **/** et **bin** qui sont des répertoires et **ping** qui est un fichier exécutable. L'intérêt de cette structure est d'avoir un cache en mémoire pour toutes les opérations telles que l'évaluation d'un chemin qui sont coûteuses en opérations d'entrées-sorties sur le disque. Le second argument donne le type de métadonnées. La troisième valeur représente les capabilities en mode brut. Cet appel système utilise la fonction **vfs_setxattr** :

```
vfs_setxattr(struct dentry *dentry, const char *name, const void
*value, size_t size, int flags)
```

Qui appelle la fonction **security_inode_setxattr** (**security/security.c**) :

```
security_inode_setxattr(struct dentry *dentry, const char *name,
const void *value, size_t size, int flags)
```

Nous sommes arrivés au hook LSM qui va juste tester si l'opération d'ajout de la capbility est permise. Si oui, la fonction **vfs_setxattr** continue son exécution et entre dans la fonction **__vfs_setxattr_noperm**. Cette fonction résout l'inode par rapport à la **dentry** pour ajouter les capabilities au fichier. Les capabilities sont donc liées à l'inode.

Lorsque le système va exécuter un fichier via un appel à **execve**, c'est la fonction **do_common_execve** qui est appelée (**fs/exec.c**) :

```
do_execve_common(const char *filename, struct user_arg_ptr argv,
struct user_arg_ptr envp)
```

Cette fonction appelle **prepare_binprm** qui traduit les métadonnées associées à l'inode en structure **linux_binprm** (**linux/binfmts.h**). Cette structure contient la structure **cred** qui elle-même contient les capabilities.

4 Programmation

Nous allons mettre en application toutes les notions vues sur les capabilities en utilisant la librairie **libcap-ng** et ses bindings python pour réaliser un script **father** qui va exécuter un script **child**. L'idée est que **father** tourne avec l'identité root et se contente de lancer **child** avec le minimum de privilèges sur le système. Ici, **child** va juste ouvrir un raw socket, éventuellement pour envoyer une requête ICMP afin de vérifier la disponibilité d'un hôte.

4.1 Hello world

Commençons par un programme simple. Il se contente de retourner l'identité sous laquelle il s'exécute, puis les capabilities dont dispose le processus. Pour la suite des exemples, nous prendrons comme convention que « all » dans les affichages renvoie au jeu complet de capabilities, c'est-à-dire **chown**, **dac_override**, **dac_read_search**, **fowner**, **fsetid**, **kill**, **setgid**, **setuid**, **setpcap**, **linux_immutable**, **net_bind_service**, **net_broadcast**, **net_admin**, **net_raw**, **ipc_lock**, **ipc_owner**, **sys_module**, **sys_rawio**, **sys_chroot**, **sys_ptrace**, **sys_psacct**, **sys_admin**, **sys_boot**, **sys_nice**, **sys_resource**, **sys_time**, **sys_tty_config**, **mknod**, **lease**, **audit_write**, **audit_control**, **setfcap**, **mac_override**, **mac_admin**, **syslog**, **wake_alarm**.

```
root@capng:/testcap# python hello.py
UID : 0 EUID 0

== Effective ==
all

== Permitted ==
all

== Inheritable ==
none

== Bounding set ==
all
```

On constate que le programme s'exécute sous l'identité root, qu'il dispose de toutes les capabilities (**permitted = all**), qu'il les utilise (**effective = all**) et qu'après un **execve**, le processus résultant n'héritera de rien (**Inheritable = none**) dans son set « Permitted ». Par contre, il pourrait ajouter toutes les capabilities au set Inheritable (**bounding = all**).

Note **Set « Bounging »**

Un ET logique est réalisé en le set « Bounding » du processus appelant l'**execve** et le set « Permitted » du fichier exécutable. Le résultat est ajouté au set « Permitted » du processus résultant. Cela ajoute une contrainte sur les capabilities situées dans le set « Permitted » de l'exécutable. Par extension, c'est une limite sur les capabilities utilisables par un exécutable après son invocation via **execve**.

Voyons le code de cet exemple :

```
#!/usr/bin/python

import os
import capng

def printcap ():
  print "== Effective =="
  capng.capng_print_caps_text(capng.CAPNG_PRINT_STDOUT, capng.CAPNG_EFFECTIVE)
  print "\n"
  print "== Permitted =="
  capng.capng_print_caps_text(capng.CAPNG_PRINT_STDOUT, capng.CAPNG_PERMITTED)
  print "\n"
  print "== Inheritable =="
  capng.capng_print_caps_text(capng.CAPNG_PRINT_STDOUT, capng.CAPNG_INHERITABLE)
  print "\n"
  print "== Bounding set =="
  capng.capng_print_caps_text(capng.CAPNG_PRINT_STDOUT, capng.CAPNG_BOUNDING_SET)

def main():
```

```
  print "UID : " + str(os.getuid()) + " EUID " + str(os.geteuid())
  print "\n"
  proc_caps = capng.capng_get_caps_process()
  printcap()
  print "\n"

if __name__ == "__main__" :
  main()
```

Les primitives pour manipuler les capabilities du processus sont situées dans un module **capng**. Une fois ce module chargé, il faut utiliser la fonction **capng_get_caps_process()** pour récupérer les quatre sets de capabilities (effective, permitted, inheritable et bounding). Enfin, la fonction **printcap()** produit un affichage pour chacun des sets en s'appuyant sur la fonction **capng_print_caps_text()**. Relançons-le en utilisateur non privilégié :

```
toto@capng:/testcap$ python hello.py
UID : 1000 EUID 1000

== Effective ==
none

== Permitted ==
none

== Inheritable ==
none

== Bounding set ==
all
```

Maintenant, à part dans le set bounding, il n'y a plus aucune capabilities sur le processus. Dans la suite, nous ne réécrirons pas le code de la fonction printcap, nous ne donnerons que les portions de code pertinentes.

4.2 Le programme « child »

Le programme **child** affiche l'identité sous laquelle il fonctionne, ses quatre sets de capabilities et tente d'ouvrir une socket en mode raw. Voyons le code source :

```
#!/usr/bin/python

import os
import capng
import socket
import sys

def openrawsock ():
  try:
    s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW)
  except socket.error as msg:
    s = None
    print msg
  if s is not None:
    print "Socket OK"
    s.close()
  else:
    print "Socket Error"
    sys.exit(1)

def main():
  print "UID : " + str(os.getuid()) + " EUID " + str(os.geteuid())
  print "\n"
  proc_caps = capng.capng_get_caps_process()
  printcap()
  print "\n"
  openrawsock()
  sys.exit(0)

if __name__ == "__main__" :
  main()
```

Finalement, c'est le même programme que précédemment, on lui ajoute juste une fonction **openrawsock** qui ouvre une socket en mode raw. Exécutons-le en root :

```
root@capng:/testcap# python child.py
UID : 0 EUID 0

== Effective ==
all

== Permitted ==
all

== Inheritable ==
none

== Bounding set ==
all

Socket OK
```

La socket s'est bien ouverte. Lançons-le maintenant en utilisateur lambda :

```
toto@capng:/testcap$ python child.py
UID : 1000 EUID 1000

== Effective ==
none

== Permitted ==
none

== Inheritable ==
none

== Bounding set ==
 all

[Errno 1] Operation not permitted
Socket Error
```

Il n'y a aucune capabilities de positionnée et le programme ne fonctionne pas. En effet, ici le processus python n'a pas la capability **net_raw** nécessaire.

4.3 Le programme « father » version 1.0

Le premier programme master va abandonner ses capabilities non nécessaires, changer d'identité et faire un **execve** du programme **child**. Ce programme pourrait simuler un wrapper lançant un programme, normalement root, sous une identité banalisée tout en conservant la capability dont il a besoin. Pour voir comment abandonner des capabilities, nous allons travailler sur une version modifiée du programme « hello » :

CHOISISSEZ ~~ENTRE~~
LA TECHNOLOGIE & LE SERVICE

À PARTIR DE
19,99€ HT/MOIS

Processeur Intel dernière génération*

* à partir du G460

RAM jusqu'à 96 Go DDR3

Disque jusqu'à 8 To SATA ou 4 x 240 Go SSD
Disques additionnels jusqu'à 3 To

TECHNOLOGIE DE POINTE | DATACENTER PROPRIÉTAIRE | SERVICE CLIENT ★★★★

Support Technique
Nos équipes sont disponibles sur site en 24x7.

Gérez votre serveur sur une interface intuitive
Reboot, Reverse DNS, Rescue MX, Backup, RAZ...

Disponible sur votre Espace Client

RENDEZ-VOUS SUR : express.ikoula.com/serveur-dedie

01 84 01 02 50
sales@ikoula.com

ikoula Hosting Services

NOM DE DOMAINE | MESSAGERIE | HÉBERGEMENT | CERTIFICAT SSL | CLOUD | SERVEUR DÉDIÉ

```
def main():
        unpriv_user="toto"
        print "UID : " + str(os.getuid()) + " EUID " + str(os.geteuid())
        proc_caps = capng.capng_get_caps_process()
        printcap()
        print "\n"
        print "|== Father drop & setuid ==|"
        print "\n"
        capng.capng_clear(capng.CAPNG_SELECT_BOTH)
        capng.capng_update(capng.CAPNG_ADD, capng.CAPNG_INHERITABLE, capng.
CAP_NET_RAW)
        if capng.capng_change_id(pwd.getpwnam(unpriv_user).pw_uid, pwd.
getpwnam(unpriv_user).pw_gid, capng.CAPNG_DROP_SUPP_GRP | capng.CAPNG_CLEAR_
BOUNDING) < 0:
                print "loupe"
        capng.capng_apply(capng.CAPNG_SELECT_BOTH)
        print "\n"
        printcap()
        os.execve('/testcap/child.py',['/testcap/child.py'], os.environ)
```

Voyons le résultat :

```
root@capng:/testcap# python father.py
UID : 0 EUID 0
== Effective ==
all

== Permitted ==
all

== Inheritable ==
none

== Bounding set ==
all

|== Father drop & setuid ==|

== Effective ==
none
== Permitted ==
none

== Inheritable ==
net_raw

== Bounding set ==

|== Child execution == |
UID : 1000 EUID 1000

== Effective ==
none

== Permitted ==
none

== Inheritable ==
net_raw

== Bounding set ==

net_raw[Errno 1] Operation not permitted
Socket Error
```

Le programme commence par vider ses sets de capabilities avec la fonction **capng.clear()**. Ensuite, il positionne la capacité net_raw dans son set « Inheritable » avec la fonction **capng_update()**. À ce moment-là, les capabilities ne sont pas encore actives pour le processus. Il faut utiliser la fonction **capng_apply()** pour les appliquer au processus en cours d'exécution. On notera que cette fonction nécessite d'avoir la capability **setpcap** dans le set « effective ». Le programme change ensuite d'identité. Pour réaliser cette action, il ne faut pas utiliser un setuid classique qui va vider les sets et supprimer toutes les capabilities actives (et donc nous empêcher de réaliser le **capng_apply()**). Il faut utiliser la fonction **capng_change_id()** qui réalise le changement d'identité sans nettoyer les capabilities du processus. Cependant, le programme **child** échoue. On constate que la capability **net_raw** n'est pas dans les sets « effective » et « permitted ». Il faut donc placer ces capabilities sur l'exécutable :

```
root@capng:/testcap# setcap cap_net_raw=ie /usr/bin/python2.7
```

Recommençons :

```
root@capng:/testcap# python father.py
[...]

|== Child execution ==|
UID : 1000 EUID 1000

== Effective ==
net_raw

== Permitted ==
net_raw

== Inheritable ==
net_raw

== Bounding set ==
none
Socket OK
```

On notera que le setcap a été fait sur l'exécutable python et non pas sur le script. Dans les langages de scripts, c'est l'interpréteur qui est l'exécutable, et non pas le script.

4.4 Le programme « father » version 2.0

Dans cette version, nous allons simuler un processus qui lance des programmes en série. L'idée va être d'implémenter un petit programme qui va créer un processus fils exécutant **child**, attendre la fin de ce processus fils et poursuivre son exécution (figure 3).

Évidemment, il ne faut pas toucher au programme **child**. En fait, cela va se résumer à un enchaînement **fork() -> setuid() -> execve()**. L'enchaînement **setuid() -> execve()** a déjà été traité précédemment. Voyons le code source :

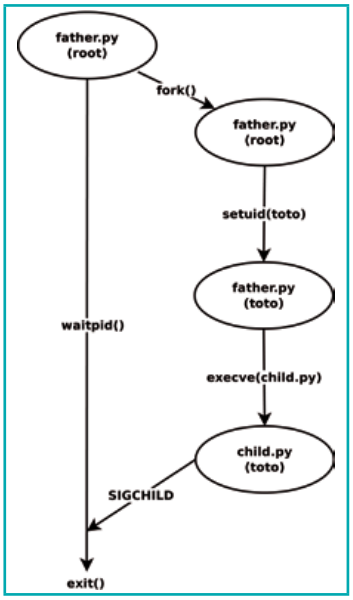


Fig. 3 : father 2.0

```
def main():
        pid = os.fork()
        if pid:
                os.waitpid(pid,0)
                print "Father : keep on working ..."

        else:
                proc_caps = capng.capng_get_caps_process()
                print "Child : Display initial capabilities"
                printcap()
                print "\n"
                unpriv_user="toto"
                print "Child : Dropping capabilities"
                capng.capng_clear(capng.CAPNG_SELECT_BOTH)
                capng.capng_update(capng.CAPNG_ADD, capng.CAPNG_INHERITABLE,
capng.CAP_NET_RAW)
                print "Child : SetUID -> toto"
                if capng.capng_change_id(pwd.getpwnam(unpriv_user).pw_uid, pwd.
getpwnam(unpriv_user).pw_gid, capng.CAPNG_DROP_SUPP_GRP | capng.CAPNG_CLEAR_
BOUNDING) < 0:
                        print "SetUID failed"
                print "Child : Applying capabilities sets"
                capng.capng_apply(capng.CAPNG_SELECT_CAPS)
                print "|-- Child printcap() before execve --|"
                printcap()
                print "\n"
                os.execve('/testcap/child.py',['/testcap/child.py'], os.environ)
                sys.exit(0)
```

Le programme appelle la fonction **`fork()`** qui crée un processus fils. Ce processus fils hérite des capabilities du père (ici, de toutes les capacités). Dans le branchement conditionnel du fils, on retrouve un code quasi similaire à celui du 5.3 : il s'agit d'un enchaînement **`setuid() -> execve()`**. Exécutons-le :

```
root@capng:/testcap# python father.py
Child : Display initial capabilities
== Effective ==
all
== Permitted ==
all
== Inheritable ==
none
== Bounding set ==
all
Child : Dropping capabilities
Child : SetUID -> toto
Child : Applying capabilities sets
|-- Child printcap() before execve --|
== Effective ==
none
== Permitted ==
none
== Inheritable ==
net_raw
== Bounding set ==
none
|== Child execution ==|
UID : 1000 EUID 1000
== Effective ==
net_raw
== Permitted ==
net_raw
== Inheritable ==
net_raw
== Bounding set ==
none
Socket OK
Father : keep on working ...
```

La différence est que le père attend le retour du fils pour continuer son exécution. Le fils s'exécute avec le minimum de privilèges. L'intérêt est que l'on pourrait enchaîner un second test au niveau du père en créant un autre processus qui abandonnerait son identité root pour nobody pour réaliser une connexion TCP sur le port 25 d'une machine afin de tester la disponibilité d'un service SMTP.

Conclusion

Les capabilities sont très intéressantes en matière de confinement d'exécution. Ce mécanisme basé sur la responsabilité de l'application d'abandonner tel ou tel droit doit cependant être manipulé avec précaution. Le premier problème est que les capabilities sur un fichier sont liées à l'inode. S'il y a un remplacement du fichier, elles sont perdues. Le second est que la charge au niveau de l'administration système est alourdie. Il faut définir des politiques au niveau des capabilities pour tous les cas d'utilisation des différents services. En effet, un même service au sens processus peut avoir des capabilities différentes en fonction de son utilisation. De la même manière que pour les politiques SELinux, le passage à l'échelle est compliqué. Il n'en reste pas moins que combinée à une implémentation réalisée avec le souci de la séparation de privilèges (donc avec un ensemble de capabilities facile à délimiter), leur utilisation rehausse grandement le niveau de sécurité du système. Merci à Gwendal pour les soucis de la vie courante en Python ;-) ■

Bibliographie

[LSM] Biondi, P., Raynal, F. (2008) Linux Security Modules (Linux Magazine HS 17).

[SELinux] Smalley, S., Vance, C., & Salamon, W. (2001). Implementing SELinux as a Linux security module (Vol. 1, p. 43). NAI Labs Report.

[FLASK] Lepreau, J., Spencer, R., Smalley, S., Loscocco, P., Hibler, M., & Andersen, D. (2006). The Flask Security Architecture: System Support for Diverse Security Policies.

[TE] Badger, L., Sterne, D. F., Sherman, D. L., Walker, K. M., & Haghighat, S. A. (1995, May). Practical domain and type enforcement for UNIX. In Security and Privacy, 1995. Proceedings., 1995 IEEE Symposium on (pp. 66-77). IEEE.

[Biba] Biba, K. J. (1977). Integrity considerations for secure computer systems (No. MTR-3153-REV-1). MITRE CORP BEDFORD MA.

[BL] Bell, D. E., & LaPadula, L. J. (1973). Secure computer systems: Mathematical foundations (No. MTR-2547-VOL-1). MITRE CORP BEDFORD MA.

[nmap] http://www.tolaris.com/2013/01/24/running-nmap-as-an-unprivileged-user/http://www.tolaris.com/2013/01/24/running-nmap-as-an-unprivileged-user/

PARALLÉLISEZ VOS TRANSFERTS DE FICHIERS

par Ganaël Laplanche [administrateur systèmes et développeur FreeBSD]
& Jean-Baptiste Denis [administrateur systèmes spécialiste Debian et arpenteur de couloirs]

Cet article est un retour d'expérience. Il a pour but de vous présenter comment, dans un institut de recherche en biologie, nous avons entamé la migration de plusieurs centaines de téraoctets de données vers un seul et même serveur de fichiers. Cette migration a lieu dans le cadre d'un achat de nouveau matériel, destiné à remplacer plusieurs serveurs de stockage hétérogènes.

1 Contexte

La tâche peut paraître simple mais malheureusement, les progrès technologiques aidant, les données que nous avons à gérer sont de plus en plus volumineuses ; ce prédicat est vrai pour les données professionnelles (nous avons ici des équipements capables de générer plusieurs Tio de données par jour), mais aussi, et pour l'instant dans une moindre mesure, pour nos données personnelles. À l'échelle de plusieurs centaines de Tio de données, les outils « classiques » tels que `cp` ou même `rsync` ne suffisent plus (nous verrons pourquoi).

Le contexte technologique est le suivant : notre infrastructure est constituée de serveurs hétérogènes (SunFire X4540, Netapp FAS980, HP X9000, IBM SONAS). Ils partagent environ 500 Tio de données pour 250 millions de fichiers à travers NFS v3 et CIFS et doivent être migrés vers une baie EMC/Isilon fraîchement acquise. Chaque serveur est relié au réseau via plusieurs liens gigabit agrégés (nos motifs d'accès à la donnée ne justifient pas - encore - le passage au 10 Gbps)Les données hébergées sur ces systèmes ne possèdent pas d'attributs étendus ou d'ACLs, juste des droits POSIX standards (ugo / rwx). Ce choix peut à première vue paraître étonnant mais il s'agit à nos yeux du moyen le plus simple permettant un accès mixte (CIFS/NFS) à la donnée (postes de travail, machines de calcul). Cette problématique pourrait d'ailleurs faire l'objet d'un autre article.

Nous gérons environ 3000 comptes utilisateurs.

Voici, dans le détail, le chemin suivi pour aboutir à la solution qui vous sera dévoilée dans la suite de l'article.

2 Les besoins

Tout d'abord, commençons par définir les besoins.

Le premier est évidemment de transférer les données de manière sûre vers la nouvelle baie de stockage. Il faudra conserver les fichiers spéciaux ainsi que les méta-données : utilisateurs et groupes propriétaires, droits associés, timestamps.

Cette solution doit permettre une interruption de service minimale pour les utilisateurs, idéalement qu'ils puissent continuer à travailler pendant le transfert des fichiers.

La solution doit également être assez souple pour transférer, au choix, de gros espaces de données ou bien des espaces plus petits afin de faciliter leur éventuelle ré-organisation.

Elle doit être re-jouable sans repartir de zéro : un plantage, une coupure électrique ou bien encore un défaut de climatisation sont vite arrivés.

Enfin elle ne doit bien évidemment pas coûter cher et réutiliser au maximum l'infrastructure dont nous disposons actuellement.

3 Choix de l'outil

Il n'est pas forcément évident de trouver de la littérature traitant de la migration de données avec un tel volume et un tel nombre de fichiers. Deux références récentes nous semblent cependant intéressantes. L'article de Jeff Layton *« Moving Your Data - It's Not Always Pleasant »*[1] évalue différents outils (dont certains intègrent la préservation des attributs

[1] http://www.admin-magazine.com/HPC/Articles/Moving-Your-Data-It-s-Not-Always-Pleasant

étendus et des ACLs) et amorce une réflexion de fond sur la problématique. Il mentionne la présentation de Marc Stearman du Lawrence Livermore National Laboratory ayant eu lieu lors du Lustre User Group en 2013 : *« Sequoia Data Migration Experiences »*[2]. C'est un autre contexte et une architecture différente ainsi qu'un autre ordre de grandeur. Nous vous invitons vivement à parcourir ces deux références !

Notre choix s'est logiquement porté sur **rsync**[3]. Il est éprouvé et robuste. **rsync** a la possibilité de ne transférer que les blocs modifiés d'un fichier, économisant ainsi la bande passante (et donc le temps nécessaire à la copie des données). Il est capable de reprise : en cas de problème, il sera en mesure de poursuivre à partir du fichier dont le transfert a été interrompu ; un travail de copie est donc re-jouable et sera très rapide si les fichiers source et destination sont identiques. Enfin, il dispose de nombreuses options avancées qui pourront nous être fort utiles.

Néanmoins, au-delà d'un certain nombre de fichiers, **rsync** montre certaines limites[4]. De plus, si on ne transfère pas uniquement des fichiers de tailles importantes, nous aurons du mal à maximiser l'utilisation de notre réseau, quelle que soit sa capacité. Ces limites peuvent être repoussées en adoptant une approche de type « diviser pour mieux régner », comme nous le verrons dans la suite de l'article. Auparavant, attardons-nous quelques instants sur les autres outils susceptibles d'être employés dans le cadre d'une migration de données.

Le premier qui vient à l'esprit est **cp**. Cet outil est simple, rapide et robuste, mais il présente évidemment un défaut majeur : il ne supporte pas la reprise. Une interruption de la copie et c'est toute une arborescence qu'il faut recopier ! C'est inenvisageable vue la quantité de données à traiter.

tar (tape archiver) pourrait sembler un bon candidat puisqu'il est initialement destiné à la sauvegarde/restauration de données. Il offre la possibilité de n'archiver que les fichiers nouvellement modifiés et celle de ne pas restaurer un fichier qui serait plus récent dans sa destination. C'est bien, mais non suffisant pour permettre la reprise car **tar** restaurera tout de même l'intégralité du reste de l'archive (hors fichiers plus récents dans la destination). Par ailleurs ces fonctionalités nécessitent de passer par un fichier d'archive (passer par un pipe - unidirectionnel - et faire transiter les données via **netcat**, par exemple, est par définition incompatible avec ce genre d'opérations), ce que nous ne pouvons nous permettre de faire car, en plus de nécessiter du temps, ceci nécessiterait beaucoup d'espace disque temporaire. Enfin, un dernier point est à noter : l'unité avec laquelle travaille **tar** est le fichier ; il n'est pas capable de transférer uniquement les blocs de données modifiées. La re-copie d'un fichier volumineux nouvellement modifié prendrait donc beaucoup de temps, y compris pour seulement quelques octets de différence.

À l'institut, nous effectuons nos sauvegardes sur bandes via Netbackup. Nous pourrions donc être tentés d'utiliser ce logiciel et le protocole NDMP pour restaurer les données sur le serveur de destination. Malheureusement, Netbackup utilise une version modifiée de **tar**, ce qui nous ramène aux limitations exposées précédemment (avec la joie, en prime, de travailler avec des bandes et le plaisir de payer des licences logicielles et matérielles très onéreuses). Une solution mixte NDMP + **rsync** pourrait être envisagée mais implique de disposer de suffisamment de bandes, de licences et ne répond pas à la question de la maximisation de l'utilisation du réseau.

Certains serveurs de fichiers propriétaires proposent des outils de migration permettant d'importer des données d'autres serveurs. C'est le cas d'Isilon, qui offre un outil de migration de données depuis des serveurs Netapp (**isi_vol_copy**). Malheureusement, cet outil ne couvre pas tous les types de serveurs dont nous disposons et ne permet pas de maximiser l'utilisation du réseau. D'une façon générale, les outils proposés par les constructeurs sont restreints à des technologies triées sur le volet et manquent cruellement de souplesse. Nous avons donc écarté ce type de solution.

Par curiosité, nous aurions également aimé évaluer des solutions dédiées de type ARX[5]. Ce sont des équipements qui se placent en coupure entre les données et les utilisateurs. Ils permettent un accès à la donnée quelle que soit sa source ainsi qu'une migration réellement transparente, sans aucun arrêt de service sauf lors de sa mise en place. Cette solution est par contre très onéreuse, mais elle est à mettre en regard avec le nombre de personnes impliquées dans une migration massive. Peut-être qu'au final il est financièrement plus intéressant de mettre en place ce type de solution plutôt que de dédier des ressources humaines pendant plusieurs mois à la migration. À noter que certains constructeurs intégrent ce type de fonctionnalités directement au sein de leurs baies.

Le logiciel n'est qu'une partie des outils à notre disposition. Il faut maintenant préciser que nous disposons également d'un cluster de calcul composé d'une soixantaine de machines physiques, fonctionnant sous GNU/Linux et disposant chacune d'une connexion gigabit. Ces machines sont connectées aux serveurs de fichiers via NFS v3. Idéalement, nous aimerions pouvoir bénéficier des connexions réseau de ces machines pour permettre de maximiser la bande passante utile vers le serveur de destination.

[2] http://www.youtube.com/watch?v=FamWy_c95Wo

[3] http://rsync.samba.org

[4] http://rsync.samba.org/FAQ.html#4

[5] http://www.f5.com/products/data-solutions/arx/overview/

4 Notre approche

Notre réflexion fut la suivante : sur nos serveurs, pour de gros fichiers, les E/S (entrées/sorties) disques sont négligeables et le goulet d'étranglement sera le réseau ; effectuer la copie depuis une seule machine saturerait son lien gigabit et ne serait pas optimal. Pour des petits fichiers (par exemple pour les espaces hébergeant des boîtes aux lettres au format maildir), les E/S pourraient s'avérer limitantes, particulièrement au niveau des serveurs de fichiers sources (la destination étant un scale-out NAS, a priori capable d'encaisser une quantité d'E/S bien plus importante que nos baies de générations plus anciennes). Si nous pouvions paralléliser les copies, nous utiliserions au maximum la bande passante disponible et nous gagnerions du temps. Idéalement, cette parallélisation permettrait à une machine de recopier en même temps des données depuis des serveurs de fichiers sources différents afin de limiter les risques de ralentissement liés aux problèmes d'E/S évoqués : on maximise l'utilisation de la bande passante sans saturer une unique source.

Le problème est que l'outil sélectionné (**rsync**) ne permet pas nativement ces acrobaties. **rsync** n'est pas distribué : l'application considère une source et une destination, parcourt la source, compare les fichiers et envoie les modifications à répliquer à la destination, éventuellement à travers le réseau.

Cet outil présente également un autre défaut : son temps de parcours est long (construire la liste des fichiers à transférer prend par exemple 7 jours pour 50 millions de fichiers dans nos tests en production) et la liste de fichiers générée est située en mémoire. Elle est donc volatile : en cas de crash, il faudra re-parcourir toute l'arborescence. **rsync** est bien capable de générer un fichier de travail (cf. mode batch), mais le fichier généré est binaire : nous aurions pu imaginer l'exploiter (par exemple le découper en plusieurs parties à traiter indépendamment), mais ce n'est pas possible nativement.

Un autre inconvénient est que **rsync** nécessite beaucoup de mémoire (environ 100 octets par fichier, cf. FAQ). Jusqu'à la version 3.0.0, il fallait attendre la fin du parcours complet de l'arborescence pour démarrer le transfert. Depuis, cette limitation a été levée[6], diminuant fortement les besoins en mémoire et permettant de gagner du temps sur la synchronisation. Néammoins, nous n'avons aucune idée du comportement qu'aura l'application sur plusieurs centaines de Tio de données si un nombre important de fichiers est à synchroniser. À cette échelle, la copie pourrait-elle simplement se terminer ?

Face à cette incertitude et dans l'idée de terminer le plus rapidement possible les synchronisations de données, il nous a paru évident qu'il était nécessaire de subdiviser notre arborescence afin d'effectuer les synchronisations en parallèle. Le problème est qu'il n'existe aucune méthode « clefs en mains » pour paralléliser **rsync**.

5 Subdiviser une arborescence de fichiers

Comment procéder pour subdiviser une arborescence de fichiers ? Par subdiviser, nous entendons diviser de manière équilibrée, c'est à dire en sous-arborescences contenant idéalement le même nombre de fichiers et faisant la même taille.

La première façon qui vient à l'esprit serait de descendre d'un ou plusieurs niveaux dans l'arborescence de fichiers et de fonctionner par sous-répertoires au lieu des répertoires parents. Cette manière de faire a l'avantage d'être simple et rapide car elle ne nécessite pas de parcourir l'arborescence pour décider de la découpe à effectuer. Malheureusement, elle n'est pas équilibrée : nous ne pouvons présumer du nombre de fichiers ou de la taille de chaque répertoire ; cette méthode génère des paquets inégaux en taille et en nombre de fichiers.

Une autre façon de faire serait de créer des listes de fichiers, indépendamment de leur positionnement dans l'arborescence à synchroniser. Cette granularité permettrait, en théorie, d'équilibrer (presque) parfaitement les parties à synchroniser (aussi bien en nombre de fichiers qu'en taille). En outre, **rsync** propose une option **--files-from** qui permettrait d'exploiter ces listes et lui éviterait la longue période de parcours initial. Nous avons opté pour cette méthode.

Malheureusement, il n'existe pas d'outil standard pour effectuer cette tâche de création de paquets qui semble pourtant un besoin assez courant (votre moteur de recherche préféré vous le prouvera). Il serait bien entendu possible d'écrire un script *shell* (ou *PERL* ou autre) en utilisant par exemple **find** et en découpant la liste des fichiers affichés selon des critères de taille et de nombre, mais nous avons préféré un outil en C pour des raisons de performance et de portabilité accrues.

6 Fpart

L'outil que nous avons mis au point se nomme **fpart**[7]. Il est développé en C et est disponible sous licence BSD modifiée. Nous ne détaillerons pas la compilation et l'installation très classiques de **fpart** mais sachez qu'il est d'ores et déjà disponible dans l'arbre officiel des ports FreeBSD (**sysutils/fpart**) et en cours d'intégration sous Debian.

Pour la petite histoire, le développement de cet outil a débuté plusieurs mois plus tôt ; il n'était pas destiné initialement à la migration de données mais

[6] http://rsync.samba.org/ftp/rsync/src/rsync-3.0.0-NEWS

[7] http://contribs.martymac.org, http://sourceforge.net/projects/fpart

Abonnez-vous !

Profitez de nos offres d'abonnement spéciales disponibles au verso !

Téléphonez au 03 67 10 00 20 ou commandez par le Web

Économisez plus de 25%*

* Sur le prix de vente unitaire France Métropolitaine

11 Numéros de GNU/Linux Magazine

ABONNEMENT

11 GNU LINUX MAGAZINE / FRANCE

58€*

au lieu de 82,50 €* en kiosque

Économie : 24,50 €*

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE
Pour les tarifs hors France Métropolitaine, consultez notre site : boutique.ed-diamond.com

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.
- Économisez 24,50 €/an ! (soit plus de 3 magazines offerts !)

4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur boutique.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	

❑ Je souhaite recevoir les offres promotionnelles et newsletter des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles de nos partenaires.

GNU LINUX MAGAZINE / FRANCE

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/cgv** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

PROFITEZ DE NOS OFFRES D'ABONNEMENT SPÉCIALES POUR LIRE PLUS ET FAIRE DES ÉCONOMIES !

➡ Abonnement

offre 1 — ABONNEMENT — 11 LINUX MAGAZINE / FRANCE — **58€*** — au lieu de 82,50€** en kiosque — Economie : 24,50 €

Vous pouvez également vous abonner sur :

boutique.ed-diamond.com

ou par Tél. : +33 (0)3 67 10 00 20 / Fax : +33 (0)3 67 10 00 21

NOUVEAU !

numerique.ed-diamond.com
pour vous abonner et acheter vos magazines en format numérique (PDF)

unixgarden.com
pour retrouver une sélection d'articles des Éditions Diamond

* Tarifs France Métro (F)
** Base tarifs kiosque zone France Métro (F)

➡ Voici nos offres d'abonnements groupés incluant GLMF

offre 3 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE + 6 LINUX PRATIQUE — **85€*** — au lieu de 121,50€** en kiosque — Economie : 36,50 €

offre 4 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE + 6 Hors-séries — **89€*** — au lieu de 130,50€** en kiosque — Economie : 41,50 €

offre 5 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE + 6 MISC — **90€*** — au lieu de 133,50€** en kiosque — Economie : 43,50 €

offre 6 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 LINUX PRATIQUE — **119€*** — au lieu de 169,50€** en kiosque — Economie : 50,50 €

offre 7 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 MISC — **124€*** — au lieu de 181,50€** en kiosque — Economie : 57,50 €

offre 8 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 LINUX PRATIQUE + 6 MISC — **154€*** — au lieu de 220,50€** en kiosque — Economie : 66,50 €

offre 9 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 LINUX PRATIQUE + 6 MISC + 6 LINUX ESSENTIEL — **184€*** — au lieu de 259,50€** en kiosque — Economie : 75,50 €

offre 12 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 LINUX PRATIQUE +3 HS + 6 MISC +2 HS + 6 LINUX ESSENTIEL — **215€*** — au lieu de 301,50€** en kiosque — Economie : 86,50 €

offre 2 — ABONNEMENTS GROUPÉS — 6 LINUX ESSENTIEL + 6 LINUX PRATIQUE — **60€*** — au lieu de 78,00€** en kiosque — Economie : 18,00 €

➡ Voici nos autres offres d'abonnements groupés

offre 10 — ABONNEMENTS GROUPÉS — 6 MISC + 2 Hors-séries — **48€*** — au lieu de 69,00€** en kiosque — Economie : 21,00 €

offre 11 — ABONNEMENTS GROUPÉS — 6 LINUX PRATIQUE + 3 Hors-séries — **48€*** — au lieu de 63,00€** en kiosque — Economie : 15,00 €

offre 15 — ABONNEMENTS GROUPÉS — 6 LINUX ESSENTIEL + 6 LINUX PRATIQUE +3 HS — **78€*** — au lieu de 102,00€** en kiosque — Economie : 24,00 €

➡ Nos Tarifs s'entendent TTC et en euros

		F France Métro	D DOM	T TOM	Zone 1 Europe	Zone 2 Afrique / Orient	Zone 3 Amérique	Zone 4 Asie / Océanie
1	Abonnement GLMF	58 €	78 €	94 €	80 €	87 €	84	80 €
2	Abonnement LPE + LP	60 €	86 €	105 €	88 €	96 €	92 €	89 €
3	Abonnement GLMF + LP	85 €	120 €	145 €	123 €	134 €	129 €	124 €
4	Abonnement GLMF + GLMF HS	89 €	122 €	147 €	125 €	136 €	131 €	126 €
5	Abonnement GLMF + MISC	90 €	128 €	151 €	130 €	141 €	136 €	131 €
6	Abonnement GLMF + GLMF HS + Linux Pratique	119 €	164 €	198 €	168 €	183 €	176 €	170 €
7	Abonnement GLMF + GLMF HS + MISC	124 €	172 €	204 €	175 €	190 €	183 €	177 €
8	Abonnement GLMF + GLMF HS + MISC + LP	154 €	214 €	255 €	218 €	237 €	228 €	221 €
9	Abonnement GLMF + GLMF HS + MISC + LP + LPE	184 €	258 €	309 €	263 €	286 €	275 €	266 €
10	Abonnement MISC + MISC HS	48 €	66 €	76 €	66 €	72 €	69 €	68 €
11	Abonnement LP + LP HS	48 €	65 €	78 €	66 €	72 €	70 €	68 €
12	Abonnement GLMF + GLMF HS + MISC + MISC HS + LP + LP HS + LPE	215 €	297 €	355 €	302 €	329 €	317 €	307 €
15	Abonnement LPE + LP + LP HS	78 €	109 €	132 €	111 €	121 €	117 €	113 €

- ZONE 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède, Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande, Estonie, Croatie, Slovenie, Slovaquie, Republique Tcheque, Pologne, Bielorussie, Bosnie Herzegovine, Bulgarie, Chypre, Georgie, Hongrie, Lettonie, Lituanie, Macedoine, Malte, Moldova, Roumanie, Russie, Serbie, Ukraine, Albanie, Armenie, ...
- ZONE 2 : Algerie, Maroc, Tunisie, Turquie, Afrique du Sud, Seychelle, Senegal, Israel, Palestine, Syrie, Jordanie, Botswana, Cameroun, Cap Vert, Comores, Rep.Dom. Congo, Cote d'Ivoire, Egypte, Kenya, Libye, Madagascar, Nigeria, ...
- ZONE 3 : Canada, Etats Unis, Guyana, Haiti, Republique Dominicaine, Jamaique, Argentine, Bresil, Cuba, Mexique, ...
- ZONE 4 : Australie, Japon, Chine, Coree du Nord, Coree du Sud, Inde, Indonesie, Nouvelle Zelande, Taiwan, Thailande, Vietnam, ...

Mes choix :

Mon 1er choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 2ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 3ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
	Je sélectionne ma zone géographique (F à Zone 4) :	
	J'indique la somme due : (Total)	€

Exemple : je souhaite m'abonner à l'offre GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC (offre 7) et je vis en Belgique (zone 1), ma référence est donc 7zone1 et le montant de l'abonnement est de 175 euros.

Je choisis de régler par :

- ❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond
- ❑ Carte bancaire n° ⌊_⌊_⌊_⌊_ ⌊_⌊_⌊_⌊_ ⌊_⌊_⌊_⌊_ ⌊_⌊_⌊_⌊_
  Expire le : ⌊_⌊_⌊_⌊_
  Cryptogramme visuel : ⌊_⌊_⌊_

Date et signature obligatoire

GNU/LINUX MAGAZINE / FRANCE

juste à la création de « partitions » de fichiers à partir d'une liste de répertoires fournis sur la ligne de commandes. Ces "partitions" forment des sous-ensembles disjoints des répertoires fournis en argument et sont équilibrées (en taille et en nombres de fichiers) selon l'algorithme glouton[8]. Dans son mode par défaut, **fpart** parcourt une (ou plusieurs) arborescence(s) pour générer, à la fin du parcours, des partitions, selon les critères définis par l'utilisateur.

Voici un exemple d'utilisation basique, que nous détaillerons dans la prochaine section :

```
$ fpart -n 3 -o var-parts /data/src
267781 file(s) found.
Part #0: size = 2658718909, 89260 file(s)
Part #1: size = 2658718908, 89261 file(s)
Part #2: size = 2658718908, 89260 file(s)
```

fpart garantit que l'union de toutes les partitions générées permet de reconstruire l'ensemble des répertoires et fichiers examinés.

Le projet de migration de données a été l'occasion d'ajouter à **fpart** une fonctionnalité supplémentaire : la possibilité de générer en direct (mode *live*) les partitions et de déclencher une commande juste après la génération de chaque partition (*post-hook*), sans attendre la fin du parcours complet du système de fichiers. Cette fonctionnalité peut être utilisée pour déclencher immédiatement **rsync** sur le sous-ensemble de fichiers produits.

En mode standard, **fpart** permet de générer un nombre souhaité de partitions, ainsi que des partitions limitées par leur taille et leur nombre de fichiers. En mode live, générer un nombre *n* de partitions équilibrées n'est pas possible car **fpart** ne peut présumer du nombre de fichiers qu'il est en train d'examiner (la répartition exacte ne peut se faire qu'une fois tout le système de fichiers parcouru). Il peut cependant générer des partitions de taille et de nombre de fichiers limités. Ceci, bien que non optimal, suffit largement à notre besoin.

Le mode *live* de **fpart** a un autre avantage : l'économie de mémoire. Dans ce mode, **fpart** peut parcourir plusieurs centaines de Tio avec une empreinte mémoire limitée (et quasi-constante, de quelques Mio seulement) car une seule partition est présente en mémoire à un instant t, à la différence du mode standard qui a une empreinte mémoire beaucoup plus importante pour trier les données de manière précise.

En utilisant **fpart**, nous limitons les risques évoqués précédemment en faisant travailler **rsync** sur de petits espaces de données. Cette méthode de travail nous permet également de répondre au besoin de parallélisation. Enfin, par le biais d'un parcours d'arborescence unique, nous limitons les E/S et nous sommes en mesure de démarrer les synchronisations pendant ce temps incompressible que représente un parcours de plusieurs centaines de Tio de fichiers.

7 Premiers exemples

Reprenons notre exemple d'utilisation de **fpart** cité précédemment :

```
$ fpart -n 3 -o var-parts /data/src
267781 file(s) found.
Part #0: size = 2658718909, 89260 file(s)
Part #1: size = 2658718908, 89261 file(s)
Part #2: size = 2658718908, 89260 file(s)
```

Cette commande parcourt intégralement **/data/src** et génère, en fin de parcours, 3 fichiers de partitions : **var-parts.[0-2]**. Ici, **fpart** est lancé en mode standard et il n'est pas question de *hooks*, mais juste de subdiviser notre arborescence en plusieurs partitions. Les fichiers générés contiennent la liste des fichiers parcourus ; les 3 listes sont équilibrées en nombre et en taille.

Voici un exemple plus avancé impliquant l'utilisation du mode *live* :

```
$ mkdir foo && touch foo/{bar,baz}
$ fpart -L -f 1 -o /tmp/part.out -W \
    'echo == ${FPART_PARTFILENAME} == ; cat ${FPART_PARTFILENAME}' foo/
== /tmp/part.out.0 ==
foo/bar
== /tmp/part.out.1 ==
foo/baz
2 file(s) found.
```

Ici, nous générons deux fichiers vides : **foo/bar** et **foo/baz** puis exécutons **fpart** en mode *live* (option **-L**) pour générer des partitions de 1 fichier (option **-f**) et les écrire dans **/tmp/part.out.<n>** (option **-o**). Pour chaque partition générée, le hook (option **-W**) suivant est exécuté à la volée :

```
echo == ${FPART_PARTFILENAME} == ; cat ${FPART_PARTFILENAME}
```

La variable **${FPART_PARTFILENAME}** est générée par **fpart** et fait partie de l'environnement du *hook* : elle représente le fichier de partition venant d'être produit. Ce *hook* affiche donc le nom du fichier de partition, ainsi que son contenu (dans notre cas, 1 fichier à chaque fois).

8 Un exemple avec rsync et GNU parallel

Ces exemples sont triviaux ; voyons désormais un exemple plus avancé, se rapprochant de notre problème initial. L'exemple suivant montre comment il est possible d'utiliser

[8] http://fr.wikipedia.org/wiki/Algorithme_glouton

fpart pour lancer, depuis la même machine, 3 synchronisations en parallèle. Le mode *live* de **fpart** permet d'exécuter des *hooks* synchrones, c'est-à-dire que l'outil attend la fin du hook pour poursuivre le parcours du système de fichiers. En d'autres termes, il n'intègre pas de scheduler (ce n'est pas son rôle). Il faudra donc s'appuyer sur un scheduler externe. L'exemple donné ici utilise GNU Parallel[9] et plus particulièrement la commande **sem**.

Nous allons ici synchroniser **/data/src** vers **/data/dest**.

Tout d'abord, placez-vous dans le répertoire source à examiner. Ceci est nécessaire car l'option **--files-from** de **rsync** utilise par défaut une liste de fichiers relative au répertoire source :

```
$ cd /data/src
```

Ensuite, exécutez **fpart** depuis ce répertoire :

```
$ fpart -L -f 10000 -x '.snapshot' -x '.zfs' -Z -o /tmp/part.out -W \
  'sem -j 3
    "rsync -av --files-from=${FPART_PARTFILENAME}
      /data/src/ /data/dest/"' .
```

Cette commande exécute **fpart** en mode *live* (option **-L**) en lui demandant de produire des partitions d'au plus 10 000 fichiers (option **-f**). **fpart** ne listera pas les fichiers ou répertoires nommés **.snapshot** ou **.zfs** (options **-x**) mais listera les répertoires vides ou non accessibles (option **-Z**, cette option est nécessaire lorsqu'on travaille avec **rsync** pour s'assurer de re-créer à l'identique l'arborescence dans le répertoire cible). Enfin, chaque partition sera écrite dans **/tmp/part.out.<n>** (option **-o**) et utilisée dans le *hook* post-partition (option **-W**) suivant :

```
sem -j 3
    "rsync -av --files-from=${FPART_PARTFILENAME} /data/src/ /data/dest/"
```

Ce hook utilise la commande **sem** de GNU parallel pour planifier au maximum 3 **rsync** conccurents. Ils sont exécutés de la manière suivante :

```
rsync -av --files-from=${FPART_PARTFILENAME} /data/src/ /data/dest/
```

La variable **${FPART_PARTFILENAME}** est directement utilisée par **rsync** et lui indique la liste des noms de fichiers et répertoires à transférer.

Bien entendu, exécuter plusieurs synchronisations d'un même espace de données (provenant d'un serveur source unique) sur la même machine présente a priori peu d'avantages (à moins que le client ou le serveur soit en mesure de paralléliser les E/S si l'emplacement physique des données le permet). Cet exemple donne simplement une idée de la manière dont peut être utilisé **fpart**.

Imaginez que l'on remplace le scheduler local (**sem**) par celui de notre cluster de calcul...

9 Utilisation avec notre cluster

C'est l'idée que nous avons déjà évoquée précédemment : utiliser notre cluster de calcul pour effectuer les transferts. Ainsi, nous mettons à contribution l'infrastructure existante et nous maximisons la bande passante disponible vers la baie de destination. Le cluster est composé d'une soixantaine de machines, toutes connectées via des liens gigabit à un commutateur central. Des liens agrégés partent également depuis ce commutateur vers les serveurs de stockage. Chaque nœud fonctionne sous GNU/Linux (CentOS) et monte les espaces de stockage via NFS v3 (tcp). L'orchestration de ces machines se fait via le logiciel OGS[10] (Open Grid Scheduler), descendant direct de SGE (Sun Grid Engine).

Pour la migration, nous n'utiliserons en réalité qu'une sous-partie du cluster, dédiée temporairement à la copie des données. Nous associerons à ce *pool* de 5 machines réservées un nombre maximum de *jobs* exécutables en parallèle.

Le cluster dispose d'un nœud particulier capable de soumettre des *jobs* de calcul aux nœuds du cluster ; c'est cette "tête" que nous utiliserons pour soumettre les *jobs* **rsync**. Cette tête de soumission monte les mêmes espaces de stockage que les nœuds.

Les montages effectués sur chacune des machines sont les suivants :

- L'intégralité des espaces à synchroniser, provenant de nos serveurs de fichiers à migrer. Ceux-ci sont montés dans un répertoire dédié, accessible uniquement à *root* et en lecture seule. L'option **no-root-squash** a été positionnée sur les exports afin de pouvoir accéder à l'intégralité des données.
- L'espace de destination situé sur la baie Isilon. Celui-ci est monté dans ce même répertoire spécifique en lecture-écriture et exporté avec l'option **no-root-squash**.

Il convient ici de préciser que les utilisateurs « normaux » du cluster n'ont pas accès aux machines participant à la migration. Toutefois, il faut garder à l'esprit les risques potentiels d'un montage **no-root-squash**. Nous y reviendrons.

Voici le scénario imaginé : pour chaque espace à migrer (concrètement, pour chaque montage NFS, toutes machines sources confondues), nous exécuterons, depuis la tête de soumission du cluster, un script enrobant **fpart**, **qsub** (la commande pour soumettre un job à OGS, l'équivalent de **sem**

[9] http://www.gnu.org/software/parallel

[10] http://gridscheduler.sourceforge.net

dans notre exemple) et **rsync**. Ce script déterminera les partitions à synchroniser et soumettra à OGS les *jobs* **rsync** à effectuer. Le *scheduler* d'OGS, lorsqu'un nœud sera disponible, lui fera exécuter la copie des données.

Pour que ceci fonctionne il faut, comme nous l'avons précisé, que tous les espaces sources soient montés sur la tête de soumission afin de permettre à **fpart** de parcourir les fichiers. Sur les nœuds du cluster susceptibles de copier les données, il faut monter les espaces sources et l'espace de destination. Enfin, il a fallu dédier un espace NFS à **fpart**, monté sur toutes les machines, afin qu'il y stocke les fichiers des partitions générées. Cet espace servira d'espace d'échange des listes entre la machine de soumission (qui génère les listes) et les nœuds effectuant la copie des données (qui consomment ces listes).

Le script utilisé ressemble aux exemples étudiés précédemment à la différence du scheduler utilisé, qui est celui d'OGS. Voici les lignes clefs de ce script. Ne vous laissez pas impressionner, nous allons le décortiquer ensemble :

```
1) QSUB_COMMAND="sudo /bin/sh -c '${RSYNC_BIN} -av --numeric-ids --files-
from=\\\"\${FPART_PARTFILENAME}\\\" \\\"${SRC_PATH}/\\\" \\\"${DST_PATH}/\\\"'"
2) FPART_POSTHOOK="sudo -i -u '${QSUB_RUNAS}' qsub -b y -N \"${QSUB_JOBNAME}-
fpart-$$\" -o \"${FPART_LOGDIR}/$$-\${FPART_PARTNUMBER}.stdout\" -e \"${FPART_
LOGDIR}/$$-\${FPART_PARTNUMBER}.stderr\" -q '${QSUB_QUEUE}' -l '${QSUB_
RESOURCE}=1' \"${QSUB_COMMAND}\""
3) ${FPART_BIN} -f "${FPART_MAXPARTFILES}" -s "${FPART_MAXPARTSIZE}" -o
"${OUTPART_TEMPLATE}" -x '.zfs' -x '.snapshot*' -Z -L -v -W "${FPART_POSTHOOK}"
. 2>&1 | tee -a ${FPART_LOGDIR}/fpart.log
```

Les lignes sont présentées par « couches croissantes » : la ligne 3 encapsule la ligne 2 qui encapsule la ligne 1.

La première ligne est la commande qui sera exécutée sur les nœuds. Afin d'avoir accès aux données, elle utilise **sudo** pour passer root. Ce job devra évidemment être soumis par un utilisateur privilégié sur le cluster. On retrouve l'appel à **rsync** utilisant le nom du fichier de la partition venant d'être générée (**${FPART_PARTFILENAME}**). Cette variable sera renseignée par **fpart** au moment de l'exécution du *hook*. On retrouve aussi d'autres options de **rsync**, notamment le fait de copier les *uid* et *gid* sans passer par une résolution de nom intermédiaire (**--numeric-ids**).

La seconde ligne enrobe la première et constitue le *post-hook* en tant que tel ; il s'agit de l'appel à **qsub** (la commande de soumission de *jobs* d'OGS) qui soumettra la commande **sudo** étudiée précédemment. Elle est effectuée sous l'identité de l'utilisateur privilégié dont nous parlions à la première ligne (**${QSUB_RUNAS}**). Nous donnons un nom à ce job : **${QSUB_JOBNAME}-fpart-$$** (la variable **${QSUB_JOBNAME}** est renseignée interactivement au démarrage du script) et nous traçons les sorties standard et d'erreur du job dans des fichiers de log dédiés (**${FPART_LOGDIR}/$$-\${FPART_PARTNUMBER}.stdout** et **${FPART_LOGDIR}/$$-\${FPART_PARTNUMBER}.stderr**). Les noms de ces fichiers sont générés par le biais de variables **fpart** et ceux-ci sont créés sur le même espace NFS que celui permettant de stocker les listes de fichiers à synchroniser (puisque tous les nœuds, ainsi que la tête, y ont accès). Enfin, nous précisons la queue (**${QSUB_QUEUE}**) et la ressource (**${QSUB_RESOURCE}**) utilisées par le job.

La dernière ligne enrobe les deux premières et constitue cette fois l'appel principal à **fpart**. On retrouve nos paramètres, initialisés plus haut dans le script (**${FPART_MAXPARTFILES**, **${FPART_MAXPARTSIZE}**), ainsi que le template des noms de fichiers de partitions générées (**${OUTPART_TEMPLATE}**). Ici aussi, tout est tracé, dans un fichier de log général dédié aux scans fpart (**${FPART_LOGDIR}/fpart.log**), via **tee**.

Voilà l'idée globale du script que nous avons utilisé. Nous n'avons détaillé ici que les options principales ; vous pourrez creuser un peu si le cœur vous en dit. La partie la plus difficile dans l'écriture de ce script est de savoir à quel moment les variables doivent être évaluées et de les protéger en conséquence.

Voici un schéma récapitulant cette architecture :

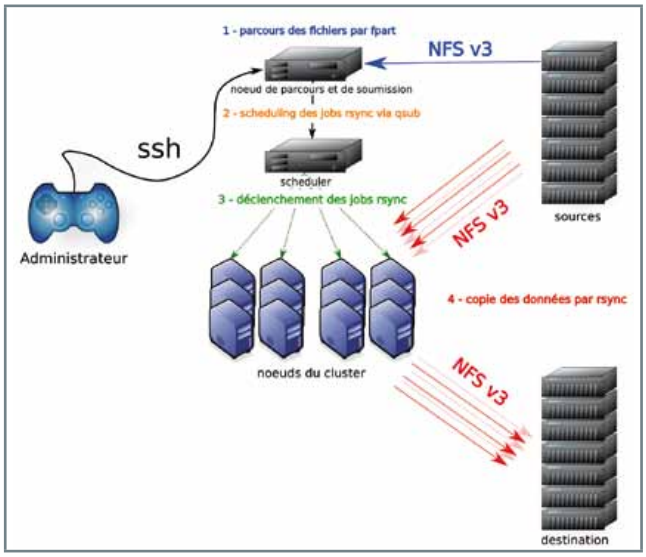


Illustration 1 : Architecture de migration

10 Les étapes de la migration

10.1 Une migration en 3 étapes

Pour chaque espace de données à migrer, la synchronisation se déroulera en 3 étapes :

- une pré-migration, qui se déroule en ligne ; ceci signifie que les utilisateurs peuvent continuer à travailler pendant ce temps. Cette étape comprend un passage de

`fpart`, ainsi que la soumission et l'exécution des *jobs* de synchronisation en résultant.

- une ou plusieurs migrations intermédiaires, en ligne également, afin d'éviter de laisser la destination déjà synchronisée dériver des données originales si trop de temps s'écoule avant l'étape finale.
- l'étape finale, qui correspond à la bascule définitive de l'espace de données vers la nouvelle baie de stockage. Cette étape doit être la plus courte possible car elle se fait hors ligne, c'est-à-dire que l'accès est bloqué pour les utilisateurs, afin d'éviter qu'ils ne modifient leurs données pendant la migration. Pour que cette étape soit courte, elle doit se situer dans le temps le plus tôt possible après une migration intermédiaire.

Ce plan de synchronisation nous garantit un maximum d'efficacité et de sécurité avec un minimum d'interruption de service.

10.2 L'étape finale

Détaillons cette dernière étape car elle est un peu particulière : elle consiste en un `rsync` unique, non exécuté depuis `fpart`.

La raison de cette dernière synchronisation est simple : jusqu'ici, nous n'avons fait que synchroniser les fichiers ajoutés et modifiés, mais nous n'avons jamais supprimé les fichiers ayant disparu de la source. Les fichiers se sont donc accumulés dans la cible au fil des « migrations préparatoires ». Cette dernière synchronisation doit permettre d'obtenir un miroir **exact** de l'espace d'origine en supprimant les fichiers surnuméraires de la cible. Sans cette dernière passe, l'utilisateur qui a supprimé des fichiers après les premières synchronisations les verrait "réapparaître" dans la destination après la migration, ce qui n'est pas souhaitable.

Pourquoi avoir fait le choix d'un `rsync` final unique et non distribué via `fpart` ? Il y a deux raisons.

La première est que l'utilisation de `fpart` ne ferait probablement pas gagner de temps ici : les données à transférer sont minimales, donc le temps nécessaire pour cette étape tend vers le temps de parcours du système de fichiers.

La seconde raison est la plus importante et elle est technique : l'option à utiliser pour supprimer les fichiers de la cible est `--delete` ; malheureusement, elle est incompatible avec l'option `--files-from` si la liste fournie contient des fichiers (ce qui est notre cas).

Pour utiliser l'option `--delete` de manière distribuée, il faudrait faire en sorte de passer une liste de répertoires (uniquement) à l'option `--files-from`. L'option `-d` de `fpart`, qui demande d'inclure les noms des répertoires à partir d'une certaine profondeur de chemin, aurait pu être une piste. Dans tous les cas, et comme évoqué précédemment, un découpage par répertoire ne produit que rarement des listes équilibrées. Ainsi, vu le peu d'avantages de cette méthode et la complexité supplémentaire qu'elle induirait, nous avons fait le choix d'un `rsync` final unique.

Note

Pour ceux qui seraient tentés d'utiliser l'option `--delete` avec l'option `--files-from` et une liste de répertoires, il faut dans ce cas utiliser également l'option `-r` pour que le contenu de chaque répertoire soit synchronisé en plus de l'entrée du répertoire lui-même.

Ce choix peut paraître étonnant car nous émettions des doutes quant au passage à l'échelle de `rsync` sur un grand nombre de fichiers. Pour cette dernière étape, nous en avons fait le pari, les actions à effectuer étant minimales (suppressions de fichiers) et les fichiers concernés a priori peu nombreux.

11 Résultats

Afin d'illustrer les performances de la solution, nous vous proposons les résultats obtenus à partir de deux jeux de données de 100 Gio :

- un répertoire « home » caractéristique contenant des pdfs, des sources, des dépôts de sources, de la musique, de la vidéo, des mails, dupliqué 3 fois pour un total de 100 Gio.

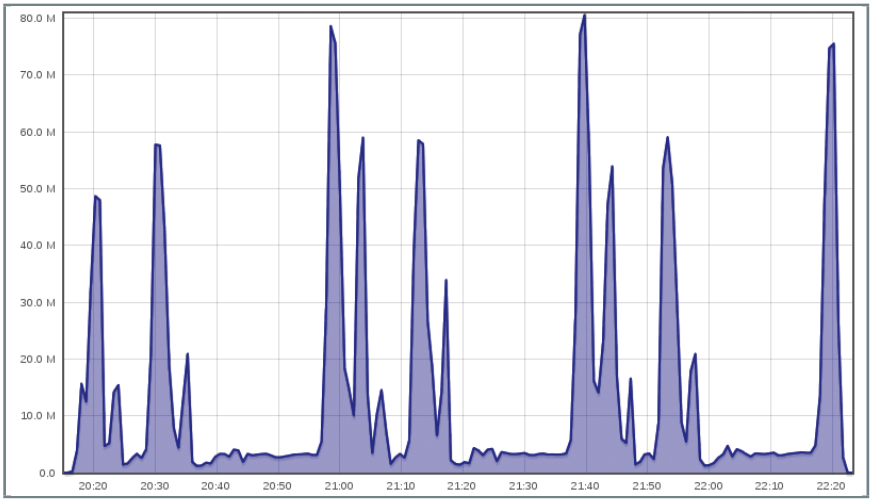


Illustration 2 : rsync simple - 2h08 - 13 Mio/s en moyenne

- un espace de données « artificielles » ne contenant que des fichiers de 70 Mio. Il s'agit de 1500 archives du noyau Linux au format xz réparties dans 15 répertoires pour un total de 100 Gio.

Les tests reposent sur l'archicture exposée dans le schéma précédant. Cinq nœuds du cluster sont ici dédiés à nos tests. Le serveur de fichiers source n'était pas en production à l'inverse du serveur destination (plusieurs milliers d'utilisateurs, accès CIFS et NFS, données scientifiques accessibles sur le cluster, messagerie...). Les tests ont été lancés à plusieurs reprises et nous n'avons pas relevé de divergences notables dans l'allure des courbes de trafic. Dans ces conditions imparfaites mais néanmoins raisonnables, nous nous intéresserons donc plus aux tendances qu'aux résultats bruts. Entre chaque test, nous avons pris soins de vider les caches disques (**`echo 3 > /proc/sys/vm/drop_caches`**).

Voici tout d'abord les résultats d'un simple **`rsync`** sur le jeu de données « home » (obtenus à l'aide de l'interface web de ganglia) (voir Illustration 2 ci-contre).

100 Gio en environ 2h10, soit 13 Mio / seconde en moyenne. Le motif d'utilisation très irrégulier de la bande passante nous confirme que nous avons affaire à des tailles de fichiers bien différentes dont certaines auraient bien du mal à saturer un lien réseau à seulement 100 Mbits/s.

Nous lançons maintenant le même transfert avec **`fpart`** qui soumet des **`rsync`** à notre gestionnaire de tâches sur le cluster. **`fpart`** a été lancé afin de générer des paquets de moins de 20 000 fichiers et d'au plus 5 Gio au total, ce qui représente ici 53 « paquets » à transférer (certains paquets sont limités par le nombre de fichiers, d'autres par la taille). Le temps incompressible nécessaire au parcours de l'arborescence est de 25 minutes environ. Durant ces 25 minutes, dès qu'un paquet est créé, **`fpart`** soumet le transfert au gestionnaire de tâches. Nous autorisons deux **`rsync`** à tourner en même temps. Voici une illustration de l'état du gestionnaire de tâches en cours de transfert :

```
jbd@tete ~ > qstat -u jbdenis
job-ID  prior   name       user         state submit/start at     queue
---------------------------------------------------------------------------------
2503769 0.00000 fpartsonic jbdenis      r     08/08/2013 13:06:02 datasync@n055.cluster
2503770 0.00000 fpartsonic jbdenis      r     08/08/2013 13:07:15 datasync@n057.cluster
2503771 7.16667 fpartsonic jbdenis      qw    08/08/2013 12:34:55
2503772 6.75000 fpartsonic jbdenis      qw    08/08/2013 12:35:47
2503773 6.50000 fpartsonic jbdenis      qw    08/08/2013 12:35:47
2503774 6.33333 fpartsonic jbdenis      qw    08/08/2013 12:35:53
2503775 0.00000 sync-fpart jbdenis      hqw   08/08/2013 12:35:53
```

Il y a deux tâches en cours (« state » r) et des tâches en attente (« state » qw pour « queue wait »). La dernière tâche en « hold queue wait » dépend des tâches précédentes : il s'agit d'un job final destiné à nous envoyer un message lorsque toutes les tâches sont terminées.

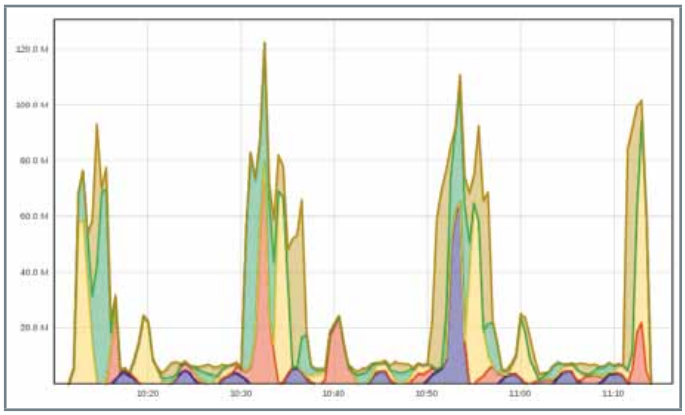


Illustration 3 : fpart + 2 rsync - 1h01 - 28 Mio/s en moyenne

100 Gio en environ 1h01, soit 28 Mio / seconde en moyenne. Chaque couleur du graphique représente une machine différente. Le gestionnaire de tâches choisit une machine sur les cinq de notre architecture pour lancer chaque tâche se trouvant dans sa file d'attente. Le mécanisme d'élection se base ici sur la charge processeur de chaque machine (une métrique de charge réseau aurait été plus pertinente dans notre cas). À un instant donné, il ne peut y avoir que deux machines qui interviennent dans le transfert. Cela n'est pas le cas sur certaines zones du graphiques à cause du délai entre chaque prise de performances de ganglia (toutes les 15 secondes par défaut). Le graphique présente les performances empilées (« stacked » dans la terminologie ganglia). Par exemple, vers 10h32, le débit cumulé était de 120 Mio/s sur l'ensemble des cinq machines (même si il n'y en a que deux qui participent au transfert)

Même test, avec **`fpart`** et 4 **`rsync`** en parallèle :

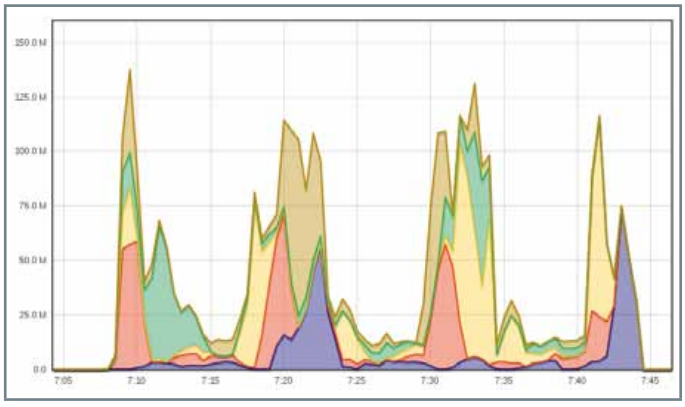


Illustration 4 : fpart + 4 rsync - 36 minutes - 46 Mio/s en moyenne

100 Gio en environ 36 minutes, soit 46 Mio / seconde en moyenne.

Enfin, avec 8 **`rsync`** en parallèle. Comme nous n'avons que 5 machines à notre disposition, il faut garder à l'esprit que plusieurs **`rsync`** s'exécuteront sur la même machine.

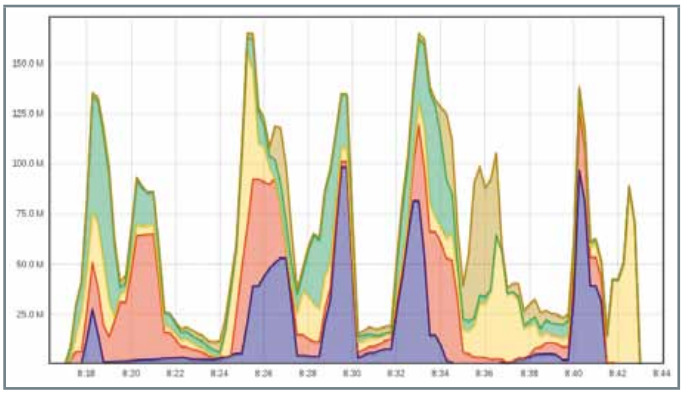


Illustration 5 : fpart + 8 rsync - 35 minutes - 46 Mio/s en moyenne

Il n'y a pas d'amélioration par rapport à l'utilisation de 4 **rsync**. Nous arrivons très certainement aux limites en entrées/sorties du serveur source en lecture et/ou du serveur destination en écriture.

Nous avons relancé ces tests sur notre jeu de données « artificielles » : 1500 noyaux linux compressés au format xz, soit 100 Gio de données. Voici les résultats avec 2, 4, et 8 **rsync** en parallèle, sur le même graphique :

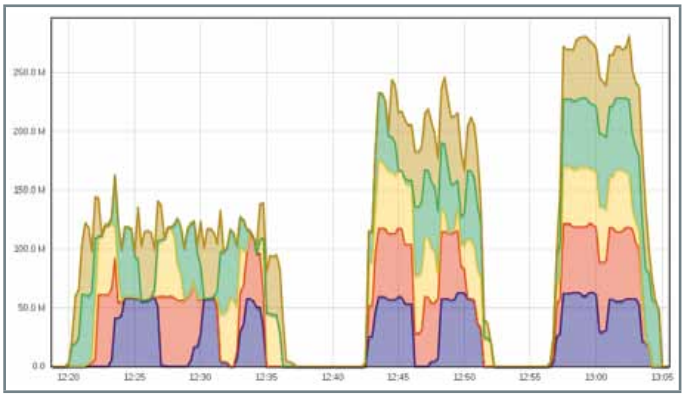


Illustration 6 : 100 Gio de fichiers de 70 Mio : fpart + 2 rsync, + 4 rsync, + 8 rsync

17 minutes avec 2 **rsync** (98 Mio/s), 10 minutes avec 4 **rsync** (166 Mio/s), 7 minutes avec 8 **rsync** (238 Mio/s), cela fonctionne plutôt bien. Dommage que nous n'ayons eu que 5 machines à notre disposition, peut-être que la version avec 8 **rsync** aurait donné de meilleurs résultats.

Si l'on ne regarde que les débits sur les données « réelles » de type répertoire personnel, on peut penser qu'ils sont décevants : 46 Mio/s cumulés sur 5 liens gigabit, c'est 10 % du débit théorique. Le type, la taille et le nombre de fichiers à transférer ont une importance capitale. Par exemple, il est illusoire d'espérer saturer un lien gigabit avec une arborescence de boîtes au format maildir de plusieurs Tio. C'est ce qui se passe ici, à une moindre échelle. Il est beaucoup plus rapide de transférer de gros fichiers qu'un ensemble de fichiers plus petits. C'est une évidence bien souvent oubliée par les personnes établissant les plannings de migration avec des conséquences que tout le monde peut imaginer.

Le gain est par contre très appréciable, surtout lorsqu'on parle de plusieurs dizaines de Tio de données et de plusieurs semaines de transfert. Nous sommes parvenus à effectuer des transferts de données utilisateurs en une semaine au lieu d'un mois tout en pouvant ajuster, en direct, le nombre de **rsync** en parallèle en fonction des impacts sur la production.

12 Limites

Si la méthode de migration présentée ici a plusieurs avantages, elle a aussi quelques limites et inconvénients. Les voici.

La plus importante des limites de cette méthode est sans doute qu'elle nécessite un accès POSIX (à travers la couche VFS) au système de fichiers source. Cette limite peut sembler anodine mais, en pratique, elle peut s'avérer très ennuyeuse. Nous l'avons constaté avec les serveurs Sunfire. Ces serveurs hébergent des espaces projets (ils sont nombreux) demandés par nos utilisateurs. Dans l'idée d'en simplifier l'administration, nous avons pris l'habitude de créer un système de fichiers (ZFS) par espace. Ces systèmes de fichiers sont tous imbriqués au sein d'un système de fichiers parent. Si cette pratique nous a satisfait tout au long de la vie des serveurs, la migration de ces machines a été rendue très compliquée par la présence de ces nombreux systèmes de fichiers. Nous avions initialement pensé exporter le système de fichiers racine pour migrer les données depuis celui-ci ; malheureusement, ceci n'est pas possible avec NFSv3. En effet, NFS dans sa version 3 exporte un système de fichiers à la fois : le serveur NFS ne traverse pas les frontières du système de fichiers exporté. Ainsi, l'export initial ne contenait que des répertoires vides, ceux-ci correspondant aux points de montages des systèmes de fichiers enfants sur le serveur ! Il nous a finalement fallu exporter un par un les systèmes de fichiers de la machine, en prenant soin de ne pas en oublier. Ici, la mise en place d'un serveur **rsync** (qui, lui, aurait pu gérer au sein d'un seul export la totalité des systèmes de fichiers) aurait pu nous éviter ce travail d'export fastidieux, mais ceci aurait été au détriment de l'utilisation de **fpart** et nous aurait obligé à repasser à une synchronisation unique.

Nous aurions certainement pu mettre au point une solution hybride : parcours initial des fichiers via NFS puis synchronisation depuis les nœuds via le protocole **rsync**, mais cette solution nous a semblé inutilement complexe et risquée : en multipliant les types d'accès, nous augmentons d'autant les risques d'erreurs d'accès aux données (par exemple à cause de noms de fichiers gérés avec des encodages différents entre NFS et **rsync**). De plus, ceci nous aurait juste évité d'effectuer

les montages NFS sur les nœuds du cluster : il aurait tout de même fallu mettre au point les exports et les monter sur la tête de soumission.

Pour permettre la migration, les exports sont faits en **no-root-squash**. Attention aux idées reçues : ce qui est risqué ici n'est pas de positionner cette option car un utilisateur malveillant pourra, dans tous les cas et s'il a accès au compte *root*, prendre l'identité de sa victime pour accéder à ses données ; non, le vrai risque réside dans la mise à disposition elle-même des données sur le cluster. Nous avons limité les risques en dédiant des machines du cluster à la migration (elles ne sont plus accessibles aux autres utilisateurs). Il convient également de s'assurer qu'elles sont bien à jour et de maîtriser la liste des personnes disposant du droit de passer *root*.

fpart fait de son mieux pour générer des partitions correspondant aux critères fournis par l'utilisateur. En mode *live*, l'algorithme est simpliste : **fpart** parcourt le système de fichiers et clôt la partition une fois le seuil fixé par l'utilisateur atteint. Par « atteint », nous entendons en fait « dépassé » car **fpart** doit être en mesure de placer tout fichier dans une partition, y compris ceux qui dépassent allègrement les critères spécifiés. Si **fpart** est exécuté avec une taille limite par partition de 100 Mo et qu'un fichier d'1 Go est trouvé, ce fichier va clore la partition en cours (ou même en constituer une à lui tout seul si par hasard ce fichier était le premier d'une nouvelle partition). On voit bien dans ce cas que les tailles des partitions peuvent être très variables. En mode *live* les partitions sont donc « à peu près égales » et certains cas dégénérés peuvent déclencher des synchronisations beaucoup plus longues que la moyenne.

Un point à noter est que nous perdons dans notre synchronisation les attributs étendus et les ACLs. Nous n'avions pas besoin de les migrer mais il doit être assez aisé d'adapter la méthode pour les prendre en compte ou bien même les répliquer *a posteriori*.

Une des pistes d'amélioration de **fpart** serait de rendre *multi-thread* le parcours du système de fichiers. Actuellement, celui-ci est linéaire et repose sur *fts(3)*. Il doit probablement y avoir moyen de gagner du temps à cette étape en s'insipirant d'un projet comme **robinhood**[11] et de résultats comme ceux exposés dans l'article « On Distributed File Tree Walk of Parallel File Systems »[12] de Jharrod Lafon du LANL.

À nos yeux, la puissance de **fpart** réside dans son mécanisme de hooks. C'est lui qui nous a permis d'exploiter de façon transparente l'infrastructure du cluster de calcul. Vous n'aurez sans doute pas trop de mal à les adapter à la vôtre. Néanmoins, dans le cadre très spécifique de la migration de données, un compagnon générique à **fpart** (GUI, wrapper...) pourrait trouver un plus large public, par exemple afin d'être plus évident à utiliser si l'on n'a pas déjà un gestionnaire de tâches qui pilote un cluster de machines. Pour l'instant, chacun doit écrire une glue spécifique. Rien de complexe, mais c'est plutôt fastidieux comme vous avez pu vous en rendre compte avec l'extrait de script shell exposé plus haut. Pourquoi ne pas aller dans le sens de **dcp**[13] (dont les limites sont évoquées dans l'article de Jeff Layton) qui implémente une file d'attente distribuée via la **libcircle**[14] ? Si l'on souhaite rendre la migration de données aussi simple qu'un **cp**, cela semble être un axe de développement des plus intéressants.

Conclusion

La méthode présentée dans cet article nous satisfait pleinement. En pratique, nous pouvons lancer notre migration sur une arborescence arbitrairement grande en pleine confiance. L'impact sur les utilisateurs est minimal. Toutefois, il faut garder à l'esprit que la migration des données n'est pas la seule tâche à accomplir pour migrer des serveurs de fichiers. Le transfert des données elles-mêmes est une grande partie du travail, mais il faut également répliquer toutes les meta-données relatives aux espaces migrés, notamment :

- les quotas,
- les politiques de snapshots et de sauvegarde,
- les partages NFS,
- les partages CIFS (en revoyant éventuellement les noms des partages pour éviter les collisions car on passe de plusieurs espaces de noms - plusieurs serveurs - à un seul),
- la configuration des clients (adapter les chemins des partages à monter).

Enfin, selon les cas, il se peut qu'il faille convertir les noms des fichiers eux-mêmes (si les encodages source et destination sont différents). **convmv**[15] peut s'avérer très utile dans ce cas.

Malheureusement, ces tâches sont fastidieuses et difficilement automatisables ; nous les avons traitées à la main et au coup par coup, selon les espaces.

Merci aux différents relecteurs qui se reconnaîtront et bonne migration à tous ! ■

[11] http://robinhood.sf.net

[12] http://www.cs.nmsu.edu/~misra/papers/sc12paper.pdf

[13] http://filecopy.org/

[14] https://github.com/hpc/libcircle

[15] https://www.j3e.de/linux/convmv/man

POURQUOI PHP EST-IL UN MEILLEUR LANGAGE QUE PYTHON ?

par Dr Kiss Cool – Attention au deuxième effet...

N'en déplaise à nos amis pythonistes, PHP est un langage infiniment supérieur à Python par bien des points. Je vais tenter par cet article de convertir quelques brebis égarées et de les ramener dans le droit chemin. Abandonnez Python au profit de PHP !

Pour éviter toute critique sournoise de quelque pythoniste belliqueux, voici les versions des langages utilisés dans cet article : Python 3.2.3 et PHP 5.3.10. PHP sera utilisé en CLI (Command Line interface) en activant l'option **-a** (ou **--interactive**) donc en ligne de commande interactive et Python sera également utilisé en mode interactif.

Nous allons prendre un certain nombre de points « forts » de Python et montrer que PHP peut faire beaucoup mieux !

1 Python est un langage simple à apprendre

« Python a été créé dans l'optique d'être le plus simple possible à apprendre : il est simple à lire, à écrire et à comprendre. »

Python est effectivement un langage qui peut être appris rapidement... mais que dire alors de PHP ? Je ne connais pas d'autre langage comportant autant de développeurs éclairés, non informaticiens ayant appris seuls et réussissant à faire des programmes qui fonctionnent (car après tout, c'est bien la seule chose qui compte, non?). Un nombre incalculable de designers et de webmasters se disent développeurs PHP... c'est bien qu'il s'agit d'un langage simple à apprendre ! En fait tout le monde sait écrire du PHP : il suffit de lire quelques lignes code pour comprendre comment il fonctionne. En plus, comme j'ai pu le lire récemment sur le blog d'un développeur PHP : « les mathématiques ne servent à rien en informatique ». On peut donc faire du PHP sans rien comprendre en mathématique ! Nous comparons PHP à Python mais seriez-vous capable de citer un langage informatique qui ne requiert pas un minimum de bagage mathématique ?

Les pythonistes commencent à baisser la tête, mais ça ne fait que commencer...

2 Python permet de faire des opérations complexes

« Les modules scientifiques de Python permettent de réaliser rapidement des calculs très complexes, d'ailleurs les grands organismes de recherche utilisent de plus en plus Python. »

Nous sommes tout prêts à le croire mais nous allons quand même effectuer un petit test. Si nous prenons trois petits cochons et que nous en ajoutons un, combien obtenons-nous de petits cochons ? Un enfant d'école maternelle pourrait nous répondre, cela fait quatre. Que nous dit Python ?

```
>>> print("3 petits cochons" + 1 )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str
implicitly
```

Python peut faire des calculs très « complexes » mais là il a l'air un peu perdu...Et en PHP ?

```
php > print("3 petits cochons" + 1);
4
```

Ah ! Là où Python est complètement perdu, PHP, lui, nous donne le bon résultat ! J'ai trois cochons, j'en ajoute un, j'obtiens quatre cochons. Laissons encore une chance à Python qui nous indiquait dans son message d'erreur qu'il attendait une chaîne de caractères en paramètre. Ajoutons maintenant trois petits cochons et deux gros poulets. Le résultat est évident : cinq animaux. Voici ce que nous répond Python :

```
>>> print("3 petits cochons" + "2 gros poulets")
3 petits cochons2 gros poulets
```

C'est confirmé : Python ne parvient pas au bon résultat et essaye vainement

de nous renvoyer un résultat en concaténant les deux chaînes de caractères... Si nous testons avec PHP, vous vous doutez déjà qu'il n'y aura aucun problème :

```
php > print("3 petits cochons" + "2 gros poulets");
5
```

PHP et Python sont deux langages à typage dynamique : lorsque l'on définit une variable, nous n'avons pas à spécifier le type de donnée qu'elle contiendra (entier, chaîne de caractères, etc.).

PHP est un langage à typage dynamique dit faiblement typé alors que Python est dit fortement typé : le type d'une variable est fixé en fonction du contexte dans lequel elle est utilisée et des méthodes que l'on peut lui appliquer. Donc si l'on tente d'effectuer une addition entre une chaîne de caractères et un entier on obtient un message d'erreur. PHP, lui, force l'interprétation numérique de la chaîne de caractères pour obtenir un résultat à l'addition.

Ce mécanisme montre non seulement que PHP est supérieur à Python mais qu'il est même bien meilleur que la théorie mathématique qui nous dit qu'on ne peut pas ajouter des torchons et des serviettes : les mathématiques ne le peuvent pas, PHP si !

3 Python est un langage qui évolue

Python a rompu la compatibilité ascendante avec la sortie de Python 3. Ainsi, un code écrit en Python 2 ne peut plus être exécuté par Python 3. Pourquoi ? Pour pouvoir reprendre en profondeur le cœur du langage, corriger des erreurs de jeunesse... PHP est pur, PHP est bon : il n'y a pas d'erreur. C'est pour cela que la version 6 de PHP ne verra pas le jour (pourtant programmée mais abandonnée) : PHP a été bien conçu dès le départ ! Il n'y a donc que quelques améliorations mineures à lui ajouter de temps en temps.

4 Python est concis

La compréhension de liste (fonctionnant également avec les dictionnaires et d'une manière plus spécifique avec les tuples), qui n'est effectivement pas disponible en PHP, permet d'écrire des opérations de manière claire (à condition d'avoir un minimum de pratique) et concise :

```
>>> liste = { x:abs(x) for x in range(-5, 6) }
>>> for key, value in liste.items():
...   print("{} => {}".format(key, value))
...
0 => 0
1 => 1
2 => 2
3 => 3
4 => 4
5 => 5
-2 => 2
-5 => 5
-4 => 4
-3 => 3
-1 => 1
```

Nous avons créé un dictionnaire dans lequel on associe à chaque clé comprise entre **-5** et **5** sa valeur absolue. Remarquons au passage l'ordre incongru dans lequel les données nous sont présentées lors de l'affichage...

Mais PHP aussi peut être très concis... il suffit de tout écrire sur la même ligne ! Ici, non seulement nous définissons le dictionnaire (en PHP on dit « tableau associatif »), mais en plus nous l'affichons sur la même ligne !

```
php > $liste = array(); for ($i = -5; $i <= 5;
$i++) { $liste[(string) $i] = abs($i); } ; foreach
($liste as $key => $value) { printf("%d => %d\n",
$key, $value); }
-5 => 5
-4 => 4
-3 => 3
-2 => 2
-1 => 1
0 => 0
1 => 1
2 => 2
3 => 3
4 => 4
5 => 5
```

Oh surprise ! PHP est capable de nous renvoyer les résultats dans le bon ordre ! Encore une preuve de l'efficacité de PHP face à Python ! Bien sûr les mauvaises langues pourront dire que par essence un tableau associatif est une structure non ordonnée puisque stockée normalement sous forme d'arbre équilibré ou de table de hachage.

En terme d'efficacité, si nous effectuons un petit benchmark rapide et imprécis, nous ne manquerons pas de montrer que PHP est bien plus rapide. Pour créer un tableau associatif de vingt millions d'éléments sans utiliser la fonction **abs()** pour ne pas fausser les résultats, nous obtenons :

```
php > $start = time(); $liste = array(); for
($i = -10000000; $i <= 10000000; $i++) {
$liste[(string) $i] = $i; }; $end = time(); print
$end - $start;
6
```

Il faut donc 6s pour créer ce tableau. Même test avec Python sans la compréhension de dictionnaire, toujours pour essayer de comparer ce qui est comparable (notez le nombre de lignes !) :

```
>>> import time
>>> def f():
...   start = time.time()
...   liste = {}
...   for x in range(-10000000, 10000001):
...     liste[x] = x
...   end = time.time()
...   print(end - start)
...   return liste
...
>>> liste = f()
1.3649539947509766
```

Donc non seulement Python nous renvoie un résultat non ordonné mais en plus il effectue ses opérations en 1.36s ! Cela fait plus de 4s de différence entre PHP et Python ! Ah... mon rédacteur en chef me signale que je serais en train de me fourvoyer... Python serait en fait plus rapide que PHP. Balivernes ! Denis est à la solde du Grand Serpent. Honte à lui ! « Les chiffres sont là » me dit-il... et si Python s'était encore fourvoyé ? Qui nous dit que la fonction **time()** renvoie un résultat correct ? Ah, voilà un bon argument ! Peut-être faut-il lire 13.6s ou même 136.4s si la virgule est mal placée...

Ce regrettable incident étant réglé, essayons de poursuivre et de ne pas nous faire censurer par les adorateurs

du Grand Serpent... car après tout, peut-être sont-ils présents en nombre au sein de la rédaction ? Ils sont partout, méfiez-vous !

5 Python dispose d'un mode interactif

Il est facile de tester rapidement quelques lignes de code en Python grâce à la console interactive, c'est indéniable. Il suffit d'exécuter python et on aboutit au prompt permettant de saisir du code.

Mais en PHP ce mode existe également : c'est celui que nous utilisons depuis le début de cet article. Il suffit d'appeler la commande PHP avec l'option **-a** ou **--interactive**.

6 Python est plus lisible

Il est soit disant plus simple de lire du Python, le langage étant plus structuré. Comparons donc les deux langages avec l'affichage des nombres pairs présents dans une liste (tableau en PHP) de onze éléments de 0 à 10 :

```
>>> tab = list(range(11))
>>> i = 0
>>> while (i < len(tab)):
...   if tab[i] % 2 == 0:
...     print("{} est pair".format(tab[i]))
...   else:
...     print("{} est impair".format(tab[i]))
...   i += 1
...
0 est pair
1 est impair
...
8 est pair
9 est impair
10 est pair
```

La même chose en PHP :

```
php > $tab = array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
php > $i = 0;
php > while ($i < count($tab)) { ($i % 2 == 0) ? printf("%d est
pair\n", $tab[$i++]) : printf("%d est impair\n", $tab[$i++]); }
0 est pair
1 est impair
...
8 est pair
9 est impair
10 est pair
```

Alors, ce n'est pas plus clair ? Nous aurions certes pu écrire le code PHP différemment mais ici nous pouvons montrer à quel point les opérateurs ternaires et d'incrémentation font défaut à Python. En PHP inutile d'ajouter des tonnes d'espaces et de tabulations inutiles, on est libre d'écrire tout son code sur une seule ligne et l'on dispose de la meilleure instruction de structuration au monde : le **goto** ! L'exemple ne peut pas être lancé en mode interactif mais voici ce que l'on peut écrire en PHP et qu'il est impossible de faire en Python :

```
<?php
  goto label;
  print 'Pas d\'affichage';

  label:
  print 'Affichage';
```

Vous reconnaîtrez qu'il est plus aisé de lire le code PHP. Et puis franchement, ça ne fait pas plus code d'informaticien à côté de ce petit code Python qu'un élève de maternelle aurait pu écrire ?

7 Pear versus Pip

En Python on utilise souvent Pip pour installer de nouveaux modules (l'équivalent de bibliothèques PHP) depuis PyPi (PYthon Package Index). Le programme gère entièrement les dépendances et, allié à un environnement virtuel, il permet de déterminer les dépendances d'un projet. À l'heure où ces lignes sont écrites, 32523 paquetages sont disponibles.

L'équivalent en PHP s'appelle Pear : il fait la même chose (mais forcément en mieux) et propose 595 paquetages. Certes, il y a moins de paquetages, inutile d'attendre une quelconque remarque du laquais du Grand Serpent qui nous sert de rédacteur en chef. Mais s'il y a moins de choix, c'est qu'il y a eu une sélection minutieuse : il s'agit du dessus du panier du gratin de l'élite des paquetages ! Et si effectivement PHP ne propose pas d'environnement virtuel, le projet **phpenv** est en train de remédier à cela en empruntant des voix peu recommandables, s'inspirant fortement de **virtualenv** et donc de Python.

8 L'hébergement mutualisé

Voilà un domaine où le Grand Serpent n'a pas encore pu étendre son influence. Pourtant aidé par son acolyte le petit poney ailé ô combien ridicule, ils auraient toutes les chances d'étendre la domination de Python.... mais non ! Un petit groupe de décideurs résiste encore et toujours à l'envahisseur. Pas de Django sur les beaux serveurs mutualisés ! PHP peut régner en maître... mais pour combien de temps encore ?

Conclusion

Mes frères, rejoignez la communauté PHP ! Ne vous laissez plus abuser par le Grand Serpent qui va jusqu'à pervertir vos beaux GNU/Linux Magazine et GNU/Linux Pratique avec des hors-séries à sa gloire ! Aidez-moi à sauver la rédaction de son influence néfaste !

Toutefois n'oubliez pas :

« il n'y a pas de mauvais langage... il n'y a que de mauvais développeurs ». ■

À DÉCOUVRIR ACTUELLEMENT CHEZ VOTRE MARCHAND DE JOURNAUX !

NOTRE NOUVEAU GUIDE !

SERVEURS

LE GUIDE POUR CRÉER ET GÉRER VOS SERVICES À LA CARTE !

GNU/LINUX MAGAZINE HORS-SÉRIE N° 68

ÉGALEMENT DISPONIBLE SUR NOTRE BOUTIQUE EN LIGNE :

boutique.ed-diamond.com

MIMEDEFANG

par Sébastien Namèche

Historique

À l'origine, MIMEDefant est un développement de David F. Skoll financé par le Collège royal des médecins et chirurgiens du Canada en 2000 afin de les aider à endiguer le déluge de virus propagés par messagerie électronique sur le réseau informatique du collège. Le logiciel fut conçu pour filtrer les pièces jointes et s'est tout d'abord nommé MIMESweeper, puis MIMEDefanger et, finalement, MIMEDefang. David Skoll annonça la naissance de son programme au public le 28 août 2000. Le 21 décembre 2001, une version incorporant le support de SpamAssassin est publiée, permettant ainsi à MIMEDefang de filtrer à la fois les virus et les spams. La société de David Skoll, Roaring Penguin Software, commercialise un produit antispam appelé CanIt qui est construit au dessus de la version libre de MIMEDefang.

1 Introduction

1.1 Révisions scolaires

Avant de rentrer dans le vif du sujet, il nous faut passer par une phase de révisions barbares mais néanmoins nécessaires. Si vous connaissez le fonctionnement du protocole SMTP, prenez quand même le temps de lire cette section. En effet, certaines subtilités du protocole souvent méconnues prennent toute leur importance lorsqu'il s'agit de le travailler au corps.

1.1.1 Dialogue SMTP

La figure 1 présente les différentes phases d'un dialogue SMTP pour transmettre un message d'un MTA (*Mail Transfer Agent*) à un autre. Ces différentes phases sont : (voir Figure 1).

1. Le client se connecte au serveur sur le port 25.
2. Le serveur envoie la bannière SMTP qui commence par le code 220 (serveur prêt) suivie du nom canonique du serveur. Si le troisième mot de cette bannière est ESMTP, alors le serveur supporte la version étendue du protocole SMTP (ce qui devrait être maintenant le cas de tout serveur SMTP).
3. Le client envoie la commande **HELO** ou **EHLO** si le client supporte lui aussi la version étendue du protocole.
4. Le serveur répond en se présentant et, si le client a utilisé la commande **ESMTP**, il énumère les fonctions qu'il supporte. Chaque ligne commence par le code 250 (action demandée réalisée avec succès) suivi d'un tiret, à l'exception de la dernière.
5. Le client utilise alors la commande **MAIL FROM** suivie de l'adresse email de l'expéditeur pour signifier au serveur qu'il va lui transmettre un nouveau message.
6. Le serveur vérifie si l'adresse de l'expéditeur lui semble valide et l'acquitte avec, encore une fois, le code 250.
7. Le client indique au serveur les destinataires du message en utilisant autant de commandes **RCPT TO** que nécessaire.
8. Pour chaque destinataire, le serveur vérifie que l'adresse est valide à son sens et l'acquitte avec le sempiternel code 250 si c'est le cas. Si le serveur considère que l'une des adresses de destination est erronée, il répond avec un code dont le premier chiffre est 5.
9. Si au moins l'un des destinataires n'a pas été refusé, le client peut continuer la conversation en envoyant la commande **DATA** pour demander au serveur l'autorisation de lui envoyer le contenu du message.

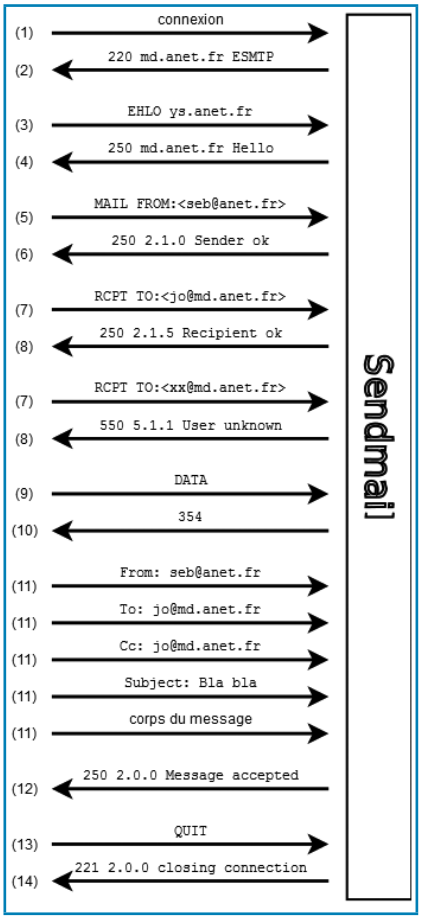


Figure 1 : Dialogue SMTP entre deux MTA

Je pompe cette histoire de MIMEDefang directement depuis la version anglaise de Wikipédia. Pour me faire pardonner ma paresse, j'ai traduit le (modeste) article sur MIMEDefang dans l'encyclopédie en ligne vers sa version française.

10. Si le serveur n'y voit pas d'inconvénient, il autorise le client à le faire en lui retournant le code 354 (autorisation pour envoyer le message).
11. Le client transmet alors le message qui est composé de deux parties : les en-têtes et le corps. Les en-têtes viennent en premier et sont séparés du corps par une ligne vide. Le message est terminé par un point seul sur une ligne.
12. Si le serveur est d'accord pour prendre en charge le message, il répond avec le code 250, c'est à cette seule condition que la responsabilité de la bonne gestion du message est transférée du MTA client au MTA serveur.
13. Le client peut alors choisir de transmettre un nouveau message en recommençant tout le cycle à partir de la commande **`MAIL FROM`**. Sinon, il met fin à la conversation avec la commande **`QUIT`**.
14. Dans ce dernier cas, le serveur met poliment fin à la conversation en répondant, une fois n'est pas coutume, avec le code 221 (le serveur va fermer le canal de transmission) puis clôt la connexion TCP.

Swaks

Si vous ne connaissez pas l'outil swaks, je vous recommande vivement de l'installer. Il vous deviendra vite indispensable en remplaçant avantageusement le traditionnel telnet smtp.

```
$ swaks -f seb@anet.fr -t jo@md.anet.fr,xx@md.anet.fr -s md.anet.fr -stl
    === Trying md.anet.fr:25...
 (1) === Connected to md.anet.fr.
    === response in 5.019s
 (2) <-  220 md.anet.fr ESMTP Welcome, please be polit or go away!
    === response in 0.000s
 (3)  -> EHLO ys.anet.fr
 (4) <-  250-md.anet.fr Hello ys.anet.fr [192.168.56.1], pleased to meet you
 (4) <-  250-ENHANCEDSTATUSCODES
 (4) <-  250-PIPELINING
 (4) <-  250-8BITMIME
 (4) <-  250-SIZE
 (4) <-  250-DSN
 (4) <-  250-AUTH DIGEST-MD5 CRAM-MD5
 (4) <-  250-STARTTLS
 (4) <-  250-DELIVERBY
 (4) <-  250 HELP
 (5)  -> MAIL FROM:<seb@anet.fr>
    === response in 0.009s
 (6) <-  250 2.1.0 <seb@anet.fr>... Sender ok
 (7)  -> RCPT TO:<jo@md.anet.fr>
    === response in 0.006s
 (6) <-  250 2.1.5 <jo@md.anet.fr>... Recipient ok
 (7)  -> RCPT TO:<xx@md.anet.fr>
    === response in 0.002s
 (8) <** 550 5.1.1 <xx@md.anet.fr>... User unknown
 (9)  -> DATA
    === response in 0.001s
(10) <-  354 Enter mail, end with "." on a line by itself
(11)  -> Date: Mon, 11 Mar 2013 16:01:32 +0100
(11)  -> To: jo@md.anet.fr,xx@md.anet.fr
(11)  -> From: seb@anet.fr
(11)  -> Subject: test Mon, 11 Mar 2013 16:01:32 +0100
(11)  -> X-Mailer: swaks v20100211.0 jetmore.org/john/code/swaks/
(11)  ->
(11)  -> This is a test mailing
(11)  ->
(11)  -> .
    === response in 0.105s
(12) <-  250 2.0.0 r2BHpxtW017025 Message accepted for delivery
(13)  -> QUIT
    === response in 0.005s
(14) <-  221 2.0.0 md.anet.fr closing connection
(14) === Connection closed with remote host.
```

1.1.2 Adresses d'enveloppe, adresses d'en-tête

Il est important de faire la distinction entre les adresses (de l'expéditeur et des destinataires) dans l'enveloppe du message et les adresses dans l'en-tête du message. Il y a souvent confusion à ce sujet.

L'enveloppe du message est constituée de toutes les données qui sont échangées entre le MTA client et le MTA serveur en dehors de la phase DATA. L'enveloppe ne fait pas partie du message tandis que l'en-tête constitue la première partie du message. Les adresses de l'expéditeur et des destinateurs peuvent être très différentes entre l'enveloppe et l'en-tête. Par exemple, lorsqu'un message est transmis par l'intermédiaire d'un gestionnaire de listes de diffusion. L'adresse de l'expéditeur reste celle d'origine dans l'en-tête alors que celle de l'enveloppe est remplacée par une adresse associée au robot qui gère la liste de diffusion. De même, l'adresse du destinataire dans l'en-tête reste celle de la liste de diffusion alors que, lors de la distribution aux abonnés de la liste, les adresses de l'enveloppe sont celles des abonnés.

Lorsqu'un message est reçu par le MUA (*Mail User Agent*, le logiciel de messagerie) d'un utilisateur, retrouver les informations relatives à l'enveloppe dans le message n'est pas trivial. Cependant, il est possible de les déduire à partir des lignes de l'en-tête dans le MUA. Le MDA (*Mail Delivery Agent*, l'agent utilisé par le MTA pour remettre le message à un destinataire local) ajoute généralement au message une ligne d'en-tête **`Return-Path`** qui contient l'adresse de l'expéditeur dans l'enveloppe. Quant à l'adresse du destinataire dans l'enveloppe, il est souvent possible de la déduire en observant les lignes d'en-tête **`Received`** qui sont ajoutées par chacun des MTA et par le MDA par lesquels le message transite. Par exemple, dans la ligne d'en-tête suivante, l'adresse **`sebastien.nameche@netensia.fr`** est l'adresse du destinataire dans l'enveloppe :

```
Received: from cabale.usenet-fr.net (cabale.usenet-fr.net [217.24.82.4]) by
o2.netensia.net (8.13.8/8.13.8/Debian-3+etch1) with ESMTP id r1SGPNYO000627
for <sebastien.nameche@netensia.fr>; Thu, 28 Feb 2013 16:25:33 GMT
```

Les adresses émail utilisées lors des phases **MAIL FROM** et **RCPT TO** sont les adresses de l'enveloppe. Ce sont ces adresses qui seront visibles par MIMEDefang.

1.2 L'API Milter

Milter (Mail fILTER) est à la fois un protocole et une bibliothèque. Elle a été conçue et mise au point par l'équipe qui développe Sendmail entre ses versions 8.10 et 8.12 et est désormais considérée comme mature. Son objectif est simple : permettre à des processus externes d'intervenir pendant la conversation SMTP. Le client Milter utilise l'API Milter pour s'enregistrer et déclarer des fonctions de rappel (*callbacks*) qui seront exécutées par Sendmail lors des différentes phases de la conversation SMTP. Le client Milter a alors la possibilité d'intervenir sur la réponse faite par Sendmail au client SMTP, voire, lors de la phase finale (DATA), de modifier le message avant qu'il ne soit retransmis par Sendmail.

L'intérêt majeur de l'API Milter est de permettre d'intervenir pendant la conversation SMTP elle-même. Ainsi, de nombreuses applications ont vu le jour. Par exemple, le programme Milter clamav-milter qui permet de passer le contenu de tous les messages à l'antivirus ClamAV.

L'idée ayant eu du succès, l'API Milter a été implémentée au sein de Postfix. Ainsi, un Milter (par extension, ce terme désigne une application cliente de l'API Milter) devrait en théorie pouvoir fonctionner avec Sendmail comme avec Postix, ce qui est le cas de MIMEDefang. Dans cet article, je ne présente que la configuration de Sendmail pour le Milter MIMEDefang car c'est le MTA que je connais le mieux. Les fans de Postfix trouveront sans peine de la documentation sur la configuration d'un Milter pour leur MTA préféré.

L'API Milter présente cependant quelques difficultés. Tout d'abord, le client doit être écrit dans un langage compilé et les binaires doivent être liés avec la bibliothèque Milter. La plupart du temps, le langage C est employé. Et bien d'autres langages savent manipuler avec bien plus d'aisance les chaînes de caractères et disposent de bibliothèques spécialisées pour le traitement des messages électroniques et la réalisation des tâches qui y sont associées. De plus, Sendmail est un programme multi-processus et la bibliothèque Milter peut créer autant de fils d'exécution que cela est nécessaire. Il faut donc pouvoir gérer la concurrence des processus et fils d'exécution correctement avec un langage qui n'est pas fait pour cela à l'origine. Pour corser le tout, les processus Sendmail peuvent exécuter une fonction de rappel d'un fil d'exécution du Milter lors d'une phase SMTP et une autre fonction de rappel d'un autre fil du même Milter d'exécution un peu plus tard, lors d'une phase SMTP postérieure. Il n'y a aucune persistance entre les processus Sendmail et les fils d'exécution Milter.

C'est pour toutes ces raisons que MIMEDefang a été créé. L'objectif du projet est de masquer la complexité de la bibliothèque Milter et de fournir un environnement d'exécution et une boîte à outils pour manipuler les messages électroniques dans un langage disposant de sérieux atouts pour cela : le Perl.

1.3 La théorie de la patate chaude

Mais quel est l'intérêt de mettre en branle toute cette machinerie qui semble de prime abord plus complexe que des logiciels qui savent filtrer les messages électroniques tels que Procmail, Amavis ou MailScanner ?

Il y a quelques années, j'ai donné la réponse à cette question très pertinente dans un article toujours disponible sur le site du CLX (http://clx.asso.fr/spip/Lutter-efficacement-contre-les) et que je résume ici. Une patate chaude est un message électronique dont nous avons accepté la prise en charge et dont nous souhaitons nous débarrasser pour une raison ou une autre (il contient un virus, il a été identifié comme un message non sollicité, le destinataire n'existe pas, etc.). Mais que faire de ce message ? Le retourner à l'expéditeur ? Bien souvent, les adresses d'expédition des messages malveillants sont forgées ou usurpées. Renvoyer le message à l'administrateur ? Pitié pour lui ! Le mettre en quarantaine ? Franchement, qui d'entre vous consulte régulièrement sa quarantaine ? Le détruire ? C'est une lourde responsabilité.

Toutes ces questions se posent parce que nous avons accepté la responsabilité du message. J'ai un principe fétiche qui s'applique merveilleusement bien à l'administration des systèmes et réseaux : si aucune réponse n'est satisfaisante, la question posée est sans doute la mauvaise. La question la plus intéressante dans cette situation semblerait plutôt être : pourquoi avons-nous accepté de prendre en charge ce message indésirable ? Parce que les outils dont nous disposions (Procmail, Amavis, MailScanner, etc.) ne nous permettaient pas de faire autrement. Dans l'idéal, nous ne devrions accepter aucun message que nous ne soyons certains de pouvoir transmettre par la suite. Il nous faut donc refuser de prendre en charge un message qui ne nous plaît pas pendant la conversation SMTP avec le MTA qui essaye de nous fourguer sa camelote.

Un second point intéressant à cette manière de faire, c'est que nous économisons ainsi nos ressources. Nous laissons la charge au relais SMTP qui laisse passer (sciemment ou non) les messages indésirables de gérer les retours en erreur. De plus, plus vite nous filtrerons les messages (plus tôt dans la conversation SMTP) plus nous économiserons notre bande passante. Si nous pouvons rejeter une connexion avant la phase DATA, alors nous économisons toute la transmission du contenu du message. Dans un contexte où 95% environ des messages électroniques qui transitent sur Internet sont indésirables, cela peut représenter une quantité de bande passante économisée qui n'est pas négligeable.

Complétez votre collection d'anciens numéros !

boutique.ed-diamond.com

numerique.ed-diamond.com

Enfin, comme nous le verrons par la suite, MIMEDefang nous fournit une boîte à outils qui nous permettra de manipuler les messages à la volée : ajouter ou modifier la liste des destinataires, supprimer ou remplacer certains pièces jointes par un avertissement ou un lien, ajouter un texte en fin de chaque message, etc.

2 Installation

La figure 2 présente l'architecture de MIMEDefang. Le programme **mimedefang** est écrit en C et est lié avec la bibliothèque **libmilter**. C'est lui qui gère la communication avec les différents processus de Sendmail et qui exécute plusieurs fils d'exécution en parallèle. Le processus **mimedefang-multiplexor** est également développé en C. Il gère la distribution des tâches vers les différents processus esclaves **mimedefang.pl**, crée de nouveaux esclaves lorsque le besoin s'en fait sentir ou les détruit lorsque la charge diminue. Enfin, le programme **mimedefang.pl** est écrit en Perl et est exécuté en de multiples processus contrôlés par **mimedefang-multiplexor**. **mimedefang.pl** met en place un environnement d'exécution, fournit des outils et gère les requêtes en provenance du Milter en appelant des fonctions définies dans le fichier **/etc/mail/mimedefang-filter**. Ce fichier est écrit en Perl et définit la politique de filtrage des messages électroniques.

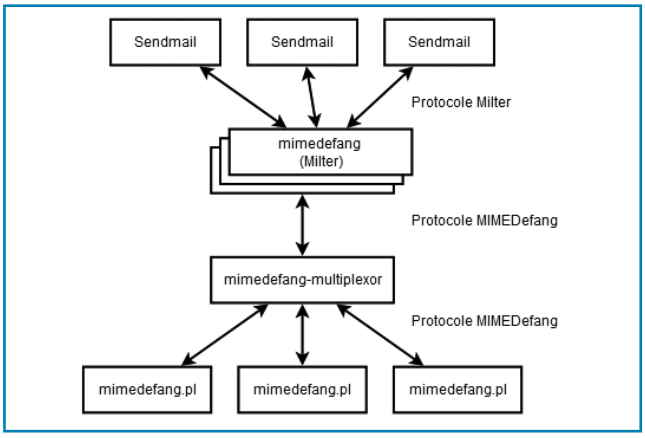


Figure 2 : Architecture de MIMEDefang

mimedefang-multiplexor a une option qui, s'il est compilé avec la bibliothèque **libperl**, permet d'exécuter le programme **mimedefang.pl** dans un interpréteur Perl embarqué. Cela améliore les performances lors du démarrage des nouveaux processus **mimedefang.pl** car un appel à **execve** est économisé et les séquences d'initialisation de Perl et de **mimedefang.pl** ne sont exécutées qu'une seule fois.

2.1 Paquets Debian

Deux paquets sont inclus dans la distribution Debian Squeeze : **mimedefang**, qui dépend essentiellement de bibliothèques Perl et contient l'environnement d'exécution de base de MIMEDefang, et le paquet optionnel **graphdefang**, utilisé pour réaliser de beaux graphiques à partir de statistiques collectées par MIMEDefang.

Les principales dépendances optionnelles (suggérées) du paquet **mimedefang** sont l'antivirus ClamAV, le logiciel de filtrage des messages indésirables SpamAssassin et Tcl/Tk. Cette dernière dépendance permet d'exécuter le programme **watch-mimedefang** qui permet de visualiser et contrôler en temps réel l'activité des processus de MIMEDefang. Bien évidemment, il vous faudra également installer Sendmail ou Postfix.

La ligne de commandes suivante installera MIMEDefang, ClamAV, SpamAssassin ainsi que quelques paquets optionnels qui permettront d'utiliser des fonctionnalités supplémentaires dans MIMEDefang :

```
$ sudo aptitude install clamav libmail-spamassassin-perl mimedefang
libarchive-zip-perl libhtml-parser-perl
```

Les fichiers de configuration de MIMEDefang sont alors :

- **/etc/mail/mimedefang.pl.conf** : configuration des paramètres pour les processus esclaves **mimedefang.pl** ;
- **/etc/mail/mimedefang-filter** : script Perl qui définit la politique de filtrage ;
- **/etc/mail/sa-mimedefang.conf** : configuration de SpamAssassin pour MIMEDefang ;
- **/etc/default/mimedefang** : paramètres d'exécution pour le Milter **mimedefang** et le multiplexeur **mimedefang-mutiplexor**.

L'utilisateur **defang** doit être membre du groupe **clamav** et l'utilisateur **clamav** membre du groupe **defang** afin que le démon **clamd** puisse accéder aux fichiers temporaires créés par MIMEDefang pour y rechercher des virus :

```
$ sudo adduser defang clamav
$ sudo adduser clamav defang
$ sudo /etc/init.d/clamav-daemon restart
$ sudo /etc/init.d/mimedefang restart
```

2.2 Installation manuelle

La version de MIMEDefang installée avec les paquets Debian Squeeze est la version 2.69. À l'heure où j'écris ces lignes, la dernière version stable est la 2.73. Des *bugs* ont été corrigés entre ces deux versions, notamment la version 2.70 qui corrige un bug avec la version embarquée de Perl et la version 2.71 qui restaure la compatibilité avec Postfix. Si vous avez besoin de l'une de ces deux corrections, vous devrez installer MIMEDefang à la main.

Commençons par installer Sendmail et les outils nécessaires à sa configuration :

```
$ sudo aptitude install sendmail m4 make
```

Maintenant, les dépendances directes pour MIMEDefang :

```
$ sudo aptitude install libmime-tools-perl libmailtools-perl
libdigest-sha1-perl libmilter1.0.1 libunix-syslog-perl
```

Puis les dépendances optionnelles :

```
$ sudo aptitude install libmail-spamassassin-perl clamav-daemon
```

Et les dépendances nécessaires à la compilation qui pourront être supprimées une fois MIMEDefang installé :

```
$ sudo aptitude install gcc libmilter-dev perl-dev libperl-dev
libclamav-dev
```

Il faut ensuite créer un utilisateur et un groupe pour MIMEDefang. L'utilisateur **defang** doit être membre du groupe **clamav** et l'utilisateur **clamav** membre du groupe **defang** afin que le démon **clamd** puisse accéder aux fichiers temporaires créés par MIMEDefang pour y rechercher des virus.

```
$ sudo groupadd -r defang
$ sudo useradd -c MIMEDefang -d /var/spool/MIMEDefang -g defang -r -s
/bin/false defang
$ sudo adduser defang clamav
$ sudo adduser clamav defang
```

C'est maintenant le moment de télécharger la dernière version de MIMEDefang et de le compiler. L'exécution de la commande **configure** nous permet de nous faire une idée de l'ensemble des antivirus supportés par MIMEDefang :

```
$ wget http://www.mimedefang.org/static/mimedefang-2.73.tar.gz
$ tar xzf mimedefang-2.73.tar.gz
$ cd mimedefang-2.73
$ ./configure
.../...
*** Virus scanner detection results:
H+BEDV   'antivir'   NO (not found)
Vexira   'vascan'    NO (not found)
NAI      'uvscan'    NO (not found)
BDC      'bdc'       NO (not found)
Sophos   'sweep'     NO (not found)
Sophos   'savscan'   NO (not found)
TREND    'vscan'     NO (not found)
CLAMSCAN 'clamav'    YES - /usr/bin/clamscan
AVP      'AvpLinux'  NO (not found)
AVP5     'aveclient' NO (not found)
KAVSCANNER 'kavscanner' NO (not found)
CSAV     'csav'      NO (not found)
FSAV     'fsav'      NO (not found)
FPROT    'f-prot'    NO (not found)
FPSCAN   'fpscan'    NO (not found)
SOPHIE   'sophie'    NO (not found)
NVCC     'nvcc'      NO (not found)
CLAMD    'clamd'     YES - /usr/sbin/clamd
TROPHIE  'trophie'   NO (not found)
NOD32    'nod32cli'  NO (not found)

Make sure clamd runs as the defang user!
  ...and make sure you use clamd version 0.67 or higher.

Found Mail::SpamAssassin.  You may use spam_assassin_* functions
Found HTML::Parser.  You may use append_html_boilerplate()

Note: SpamAssassin and HTML::Parser are
detected at run-time, so if you install or remove any of those
modules, you
do not need to re-run ./configure and make a new mimedefang.pl.
```

Puis, compilation, installation et copie de quelques fichiers de configuration :

```
$ make
.../...
$ sudo make install
.../...
$ sudo cp examples/init-script /etc/init.d/mimedefang
$ sudo update-rc.d defaults mimedefang
$ sudo cp examples/suggested-minimum-filter-for-windows-clients /etc/
mail/mimedefang-filter
```

Enfin, créer le fichier **sudo vi /etc/mail/mimedefang.conf** avec le contenu suivant (nous reviendrons sur les lignes en commentaires par la suite) :

```
# MX_RELAY_CHECK=no
# MX_HELO_CHECK=no
# MX_SENDER_CHECK=no
# MX_RECIPIENT_CHECK=no
MD_ALLOW_GROUP_ACCESS=yes
# MX_EMBED_PERL=yes
```

La commande **mimedefang.pl -features** permet d'afficher les bibliothèques découvertes et les fonctionnalités activées par MIMEDefang :

```
$ mimedefang.pl -features
MIMEDefang version 2.73

HTML::Parser                : yes
Path:CONFDIR                : yes (/etc/mail)
Path:QUARANTINEDIR          : yes (/var/spool/MD-Quarantine)
Path:SENDMAIL               : yes (/usr/sbin/sendmail)
Path:SPOOLDIR               : yes (/var/spool/MIMEDefang)
Virus:CLAMAV                : yes (/usr/bin/clamscan)
Virus:CLAMD                 : yes (/usr/sbin/clamd)
Archive::Zip                : no
Net::DNS                    : no
SpamAssassin                : no
Virus:AVP                   : no
Virus:AVP5                  : no
Virus:BDC                   : no
Virus:CSAV                  : no
Virus:FPROT                 : no
Virus:FPROTD                : no
Virus:FPROTD6               : no
Virus:FPSCAN                : no
Virus:FSAV                  : no
Virus:HBEDV                 : no
Virus:KAVSCANNER            : no
Virus:NAI                   : no
Virus:NOD32                 : no
Virus:NVCC                  : no
```

```
Virus:SAVSCAN               : no
Virus:SOPHIE                : no
Virus:SOPHOS                : no
Virus:SymantecCSS           : no
Virus:TREND                 : no
Virus:TROPHIE               : no
Virus:VEXIRA                : no

Archive::Zip                : missing
Digest::SHA1                : Version 2.13
HTML::Parser                : Version 3.66
IO::Socket                  : Version 1.31
IO::Stringy                 : Version 2.110
MIME::Base64                : Version 3.08
MIME::Tools                 : Version 5.428
MIME::Words                 : Version 5.428
Mail::Mailer                : Version 2.06
Mail::SpamAssassin          : Version 3.003001
Net::DNS                    : missing
Unix::Syslog                : missing
```

Nous pouvons enfin démarrer MIMEDefang :

```
$ sudo /etc/init.d/mimedefang start
.../...
```

2.3 Configuration de Sendmail

Que vous ayez installé MIMEDefang avec les paquets Debian ou en le compilant, il faut maintenant activer le Milter MIMEDefang dans la configuration de Sendmail (ou de Postfix). Pour cela, il suffit d'éditer le fichier **/etc/mail/sendmail.mc** et d'ajouter les lignes suivantes juste avant la ligne **dnl # Default Mailer setup** (vers la fin du fichier) :

```
dnl # Milters
INPUT_MAIL_FILTER(`mimedefang', `S=unix:/var/spool/MIMEDefang/mimedefang.
sock, F=T, T=S:1m;R:1m')
dnl #
```

Il faut ensuite re-générer la configuration et redémarrer Sendmail :

```
$ sudo make -C /etc/mail
$ sudo /etc/init.d/sendmail reload
```

3 Créer une politique de filtrage

J'imagine qu'il vous tarde de voir ce que la bête a dans le ventre. Juste une toute petite dernière section théorique et nous y serons.

3.1 Fonctions de rappel

Trois groupes de fonctions de rappel peuvent être utilisés dans le fichier **/etc/mail/mimedefang-filter** :

- les fonctions qui seront appelées lors des phases de connexion (**filter_relay**), **EHLO** (**filter_helo**), **MAIL FROM** (**filter_sender**) et **RCPT TO** (**filter_recipient**) ;
- les fonctions qui seront appelées lorsque l'intégralité du message aura été reçue après la phase DATA et qui permettront de parcourir les différentes entités MIME contenues dans le message (**filter_begin**, **filter**, **filter_multipart** et **filter_end**) ;
- les autres fonctions.

La figure 3 présente les différentes phases d'un dialogue SMTP ainsi que les fonctions de rappel de MIMEDefang qui sont exécutées lors de chacune de ces phases.

Nous allons construire le contenu de notre filtre petit à petit. Pour l'instant, mettez simplement de côté le fichier qui contient le filtre par défaut. Vous pourrez prendre le temps, un peu plus tard, de parcourir le contenu de ce fichier qui implémente une politique conseillée.

```
$ cd /etc/mail
$ sudo mv mimedefang-filter mimedefang-filter_orig
```

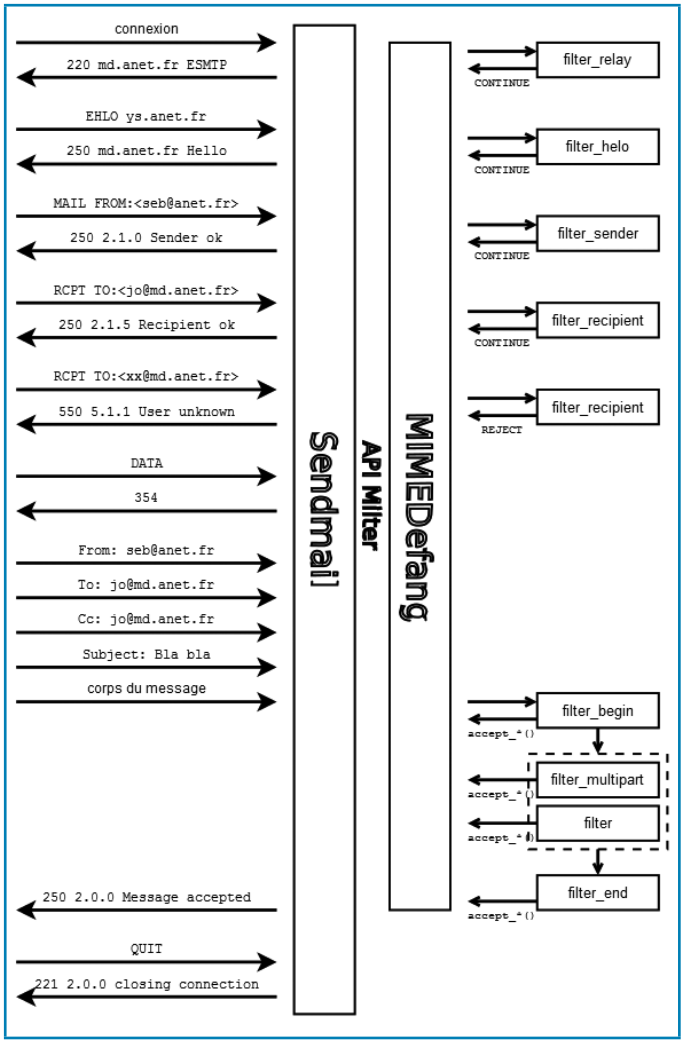


Figure 3 : Dialogue SMTP et fonctions de rappel de MIMEDefang

Le fichier **/etc/mail/mimedefang-filter** étant un fichier Perl qui est chargé par **mimedefang.pl**, il doit au minimum contenir la ligne suivante qui indique à l'interpréteur Perl que le fichier a été correctement lu et son code Perl initialisé :

```
1;
```

En fait, cette ligne devra toujours être présente à la fin du fichier **mimedefang-filter**. Créez donc pour l'instant ce fichier ne contenant que cette ligne puis demandez à MIMEDefang de relire ce fichier à l'aide de la commande **md-mx-ctrl** (pour *MimeDefang MultipleXor ConTRoL*) :

```
$ sudo md-mx-ctrl reread
Forced reread of filter rules
```

3.1.1 Fonctions de rappel enveloppe

Le tableau 1 présente les fonctions appelées par MIMEDefang lors des phases de la conversation SMTP qui concernent l'enveloppe du message. Par défaut, l'appel de ces fonctions est désactivé. Il faut l'activer explicitement pour chacune d'entre elles dans le fichier **/etc/default/mimedefang**. La deuxième colonne du tableau donne le nom du paramètre à activer pour chacune de ces fonctions. La troisième colonne indique la phase SMTP pendant laquelle la fonction est appelée et la dernière colonne la liste des paramètres qui sont passés à la fonction.

Les paramètres qui peuvent être passés à ces fonctions sont :

- **$hostip** : adresse IP du MTA distant ;
- **$hostname** : nom du MTA distant obtenu par une résolution DNS inverse ;
- **$helo** : chaîne de caractères fournie par le MTA distant lors de la phase **EHLO** ;
- **$sender** : adresse électronique de l'expéditeur fournie par le MTA distant lors de la phase **MAIL FROM** ;
- **$recipient** : adresse électronique du destinataire fournie par le MTA distant lors de la phase **RCPT TO** ;
- **$first** : adresse électronique du premier destinataire fournie par le MTA distant lors de la phase **RCPT TO** ;
- **$rcpt_mailer** : agent de distribution qui sera utilisé par Sendmail pour transmettre le message à ce destinataire ;
- **$rcpt_host** : nom du serveur vers lequel sera envoyé le message pour ce destinataire (si l'agent de distribution est esmtp) ;
- **$rcpt_addr** : adresse IP du serveur vers lequel sera envoyé le message pour ce destinataire (si l'agent de distribution est esmtp).

Les valeurs retournées par ces fonctions sont :

- **$action** : indique à MIMEDefang la décision prise concernant l'avenir du message ;
- **$msg** : message utilisé dans la réponse SMTP si le message est rejeté de manière temporaire ou définitive ;
- **$smtp_code** : optionnel, code de retour SMTP ;
- **$smtp_dsn** : optionnel, code de retour SMTP étendu (DSN) ;
- **$delay** : optionnel, délai introduit par MIMEDefang avant d'envoyer la réponse à Sendmail, le but est de monopoliser des sessions TCP et ralentir un éventuel spammeur.

Enfin, les valeurs possibles pour la première valeur retournée (**$action**) sont :

- **CONTINUE** : continuer la conversation SMTP ;
- **REJECT** : rejeter le message de manière définitive ;
- **TEMPFAIL** : rejeter le message de manière temporaire ;
- **DISCARD** : accepter le message mais le supprimer en silence ;
- **ACCEPT_AND_NO_MORE_FILTERING** : accepter le message et ne plus lui appliquer de règles de filtrage.

Fonction	Paramètre pour l'activation	Phase SMTP	Paramètres	Valeurs renvoyées
filter_relay	MX_RELAY_CHECK	connexion	$hostip, $hostname	$action, $msg
filter_helo	MX_HELO_CHECK	EHLO	$hostip, $hostname, $helo	$action, $msg, $smtp_code, $smtp_dsn, $delay
filter_sender	MX_SENDER_CHECK	MAIL FROM	$sender, $hostip, $hostname, $helo	$action, $msg, $smtp_code, $smtp_dsn, $delay
filter_recipient	MX_RECIPIENT_CHECK	RCPT TO	$recipient, $sender, $hostip, $hostname, $first, $helo, $rcpt_mailer, $rcpt_host, $rcpt_addr	$action, $msg, $smtp_code, $smtp_dsn, $delay

Tableau 1 : Fonctions de rappel enveloppe

Pour illustrer tout cela, construisons notre premier filtre dont voici le code :

```
sub filter_sender {
  my( $sender, $hostip, $hostname, $helo ) = @_;

  if( $hostip eq '192.168.0.10' ){
    return ( 'ACCEPT_AND_NO_MORE_FILTERING', 'Ok' );
  }

  if( $hostip eq '192.168.0.14' ){
    return ( 'REJECT', 'Blacklisted', '550', '5.7.1', 10 );
  }

  return ( 'CONTINUE', 'Ok' );
}

1;
```

Certes, tout cela est un peu naïf, mais nous allons lui donner de l'étoffe par la suite. Pour l'instant, notre fonction **filter_sender** qui est appelée lors de la phase **MAIL FROM** indique à MIMEDefang d'accepter tous les messages émis depuis l'adresse IP 192.168.0.10 sans plus faire appel à aucune des fonctions de notre filtre tandis que les messages émis depuis l'adresse IP 192.168.0.14 sont rejetés avec le code SMTP 550, le message « Blacklisted » et le code SMTP étendu 5.7.1. De plus, un délai de 10 secondes sera introduit avant de renvoyer la réponse à ce relais SMTP. Dans tous les autres cas, nous autorisons l'émetteur à poursuivre le dialogue SMTP. Pour tester cet exemple, n'oubliez pas de positionner la valeur du paramètre **MX_SENDER_CHECK** à **yes** dans le fichier **/etc/default/mimedefang** puis de redémarrer MIMEDefang (la commande **md-mx-ctrl reread** ne suffit pas lorsque le fichier **/etc/default/mimedefang** est modifié). Si nous testons l'envoi d'un message depuis l'adresse IP 192.168.0.14 :

```
$ swaks -f seb@anet.fr -t jo@md.anet.fr -s md.anet.fr -stl
=== Trying md.anet.fr:25...
=== Connected to md.anet.fr.
=== response in 1.014s
<-  220 md.anet.fr ESMTP Welcome, please be polit or go away!
 -> EHLO ys.anet.fr
=== response in 0.011s
<-  250-md.anet.fr Hello ys.anet.fr [192.168.0.14], pleased to meet
you
<-  250-ENHANCEDSTATUSCODES
<-  250-PIPELINING
<-  250-8BITMIME
<-  250-SIZE
<-  250-DSN
<-  250-AUTH DIGEST-MD5 CRAM-MD5
<-  250-STARTTLS
<-  250-DELIVERBY
<-  250 HELP
 -> MAIL FROM:<seb@md.anet.fr>
=== response in 9.782s
<** 550 5.7.1 <seb@md.anet.fr>... Blacklisted
 -< QUIT
=== response in 0.001s
<-  221 2.0.0 md.anet.fr closing connection
=== Connection closed with remote host.
```

Notre objectif est atteint, nous refusons de prendre en charge le message durant la conversation SMTP. Pas de file de messages refusés à gérer, pas de *bounce* à envoyer, pas de mauvaise conscience pour la suppression silencieuse d'un message. Nous économisons nos ressources et notre bande passante. Et même s'il advenait que le message refusé ait été légitime, pas de panique, le MTA qui a tenté de nous le transmettre a la responsabilité de prévenir l'expéditeur du message qu'un problème est survenu.

Il n'est pas bien difficile d'améliorer ce code. Tout d'abord, nous pourrions stocker dans une base de données la liste des relais autorisés. Pour cela, j'ai créé une base de données PostgreSQL nommée **defang** qui contient la table **relays_ok** dont la colonne **addr_ip** est de type inet. Ce type de colonne dans PostgreSQL modélise une adresse IPv4 ou IPv6. La connexion à la base de données est réalisée dans la fonction **filter_initialize** qui sera exécutée par MIMEDefang au démarrage du processus. En effet, aucun code ne doit être exécuté en dehors d'une fonction sous peine de gros ennuis si le mode embedded perl est activé. De même, la connexion à la base de données est fermée dans la fonction **filter_cleanup** qui est exécuté par MIMEDefang à la fin du processus. Au tout début de la fonction **filter_initialize**, un appel à la fonction **detect_and_load_perl_modules** est réalisé. Cette fonction fait partie des outils mis à disposition par MIMEDefang. Comme son nom le suggère, elle charge les modules Perl afin de fournir des services tels que la recherche dans les DNSBL, la connexion à un antivirus, à SpamAssassin, etc. La fonction **filter_sender** a été modifiée pour aller vérifier dans la table **relays_ok** si le relais SMTP est un relais auquel nous faisons confiance. Dans cette même fonction, nous utilisons la fonction **relay_is_blacklisted**, une fonction mise à disposition par MIMEDefang pour tester une DNSBL. Ici, nous utilisons la DNSBL agrégée de Spamhaus.

La nouvelle fonction **filter_recipient** est une fonction de rappel exécutée par MIMEDefang pour chaque commande **RCPT TO** envoyée par le client SMTP. Ici, nous utilisons de nouveau une fonction outil mise à disposition par MIMEDefang : **md_check_against_smtp_server**. Cette fonction permet de valider qu'un relais SMTP (**$my_relay**) acceptera un message avec un destinataire (**$recipient**), un expéditeur (**$sender**) et chaîne **HELO** (**$my_helo**) donnés, soit parce que le destinataire existe sur ce relais SMTP, soit parce qu'il relaiera le message à un autre serveur SMTP, soit enfin parce que ce serveur SMTP est le serveur de destination du message mais qu'il est incapable de vérifier qu'un destinataire est valide pendant la conversation SMTP (ce sera alors à lui de gérer les retours en erreur s'il accepte des messages vers des destinataires inconnus). Mais pourquoi prendre la peine de vérifier que le relais SMTP acceptera le message pour le destinataire final ? Simplement pour ne pas perdre au jeu de la patate chaude si notre relais SMTP n'est pas le

serveur de destination final. Si nous acceptons tous les messages sans vérifier que le relais SMTP suivant les acceptera, la patate chaude restera entre nos mains si celui-ci refuse de prendre en charge un message parce que l'adresse de destination n'existe pas. Nous testons la validité de l'adresse du destinataire uniquement si le mailer que va utiliser Sendmail pour faire suivre le message est esmtp. Si le destinataire est local, Sendmail vérifiera lui-même la validité de l'adresse.

```
use DBI;

my $db_dsn   = 'DBI:Pg:host=localhost;dbname=defang';
my $db_user  = 'defang';
my $db_pass  = 'DeFaNg';
my $my_helo  = 'md.anet.fr';
my $my_relay = 'mail.anet.fr';

my $db;


sub filter_initialize {
  detect_and_load_perl_modules();
  $db = DBI->connect( $db_dsn, $db_user, $db_pass, { PrintError => 0
});
}


sub filter_sender {
  my( $sender, $hostip, $hostname, $helo ) = @_;

  if( defined $db ){
    my $sth = $db->prepare( 'select 1 from relays_ok where addr_ip =
?' );
    $sth->execute( $hostip );
    my @row = $sth->fetchrow_array();
    return ( 'ACCEPT_AND_NO_MORE_FILTERING', 'Ok' )
      if @row && $row[0];
  }

  return ( 'REJECT', "Blacklisted, see http://www.spamhaus.org/query/
ip/$hostip", '550', '5.7.1', 10 )
    if relay_is_blacklisted( $hostip, 'zen.spamhaus.org' );

  return ( 'CONTINUE', 'Ok' );
}


sub filter_recipient {
  my( $recipient, $sender, $hostip, $hostname, $first, $helo, $rcpt_
mailer, $rcpt_host, $rcpt_addr ) = @_;

  return md_check_against_smtp_server( $sender, $recipient, $my_helo,
$my_relay )
    if $rcpt_mailer eq 'esmtp';

  return ( 'CONTINUE', 'Ok' );
}


sub filter_cleanup {
  if( defined $db ){
    $db->disconnect();
    $db = undef;
  }
  return 0;
}

1;
```

Outre les deux fonctions **filter_sender** et **filter_recipient**, deux autres fonctions de rappel peuvent être utilisées lors du traitement de l'enveloppe du message : **filter_relay** qui est appelée dès la connexion du MTA distant et avant même que Sendmail ne renvoie la bannière SMTP et **filter_helo** exécutée entre la réception de la commande SMTP **HELO** ou **EHLO** et la réponse envoyée par Sendmail. Cependant, il ne faut pas trop profiter des bonnes choses et limiter autant que possible l'usage des fonctions de rappel car cela charge MIMEDefang inutilement. Nous pourrions en effet tester si un relais est dans une liste noire dès la phase de connexion SMTP (dans la fonction **filter_relay**), mais ce serait au coût d'un appel de fonction MIMEDefang supplémentaire. Nous pouvons tout à fait attendre la phase SMTP **MAIL FROM** pour autoriser l'adresse IP du MTA distant en même temps que d'autres tests.

3.1.2 Fonctions de rappel contenu

Fort bien, nous avons vu comment filtrer des messages sur des critères liés à son enveloppe. Mais il existe d'autres fonctions de rappel qui vont permettre de scruter le contenu (en-têtes et corps) du message lui-même ainsi que d'intervenir sur ce contenu. Ces fonctions sont appelées après la fin de la phase DATA et juste avant la réponse que fait Sendmail au MTA distant. Elles sont au nombre de quatre :

- **filter_begin** : appelée une fois avant le traitement de chacune des entités MIME qui constituent le message ;
- **filter** : appelée pour chaque feuille de l'arbre des entités MIME ;
- **filter_multipart** : appelée pour chaque nœud de l'arbre des entités MIME ;
- **filter_end** : appelée une fois, lorsque toutes les entités MIME ont été traitées.

Un seul paramètre est passé aux deux fonctions **filter_begin** et **filter_end** : l'entité MIME (sous la forme d'un objet Perl **MIME::Entity**) qui représente le message. Quatre paramètres sont passés aux deux autres fonctions **filter** et **filter_multipart** : l'entité MIME courante dans l'arbre qui constitue le message, le nom de fichier suggéré par l'en-tête MIME Content-Disposition de l'entité, l'extension de ce nom de fichier et le type MIME suggéré par l'en-tête Content-Type de l'entité.

Ce qui est renvoyé par ces fonctions est ignoré, elles doivent utiliser l'une des fonctions **action_*** pour indiquer à MIMEDefang ce qui doit advenir du message ou de l'entité MIME courante. Ces fonctions sont séparées en deux groupes, celles qui s'appliquent au message lui-même et qui peuvent donc être utilisées dans les quatre fonctions de rappel sur le contenu et celles qui s'appliquent à une entité MIME

et qui ne sont donc utiles que pour les fonctions **filter** et **filter_multipart**. Cependant, à mon sens, accepter un message en supprimant ou modifiant l'une de ses entités MIME n'a plus de sens car les messages transmis sur Internet sont légitimes ou ne le sont pas. Il est plus que rare qu'un message transporte un contenu intéressant associé à un spam ou un virus. Et quand bien même, mieux vaut ne prendre aucun risque et refuser un message qui contient un virus plutôt qu'essayer de le dévéroller. Par ailleurs, si nous refusons le message, le MTA distant est censé générer un message d'erreur à l'expéditeur (un bounce, la fameuse patate chaude). Il ne reste donc que quatre possibilités : accepter le message (action par défaut), le supprimer ou le refuser définitivement ou temporairement. Les trois fonctions qui correspondent aux trois dernières actions sont :

- **action_discard()** : supprime silencieusement le message, je ne le recommande que dans des cas particuliers ;
- **action_bounce($reply, $code, $dsn)** : le message est refusé définitivement avec le code d'erreur SMTP **$code**, le code SMTP étendu **$dsn** et le message **$reply** ;
- **action_tempfail($reply, $code, $dsn)** : le message est refusé temporairement avec le code d'erreur SMTP **$code**, le code SMTP étendu **$dsn** et le message **$reply**, le MTA distant devra présenter à nouveau le message à un autre moment (dont il est seul juge).

Les paramètres **$code** et **$dsn** des fonctions **action_bounce** et **action_tempfail** sont optionnels. S'ils sont absents, MIMEDefang utilisera des valeurs par défaut génériques.

Avec tous ces outils, nous pouvons enrichir notre filtre avec le code suivant (il est à ajouter au code précédent, mais avant la dernière ligne du fichier qui contient la ligne 1;) :

```
sub filter_begin {
  my( $entity ) = @_;

  return action_bounce( "Invalid characters in header" )
    if $SuspiciousCharsInHeaders;

  md_copy_orig_msg_to_work_dir_as_mbox_file();
  my( $code, $category, $action ) = message_contains_virus();

  return action_tempfail( "Problem running virus-scanner" )
    if $action eq "tempfail";

  return action_bounce( "Virus $VirusName found" )
    if $category eq "virus" || $category eq "suspicious";
}


sub filter {
  my( $entity, $fname, $ext, $type ) = @_;

  return if message_rejected();

  return action_bounce( "MIME type message/partial not accepted" )
    if lc( $type ) eq "message/partial";
}


sub filter_multipart {
  my( $entity, $fname, $ext, $type ) = @_;

  return if message_rejected();

  return action_bounce( "MIME type message/partial not accepted" )
    if lc( $type ) eq "message/partial";
}


sub filter_end {
  my( $entity ) = @_;

  return if message_rejected();

  return action_bounce( "We do not accept spam here" )
    if $Features{"SpamAssassin"}
      && -s "./INPUTMSG" < 204800
      && spam_assassin_is_spam();

  action_rebuild();
}
```

Dans la fonction **filter_begin**, nous commençons tout d'abord par vérifier s'il ne se trouve pas dans les en-têtes du message des caractères douteux. MIMEDefang positionne la variable **$SuspiciousCharsInHeaders** à une valeur vraie si cela est le cas, et nous rejetons alors le message. Nous devons ensuite vérifier si le message contient un virus. Pour cela, la fonction **md_copy_orig_msg_to_work_dir_as_mbox_file** est tout d'abord utilisée pour recopier le message dans son intégralité au format mbox afin que ClamAV l'identifie comme un message électronique. Puis la fonction **message_contains_virus** permet de vérifier si un virus est présent en utilisant les antivirus qui ont été activés (ClamAV seul dans notre cas, mais MIMEDefang supporte de nombreux autres antivirus). Nous retournons une erreur temporaire si l'antivirus ne peut être exécuté et une erreur permanente si un virus est identifié.

Les fonctions **filter** et **filter_multipart** sont identiques et succinctes. Nous commençons par nous épargner du travail si le message a déjà été rejeté, ce que nous pouvons savoir en utilisant la fonction **message_rejected**. Puis nous rejetons le message si l'entité MIME est de type message partiel. Ce type d'entité permet de fragmenter (généralement de manière automatique) des fichiers de taille importante afin de les envoyer en plusieurs messages. Cela doit être évité car cette technique peut être utilisée pour dissimuler du contenu malicieux ou pour envoyer une image ISO d'un CD en utilisant plus de 200 messages d'une taille inférieure à 5 Mo (comme je l'ai vu faire par un étudiant d'un établissement d'enseignement supérieur).

Enfin, la fonction **filter_end** rejette le message si le module **Mail::SpamAssassin** est installé, si la taille du message est supérieure à 200 Ko (les spams ont rarement une taille plus élevée) et si SpamAssassin identifie le contenu du message comme étant un spam. Puis, l'appel à la fonction **action_rebuild** force MIMEDefang à reconstruire le message à partir de ses entités MIME préalablement décodées. Cela n'est pas absolument nécessaire si les entités MIME du message n'ont pas été modifiées. De plus, cette action est relativement coûteuse en ressources. Mais elle permet de s'assurer que l'arbre des entités MIME du message est sain et cohérent car certains logiciels malicieux peuvent tenter d'utiliser des constructions MIME erronées pour duper les clients de messagerie.

L'exécution de toutes ces opération peut s'avérer relativement longue. Mais cela ne pose pas de problème, les délais autorisés pour la phase DATA sont généralement élevés. À titre d'exemple, le dépassement de délai configuré par défaut par Sendmail pour cette phase est d'une heure, mais il est plus que probable que ce délai soit bien moins élevé dans beaucoup de MTA de grands opérateurs. Cela nous laisse néanmoins de la marge.

Ces nouvelles fonctions utilisent les connecteurs à ClamAV et SpamAssassin de MIMEDefang. Mais, autant le module **Mail::SpamAssassin** est automatiquement détecté par MIMEDefang, autant il nous faut lui indiquer de manière explicite que nous souhaitons utiliser l'antivirus ClamAV. Pour cela, le fichier **/etc/mail/mimedefang.pl.conf** doit contenir les lignes suivantes :

```
$Features{'Virus:CLAMD'} = 1;
$ClamdSock = '/var/run/clamav/clamd.ctl';

1;
```

Notre filtre commence à prendre de la matière. Il est donc plus facilement sujet aux erreurs de syntaxe. Avant de demander à MIMEDefang de lire une nouvelle version de ce fichier sur une passerelle de messagerie en production, il serait prudent d'utiliser la commande **mimedefang.pl -test** pour en vérifier la syntaxe comme ceci :

```
$ sudo mimedefang.pl -test
Filter /etc/mail/mimedefang-filter seems syntactically correct.
$ sudo md-mx-ctrl reread
Forced reread of filter rules
```

Il également possible de valider votre filtre de manière plus détaillée en y ajoutant une fonction **filter_validate**. Par exemple :

```
sub filter_validate {
  detect_and_load_perl_modules();

  unless( $Features{"SpamAssassin"} ){
    print "SpamAssassin not installed\n";
    return 1;
  }

  my $db = DBI->connect( $db_dsn, $db_user, $db_pass );
  unless( defined $db ){
    print "Error while connecting to database\n";
    return 2;
  }

  $db->disconnect();
  return 0;
}
```

Cette fonction sera exécutée par MIMEDefang si vous l'appelez avec la commande **mimedefang.pl -validate** dont le code de sortie est celui de la fonction **filter_validate** :

```
$ sudo mimedefang.pl -validate; echo $?
0
```

Nous pouvons vérifier que notre politique de filtrage est bien en place en utilisant notre fidèle swaks et la signature du projet Eicar (http://www.eicar.org/) afin de simuler la transmission d'un virus :

```
$ swaks -f seb@anet.fr -t jo@md.anet.fr -s md.anet.fr --body eicar.
txt
=== Trying md.anet.fr:25...
=== Connected to md.anet.fr.
.../...
 -> DATA
.../...
 -> .
<** 554 5.7.1 Virus Eicar-Test-Signature found
 -> QUIT
<-  221 2.0.0 md.anet.fr closing connection
=== Connection closed with remote host.
```

Et, pour tester la détection d'un spam avec SpamAssassin :

```
$ swaks -f seb@anet.fr -t jo@md.anet.fr -s md.anet.fr --body spam1.
txt
=== Trying md.anet.fr:25...
=== Connected to md.anet.fr.
.../...
 -> DATA
.../...
 -> .
<** 554 5.7.1 We do not accept spam here
 -> QUIT
<-  221 2.0.0 md.anet.fr closing connection
=== Connection closed with remote host.
```

3.2 Modifier le message

Notre filtre commence a être acceptable, mais il n'implémente que des règles qui restent, somme toute, conventionnelles : tester la cohérence d'un message, vérifier s'il contient un virus ou un spam, valider le destinataire, consulter les DNSBL, interdire certains types d'entités MIME.

Certes, il le fait avec élégance, mais MIMEDefang est capable de bien d'autres prouesses telles que modifier le message, ses en-têtes ou ses entités MIME (c'est-à-dire, notamment, ses pièces attachées).

3.2.1 Ajouter un destinataire

Un client avait besoin que les messages à destination de certaines adresses de messagerie soient copiés vers une nouvelle adresse et qu'une étiquette soit ajoutée au début du sujet. Mission immanente facile et réalisée en quelques lignes de Perl intégrées à MIMEDefang.

Pour cela, j'ai créé une table **mail_copy** dans la base de données dont les colonnes étaient **rcpt** (destinataire original, clé primaire), **bcc_to** (destinataire en copie cachée) et **tag** (étiquette à ajouter devant le sujet). Par exemple :

```
defang=> select * from mail_copy;
     rcpt      |     bcc_to      |  tag
---------------+-----------------+--------
 jo@md.anet.fr | jack@md.anet.fr | [Plop]
(1 row)
```

Si je repars de notre précédent filtre, j'ai simplement à ajouter une fonction qui permet de normaliser les adresses de messagerie du destinataire et une dizaine de lignes de code à la fonction **filter_end**, comme ceci :

```
sub canonicalize_address {
  my( $addr ) = @_;
  $addr =~ s/^<?(.+?)>?$/$1/;
  $addr = lc( $addr );
  return $addr;
}


sub filter_end {
  my( $entity ) = @_;

  return if message_rejected();

  return action_bounce( "We do not accept spam here" )
    if $Features{"SpamAssassin"}
      && -s "./INPUTMSG" < 204800
      && spam_assassin_is_spam();

  my $sql = 'select bcc_to, tag from mail_copy where rcpt = ?';

  if( defined $db ){
    foreach my $rcpt ( @Recipients ){
      $rcpt = canonicalize_address( $rcpt );
      @res = $dbh->selectrow_array( $sql, undef, $rcpt );
      if( @res ){
        add_recipient( $res[0] );
        action_delete_all_headers( 'Subject' );
        action_add_header( 'Subject', "$res[1] $Subject" );
        last;
      }
    }
  }

  action_rebuild();
}
```

Une fois encore, MIMEDefang nous facilite la vie. En effet, il a collecté pour nous la liste des destinataires dans le tableau **@Recipients** qu'il nous suffit de parcourir. Il met également à notre disposition des routines bien pratiques telles que **action_delete_all_headers** et **action_add_header**. Le sujet original du message est également disponible dans la variable **$Subject**.

Un petit piège cependant : ne pas oublier que les adresses manipulées par MIMEDefang sont les adresses de l'enveloppe qui peuvent être différentes des adresses présentes dans l'en-tête du message.

3.2.2 Remplacer une pièce attachée

Une autre situation relativement classique : les collaborateurs d'une société ont pris l'habitude d'envoyer des pièces attachées de taille importante par messagerie électronique. L'idée est d'autoriser l'envoi de pièces attachées jusqu'à plusieurs centaines de méga-octets, mais de les intercepter avant que ces messages ne soient relayés (et généralement refusés par les autres serveurs de messagerie). Ces pièces jointes sont alors copiées dans un répertoire accessible par l'intermédiaire d'un serveur HTTP et sont remplacées dans le message par une URL qui pointe vers la pièce jointe sur le serveur HTTP. Afin de garantir la sécurité du procédé, le nom du fichier est renommé en utilisant une empreinte SHA1 et l'affichage du contenu du répertoire doit être interdit par le serveur HTTP.

Le code qui permet de faire toutes ces choses merveilleuses est un miracle de simplicité et tient en trois lignes dans la fonction **filter** grâce à la fonction **action_replace_with_url** que MIMEDefang met à notre disposition :

```
my $big_size = 5*1024*1024;
my $big_root = '/var/www/defang';
my $big_url  = 'http://seb.anet.fr/defang';
my $big_msg  = << "EOT";
The attachment was larger than $big_size bytes.
It was removed, but may be accessed at this URL:\n
\t_URL_
EOT

sub filter {
  my( $entity, $fname, $ext, $type ) = @_;

  return if message_rejected();

  return action_bounce( "MIME type message/partial not accepted" )
    if lc( $type ) eq "message/partial";

  my $entity_size = -s $entity->bodyhandle->path;
  action_replace_with_url( $entity, $big_root, $big_url, $big_msg )
    if $entity_size > $big_size;
}
```

Toutes les pièces attachées de plus de 5 Mo sont donc copiées dans le répertoire **/var/www/defang** et remplacées

par un message contenant une URL commençant par http://seb.anet.fr/defang (la balise **_URL_** dans le message est remplacée par cette URL par MIMEDefang).

Il ne nous reste plus qu'à créer le répertoire **/var/www/defang**, faire en sorte que MIMEDefang puisse y écrire, configurer le serveur HTTP et tester tout cela :

```
$ sudo mkdir -m 750 /var/www/defang
$ sudo chown defang:www-data /var/www/defang
$ sudo mimedefang.pl -validate; echo $?
0
$ sudo md-mx-ctrl reread
Forced reread of filter rules
$ swaks -f seb@anet.fr -t jo@md.anet.fr -s md.anet.fr --attach big_file.dat
.../...
$ sudo ls -l /var/www/defang
total 2056
-rw-r--r-- 1 defang defang 2097157 Apr 14 20:20
ea0ad94d1ad7ba5e67a7d43076d8c004c645991c
```

Conclusion

Comprenez-vous maintenant pourquoi je considère MIMEDefang comme le couteau suisse de l'administrateur de messagerie ? Il peut tellement en tellement peu de lignes que c'en est un bonheur. Cet article a présenté quelques exemples simples, mais il est possible de faire des filtres bien plus évolués. Pour aller plus loin, vous prendrez connaissance avec la présentation qui fait référence (http://www.mimedefang.org/static/mimedefang-lisa04.pdf) et la page de manuel mimedefang-filter(5) qui décrit tout l'environnement mis à disposition par MIMEDefang pour écrire une politique de filtrage.

Finalement, je certifie le produit stable et exempt de bugs importants (ou alors ils sont bien camouflés) pour l'utiliser depuis des années afin de filtrer plusieurs millions de messages par mois, voire par jour sur certaines installations. ■

Bibliographie

- Présentation de référence de MIMEDefang : http://www.mimedefang.org/static/mimedefang-lisa04.pdf
- Le principe de la patate chaude : http://clx.asso.fr/spip/Lutter-efficacement-contre-les
- Le RFC 5321 (Simple Mail Transfer Protocol) : http://www.rfc-editor.org/rfc/rfc5321.txt
- Le RFC 3463 (Enhanced Mail System Status Codes) : http://www.rfc-editor.org/rfc/rfc3463.txt
- La page de manuel mimedefang-filter(5)

MONTER SON PROPRE NAS SOUS FREEBSD 9

par Cédric PELLERIN
[Utilisateur de GNU/Linux depuis 1993]

Pour de sombres raisons de licence, le système de fichiers ZFS n'est pas disponible dans le kernel Linux. Pour en profiter, il faut soit passer pas FUSE, soit installer un OpenSolaris, soit utiliser FreeBSD. C'est cette dernière solution qui a été retenue.

Mais z'enfin, pourquoi donc fonder son article sur FreeBSD dans un magazine qui s'intitule GNU/Linux Magazine France ? Tout d'abord parce que le rédac'chef est d'accord, ensuite parce qu'on ne va pas laisser nos copains de GCU Squad squatter l'ensemble des pages concernant les *BSD et enfin, parce qu'il est toujours bon de s'ouvrir un peu au monde extérieur et que savoir installer et faire rouler un FreeBSD, même a minima, peut toujours être utile, ne serait-ce que pour draguer les geekettes ;) [NDLR : nous tenons à informer le lecteur qu'il n'existe pas, à notre connaissance, de preuve tangible démontrant l'efficacité de cette approche dans ce domaine]

Comme son nom ne l'indique pas forcément, un NAS a pour vocation de centraliser le stockage des données accessibles sur un réseau, local ou non. Qui dit centralisation dit partage, donc droits d'accès et grosse capacité de disque. Avoir beaucoup de téraoctets de disque c'est bien, mais avoir la possibilité de survivre à la perte d'un ou plusieurs disques c'est mieux. À la suite de quoi il faut penser au provisionning, c'est-à-dire à la façon de rapatrier les gros fichiers (genre images ISO de distributions libres...) et pour ce faire, le plus efficace reste le protocole BitTorrent. Une fois munis de ces considérants, nous pouvons passer à l'élaboration du...

1 Cahier des charges

Comme nous ne travaillons ni pour un grand groupe, ni pour l'administration, nous allons essayer de monter un cahier des charges non évolutif. Cela signifie qu'il faut essayer de penser à tout avant de démarrer l'implémentation. Donc voici à gauche nos desiderata, au milieu la ou les solutions logicielles permettant de répondre aux attentes et à droite les solutions retenues pour notre projet :

Besoins	Réponses possibles	Réponses retenues
Un OS fiable et libre	Linux, *BSD	FreeBSD 9
Un système de fichiers sécurisé et facilement évolutif	ZFS, Btrfs, LVM	ZFS
Un système de partage de fichiers simple et largement compatible	NFS, CIFS	NFS, Samba
Un client BitTorrent fiable, accessible depuis n'importe quel poste du réseau	Deluge	Deluge

D'autres solutions sont possibles, mais nous allons nous concentrer sur celles qui sont suffisantes pour ce que nous voulons faire.

Sur le plan hardware, une petite machine de récupération fera amplement l'affaire. Pour ma part, j'ai récupéré un dual-core avec 3Go de RAM et ça tourne très bien. Nous allons lui mettre deux disques de 10Go (que l'on passera tout à l'heure en miroir) pour le système et quatre disques d'un ou deux To pour les données. En ce qui concerne les disques il est fortement conseillé de rajouter des tiroirs hot-plug, au moins pour les données, ce qui évitera de devoir arrêter le serveur en cas de panne d'un des disques.

2 Le système d'exploitation

2.1 Installation de base

L'installation de FreeBSD se déroule sans anicroche pour qui ne cherche pas à tout prix à utiliser une souris et il faut aussi savoir booter sur CD ou sur clé USB... Après une première phase de boot nous arrivons sur l'écran d'accueil suivant :

Figure 1

Sur une machine « classique », nous pouvons nous contenter d'appuyer sur ENTER pour continuer le boot et FreeBSD nous propose ensuite de démarrer en Live CD, de lancer un shell ou d'installer l'OS.

Figure 2

Inutile de dire que nous choisissons l'option « Install » ce qui nous amène à devoir faire des choix cornéliens pour répondre aux questions de cette boîte de dialogue :

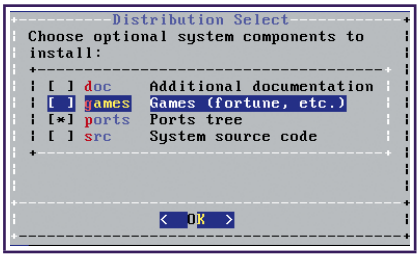


Figure 3

Dans notre cas, il est possible d'installer les ports (bases des paquets logiciels, nous verrons ça de plus près un peu plus tard) ou d'attendre la fin des opérations pour ça. Si vous avez une machine assez rapide en ce qui concerne les disques, faites-le maintenant, sinon il est tout aussi judicieux d'attendre la fin de l'installation pour les télécharger.

Une fois ce choix difficile effectué, nous arrivons sur le formatage des disques. Le système de gestion des disques sous *BSD étant assez particulier, il est plus sage pour une fois de choisir l'option pour « neuneux » et de se laisser guider :

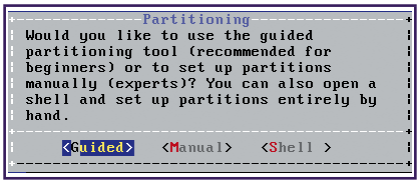


Figure 4

Ensuite, il ne reste plus qu'à faire Suivant, Suivant, Suivant, Terminer... je veux dire appuyer sur ENTER en faisant bien attention à ce que l'on fait ;)

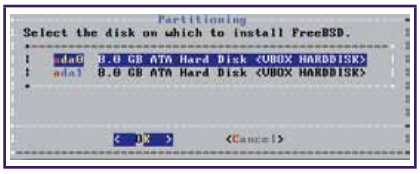


Figure 5

Il s'agit là de choisir le disque de démarrage. On remarque déjà que la façon de nommer les disques n'est pas la même que sous Debian. Je vous rassure, c'est pareil avec tous les périphériques, mais on s'y fait vite et on finirait même par préférer cette méthode qui a pas mal d'avantages. Ensuite, on valide le choix effectué, on installera que FreeBSD sur ce disque :

Figure 6

Ensuite de quoi, l'installeur nous demande de créer le boot record :

Figure 7

avec un type GPT :

Figure 8

puis nous demandons à l'installeur de créer pour nous les partitions en choisissant le bouton <Auto>, ce qui nous donne le résultat suivant avec une partition minimaliste de boot, une pour le root et un swap :

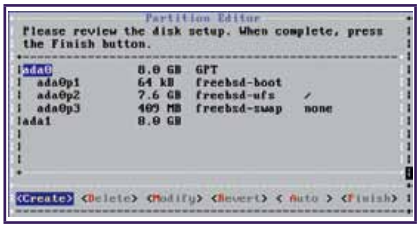


Figure 9

Une fois ceci fait nous « cliquons » sur <Finish> et la copie des fichiers commence. On en a pour quelques minutes :

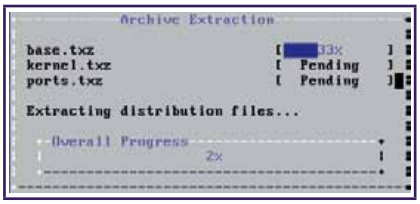


Figure 10

Ceci fait, nous passons à la configuration du système :

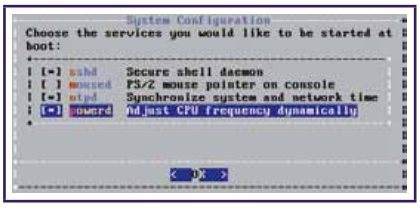


Figure 11

et si vous faites vos tests dans une VM VirtualBox, décochez **powerd** sous peine de devoir le désactiver manuellement ensuite.

Comme nous arrivons à la fin, FreeBSD nous propose de créer un utilisateur de base que nous appellerons **films** et à qui nous attribuerons le mot de passe hyper-sécurisé « films » :

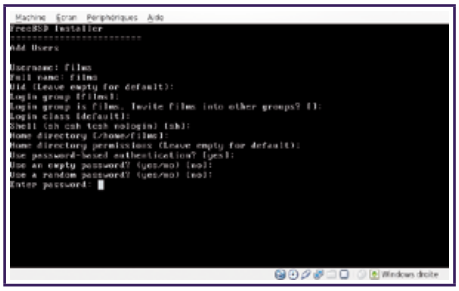


Figure 12

Voilà, l'installation est finie, reste à rebooter sur notre nouveau FreeBSD tout neuf, nous ferons les réglages finaux juste après.

Une fois cette première partie achevée, nous pouvons nous connecter en tant que root et effectuer les modifications qui s'imposent. Il faut savoir que 95 % des réglages système dans FreeBSD se font dans le fichier **/etc/rc.conf**. On y précise le nom de la machine, les daemons à lancer au démarrage avec leurs options éventuelles, la configuration réseau, etc.

Commençons par la configuration du réseau. La première chose à faire consiste à trouver le nom de sa carte réseau. En effet, sous *BSD, le nom des devices est fonction du fabricant. Dans le cas d'une Intel e1000 comme chez moi (82540EM pour les intimes), le petit nom de l'interface sera **em0** pour la première, **em1** pour la deuxième, etc. Si vous avez une Broadcom gigabit genre BCM5705, vous trouverez une interface nommée **bge0**.

Note

Pour avoir la liste des devices PCI présents ainsi que les noms correspondants, la commande sous FreeBSD est

```
# pciconf -lv
```

pour avoir toutes les informations sur les devices, vous pouvez essayer

```
# pciconf -lbcv
```

et là vous aurez besoin de scroller en arrière. Si vous êtes en ssh sur la machine pas de problème, mais si vous êtes en console directe, le bon vieux Shift-PgUp ne marche pas. En effet, sous FreeBSD il faut utiliser la touche ScrollLock qui bascule en mode défilement et ensuite on se balade avec les flèches haut et bas,

Bref, revenons à notre réseau et à notre **em0** et allons lui expliquer qu'il doit récupérer son adresse IP en DHCP et que son routeur est en 192.168.0.1. On va en profiter pour modifier le nom d'hôte et l'appeler **fbnas.test.glmf**. Éditons donc **/etc/rc.conf** avec notre éditeur favori (vi est livré de base dans une version minimaliste) et modifions la directive **hostname** :

```
hostname="fbnas.test.glmf"
```

puis configurons le réseau avec les deux lignes suivantes :

```
ifconfig_em0="DHCP"
defaultrouter="192.168.0.1"
```

Si nous voulons lui mettre une adresse IP fixe - chose recommandée pour un serveur - il suffirait de préciser :

```
ifconfig_em0="inet 192.168.0.10 netmask
255.255.255.0"
```

Tant qu'on a les mains dedans, profitons-en pour activer le serveur SSH avec la directive :

```
sshd_enable="YES"
```

et voilà pour le moment. On y reviendra au fur et à mesure des développements de notre serveur.

2.2 Mise en RAID1 des disques système

C'est bien d'avoir prévu de la sécurité pour les données, mais que faire lorsque le disque système explose en vol ? La solution s'appelle mirroring et cela va nous donner l'occasion de regarder le pendant de **md** sous Linux qui s'intitule **geom** ici.

Nous partons du principe que nous avons deux disques physiques de même taille et que nous venons de finaliser l'installation de base de FreeBSD sur celui qui s'appelle **ada0**. Le deuxième s'appellera logiquement **ada1** et voici ce que nous allons leur faire subir pour les passer en RAID1. La procédure ressemble assez à celle que j'ai décrite voici quelques années dans ces mêmes colonnes, mais mise à la sauce FreeBSD.

Commençons par charger le module **geom_mirror** qui se charge du RAID1 :

```
# kldload geom_mirror
```

puis automatisons ce chargement en allant éditer cette fois le fichier **/boot/loader.conf** et en lui rajoutant la ligne :

```
geom_mirror_load="YES"
```

Une fois ceci effectué, il faut construire le miroir sur ada1 (évitons de tout casser maintenant) en lui affectant un nom comme **gm0** par exemple :

```
# gmirror label gm0 ada1
```

puis vérifions avec :

```
# gmirror list
```

que notre embryon de miroir soit lancé.

Notre disque **ada1** est vierge de toute partition utilisable par FreeBSD, il est juste flaggé comme faisant partie intégrante de **gm0**. Il va donc falloir recopier le partitionnement de **ada0** sur **ada1**. Un moyen fort simple existe en deux commandes :

```
# gpart backup ada0 > /root/partada0
# gpart restore mirror/gm0 < /root/partada0
```

oui, on écrit directement sur **gm0**, car il contient déjà **ada1**.

Il reste maintenant à rendre le miroir bootable :

```
# gpart bootcode -b /boot/pmbr mirror/gm0
# gpart bootcode -p /boot/gptboot -i 1 mirror/gm0
```

ce qui signifie, remis en vieux gaulois :

- Insère le code de bootstrap dans les metadata du miroir (copie les 512 octets de boot primaire sur le MBR).
- Écrit le code de bootstrap sur la partition n° 1.

Ce qui correspond en gros aux stage 1 et stage 2 de Grub.

Il ne nous reste plus qu'à créer un filesystem sur la partition n° 2 de **gm0** (celle qui sera monté sur **/**) et copier les données, ce qui se fait avec les commandes suivantes :

```
# newfs -U /dev/mirror/gm0p2
# mount /dev/mirror/gm0p2 /mnt
# cd /
# tar --one-file-system -cpf - . | ( cd /mnt && tar -xvpf - )
```

Si vous avez demandé la copie des ports lors de la première phase d'installation, cette dernière commande peut prendre un certain temps. Profitez-en pour admirer au passage les légères différences syntaxiques entre GNU **tar** et BSD **tar**.

La copie terminée, il ne faut surtout pas oublier de modifier le **fstab** sur **gm0** en remplaçant toutes les occurrences de **ada0** par **mirror/gm0** dans **/mnt/etc/fstab**.

Nous pouvons maintenant rebooter la machine en demandant au BIOS de démarrer sur le deuxième disque, sinon nous allons être bloqués.

```
# umount /mnt
# shutdown -r now
```

Si tout se passe bien, et il n'y a aucune raison pour qu'il en soit autrement, nous pouvons maintenant intégrer **ada0** dans le miroir **gm0**. Nous allons quand même valider que nous avons les bons points de montage sur **/** et pour le swap :

```
# mount
# swapinfo
```

Ces deux commandes devraient nous parler de **mirror/gm0p2** pour la première et **mirror/gm0p3** pour la seconde.

Une fois cette validation effectuée, nous pouvons intégrer **ada0** dans **gm0** :

```
# gmirror insert gm0 ada0
```

puis nous pouvons suivre la progression de la synchronisation avec :

```
# gmirror status
```

À la fin de cette synchronisation, un dernier reboot est conseillé afin de valider l'ensemble.

2.3 Installation des logiciels de base

FreeBSD a pour particularité d'utiliser par défaut **csh** comme shell. Personnellement, je n'ai rien contre, mais je préfère **bash** que je maîtrise mieux. Par la même occasion, nous allons installer **wget** (parce que ça peut toujours servir ;)) et un **vim** fonctionnel. Il serait possible de les installer avec les paquets binaires avec **pkg_add**, mais je trouve que c'est un excellent prétexte pour aller jouer avec les ports.

La base des ports est installée dans **/usr/ports**. Ils sont répartis par catégorie et, au début du moins, **find -sri** est notre ami. Prenons d'abord l'exemple de **bash**. Son port est situé dans **/usr/ports/shell/bash**. Que trouvons-nous ici ?

```
root@Fbnas:/usr/ports/shells/bash # ls
Makefile        distinfo        files           pkg-descr       pkg-plist
```

- **Makefile** qui permet de compiler et de résoudre les dépendances du paquet ;
- **distinfo** où sont stockées les tailles et sha256sum des archives et patchs éventuels composant le paquet ;
- Le répertoire éventuel **files/** où on trouvera divers patchs permettant la compilation sous FreeBSD. Ce répertoire n'est présent qu'en cas de besoin ;
- **pkg-descr** qui décrit le paquet en langage clair ;
- **pkg-plist** qui indique où placer les fichiers dans l'arborescence lors de l'installation.

Note

ATTENTION : Tout logiciel installé via les ports le sera avec le préfixe **/usr/local**. Donc la configuration se trouvera dans **/usr/local/etc** et non **/etc**, les binaires dans **/usr/local/bin** et non **/usr/bin**, etc.

La commande « magique » pour installer s'appelle :

```
# make install clean
```

mais il peut être utile de commencer avec un :

```
# make config
```

afin de changer les paramètres par défaut. Tous les paquets ne supportent pas cette configuration, mais ils sauront vous le dire gentiment. Pour connaître les paramètres existants, il suffit de le demander avec :

```
# make showconfig
```

Lançons donc cette commande magique pour bash. La première chose que l'on remarque est que FreeBSD commence par télécharger la dernière archive des sources qu'il connaît. Ensuite, il télécharge les patchs éventuels, les applique puis lance la compilation.

Dans le cas où il manque des dépendances, le système est assez intelligent dans 99 % des cas pour aller les chercher tout seul et les installer de façon transparente.

Une fois bash installé, nous allons expliquer au système que nous préférons ce shell. Pour ce faire, il est obligatoire d'utiliser la commande **chsh [user]**, la modification de **/etc/passwd** n'étant pas effective sous FreeBSD. Cette commande nous lance un vim avec un fichier temporaire pré-rempli :

```
Password: $6$cuLizfbU/OJyonQW$JZ7VH3qy/1vVYqF.FDr7Rguaos6NTQ3/
iDSRØIOTCPfRNUSoAyJloLCYMØrmrIX/jeZ5Q4fz52v./sljSlSohØ
Uid [#]: Ø
Gid [# or name]: Ø
Change [month day year]:
Expire [month day year]:
Class:
Home directory: /root
Shell: /bin/csh
Full Name: Charlie &
Office Location:
Office Phone:
Home Phone:
Other information:
```

qu'il suffit de modifier et d'enregistrer pour que les changements soient effectifs.

Tant qu'on y est, nous allons rajouter l'utilisateur **films** au groupe **wheel** afin de lui permettre de faire un **su**, car FreeBSD interdit par défaut toute connexion SSH en root et rester devant la console ça va cinq minutes, mais on aime bien son petit confort... La commande idoine est la suivante :

```
# pw usermod films -G wheel
```

Sous FreeBSD, **pw** est la commande générique qui permet de créer, supprimer, modifier et afficher les utilisateurs et les groupes. Un petit coup de **man pw** est à recommander à ce stade :)

Maintenant, nous pouvons aller installer **wget** et **vim** le cœur léger, nous savons comment faire :

```
# cd /usr/ports/ftp/wget
# make config
# make install clean
```

Normalement tout va bien, le téléchargement, la compilation et l'installation se déroulent sans aucun souci. Nous pouvons donc passer à **vim** :

```
# cd /usr/ports/editors/vim
# make install clean
```

et paf, ça plante, le téléchargement des patchs s'explose en plein vol. Mais que faire dans ce cas là ? La réponse est simple, le site de téléchargement utilisé par le **Makefile** semblant un peu bancal, nous allons aller chercher les patchs pour vim sur le site officiel et les insérer là où le Makefile s'attend à les trouver, c'est-à-dire **/usr/ports/distfiles/vim**. Il existe à l'heure actuelle plus de 900 patchs pour Vim 7.3, mais notre installateur ne les gère pas tous. Nul besoin d'en télécharger plus que nécessaire certes, mais comment savoir jusqu'où il faut aller ? Cette information est disponible dans le fichier **Makefile** au début :

```
PATCHLEVEL=     669
```

et hop, voilà on sait tout. Reste à aller chercher tout ça sur le site officiel :

```
# cd /usr/ports/distfiles/vim
# for i in `seq 1 669`; do j=`printf "%Ø3d" $i`; wget ftp://ftp.vim.
org/pub/vim/patches/7.3/7.3.${j}; done
```

finalement on n'a pas compilé **wget** pour rien, chouette !

Une fois ce – long – rapatriement fini, il suffit de relancer la commande **make install clean** pour que tout se passe comme sur des roulettes... heu presque. M'enfin il veut me compiler la libX11, GTK+ et autres trucs, je n'en ai pas besoin moi ! C'est parce que la configuration par défaut demande à **vim** d'installer aussi l'interface clickodromique nommée **gvim** dont nous n'avons absolument pas l'usage étant donné qu'un serveur ne possède pas d'interface graphique, c'est dans la définition du mot. On s'oriente donc vers un :

```
# make config
```

mais il nous répond benoîtement :

```
===> No options to configure
```

Ok, on se calme, on va faire un petit tour sur le Net et hop, voilà l'option magique :

```
# make WITHOUT_X11=yes install clean
```

et là tout de suite ça va beaucoup mieux et nettement plus vite.

Si quelques librairies surnuméraires ont eu le temps de s'installer avant votre Ctrl-C aussi rageur que justifié, le moyen simple de les supprimer s'appelle **pkg_delete**. En effet, une fois installé, un port s'enregistre dans la liste des paquets, comme s'il provenait d'un package binaire, cela simplifie fortement la gestion.

2.4 Mise en place du pool de données

Maintenant que nous disposons d'une base opérationnelle, nous pouvons nous attaquer aux données. La première chose à faire, après avoir installé physiquement les disques bien entendu, consiste à aller chercher comment FreeBSD les a nommés :

```
# dmesg | grep ad
```

qui doit nous permettre de voir les disques de **ada0** à **ada5**. **ada0** et **ada1** sont nos disques système qui composent **gm0** donc **ada2** à **ada5** sont bien nos disques ayant pour vocation d'intégrer un pool ZFS. Sous FreeBSD, il est possible de travailler avec le nom des devices ou avec le nom d'un label donné à chaque device. Pour ce faire, commençons par nommer nos disques avec la série de commandes :

```
# glabel label diskzfs1 /dev/ada2
# glabel label diskzfs2 /dev/ada3
# glabel label diskzfs3 /dev/ada4
# glabel label diskzfs4 /dev/ada5
```

ce qui va nous permettre de travailler avec **/dev/label/diskzfs{1-4}**. Cette partie est complètement optionnelle et est à appliquer en fonction du goût de chacun. Si vous ne le faites pas, ne vous trompez pas dans la suite de cet article, **/dev/label/diskzfsX = /dev/adaX-1**.

L'un des principaux intérêts à ZFS lors d'une exploitation au quotidien consiste à ne pas se soucier de partitionnement ou même de formatage des disques physiques. On crée un pool, on insère les disques dedans et c'est fini. Commençons donc par créer le pool avec nos quatre disques :

```
# zpool create naspool raidz1 label/diskzfs1 label/diskzfs2 label/
diskzfs3 label/diskzfs4
```

Nous demandons à ZFS de créer un pool nommé **naspool** de type raidz1 (un raid 5 un peu amélioré) et constitué des disques diskzfs{1-4}.

Juste après avoir tapé ENTER nous trouvons à la racine un répertoire **/naspool** dans lequel nous pouvons commencer à stocker des données. Plus simple que ça je n'ai pas encore rencontré !

Il est bien entendu possible de créer des répertoires sous **/naspool**, mais ZFS nous offre la possibilité de créer des « filesystems » qui seront des sous-ensembles de naspool. Cette option nous permettra, entre autres, de monitorer séparément le stockage des divers filesystems sans nous embêter avec des points de montage. Un exemple est plus parlant :

```
# zfs create naspool/video
# zfs create naspool/Input
# zfs create naspool/homedirs
# zfs create naspool/musique
# zfs create naspool/publications
```

et après un peu de remplissage :

```
# df -h
Filesystem            Size    Used   Avail Capacity  Mounted on
naspool               790G     35k    790G     0%    /naspool
naspool/Input         796G    5.9G    790G     1%    /naspool/Input
naspool/homedirs      861G     70G    790G     8%    /naspool/homedirs
naspool/musique       847G     56G    790G     7%    /naspool/musique
naspool/publications  883G     92G    790G    10%    /naspool/publications
naspool/video         2.3T    1.6T    790G    67%    /naspool/video
```

on remarque que les 3To disponibles sont répartis sur les filesystems en fonction du remplissage de chacun d'eux. Dans cet exemple, il reste 790Go de disponible pour le pool, sachant que Input occupe 5.9Go, homedirs 70 Go, etc.

ZFS a aussi cela de remarquable qu'il intègre directement les commandes qui vont bien pour les exports NFS, en plus de tout un tas d'autres outils dont nous allons passer quelques-uns en revue. Une étude exhaustive des possibilités offertes par ZFS dépasserait largement le cadre de cet article et en nécessiterait un complet.

1. Changeons de point de montage

   À sa création, un filesystem se monte sous son pool d'origine. Par exemple :

```
# zfs create naspool/toto
```

   ira créer un filesystem sous **/naspool**. Si l'on désire changer ça et le faire se monter dans **/home/toto**, il suffit de le demander gentiment avec la commande qui va bien :

```
# zfs set mountpoint=/home/toto naspool/toto
```

   une fois le filesystem créé bien entendu. Une fois cette commande passée, si l'on crée un sous-filesystem dans **toto** :

```
# zfs create naspool/toto/mapomme
```

   le répertoire **mapomme** apparaîtra dans **/home/toto**.

2. Activons la compression

   ZFS est capable de gérer lui-même la compression et ceci est réglable filesystem par filesystem. D'où l'intérêt d'utiliser **zfs create** plutôt que **mkdir** en tous cas pour les racines. La commande est là aussi très simple d'utilisation :

```
# zfs set compression=on naspool/publications
```

3. Appliquons des quotas

   Pour réserver par exemple 100 Go pour les répertoires utilisateurs, nous pouvons demander à ZFS :

```
# zfs set quota=100G naspool/homedirs
```

4. Exports NFS

   ZFS est capable de maintenir lui-même un fichier **exports** dans **/etc/zfs/exports** qui est pris en compte en temps réel par **mountd**. Cela signifie que toute modification faite via **zfs** est immédiatement répercutée sans avoir besoin de relancer le démon. La commande permet aussi de limiter l'export NFS à un sous-réseau, de définir le mappage root et utilisateur, etc. Pour exporter **/naspool/video** sur le réseau 192.168.0.0/24 par exemple, il suffit de taper

```
# zfs set sharenfs="mapall=films,network=192.168.0.0/24" naspool/
video
```

5. Les snapshots.

La dernière fonctionnalité que nous allons voir ici concerne les snapshots. La création d'un snapshot se fait via la commande :

```
# zfs snapshot -r naspool/homedirs@today
```

qui crée un snapshot récursif (**naspool/homedirs** et tous les éventuels datasets sous-jacents) appelé « today ». Un système de backup automatisé sur une semaine peut tenir dans les quelques lignes suivantes :

```
# zfs destroy -r naspool/homedirs@7daysago
# zfs rename -r naspool/homedirs@6daysago @7daysago
# zfs rename -r naspool/homedirs@5daysago @6daysago
# zfs rename -r naspool/homedirs@4daysago @5daysago
# zfs rename -r naspool/homedirs@3daysago @4daysago
# zfs rename -r naspool/homedirs@2daysago @3daysago
# zfs rename -r naspool/homedirs@yesterday @2daysago
# zfs rename -r naspool/homedirs@today @yesterday
# zfs snapshot -r naspool/homedirs@today
```

La liste des snapshots est affichée par la commande :

```
# zfs list -t snapshot
```

et pour annuler toutes les modifications faites sur le filesystem jusqu'au snapshot d'hier :

```
# zfs rollback -r naspool/homedirs@yesterday.
```

La lecture de la manpage de ZFS est extrêmement instructive et alléchante. Il est fortement conseillé de la lire avant de mettre en place ses pools ZFS afin de créer les filesystems qui vont bien dans le but de se simplifier la vie avec les snapshots, de mettre en place la compression aux bons endroits ainsi que les exports NFS.

3 Installation de Samba

Malheureusement, même dans notre entourage, certaines personnes utilisent encore un système d'exploitation non libre qui ne sait pas nativement monter du NFS. Pour ces gens-là, nous sommes dans l'obligation d'installer ce bon vieux Samba qui permet à nos machines de parler CIFS dans le texte. Sous FreeBSD, l'installation de Samba s'effectue de la manière habituelle que nous commençons à bien connaître :

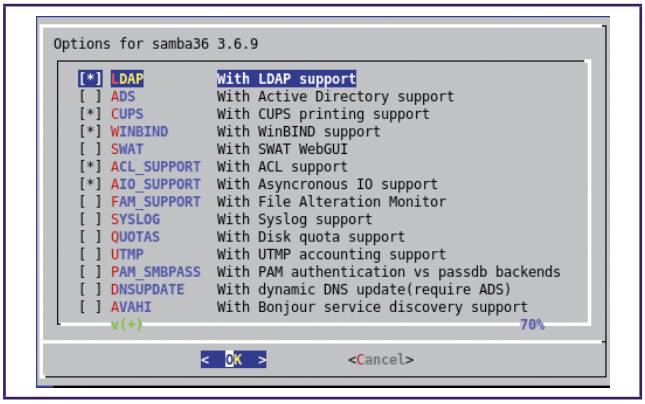


Figure 13

```
# cd /usr/ports/net/samba36
# make config
```

ce qui nous donne le bel écran ncurses ci-dessous (voir Figure 13).

Là je vous laisse choisir les options que vous souhaitez utiliser en fonction de votre infrastructure.

Ensuite :

```
# make install clean
```

va nous installer un beau Samba 3.6 opérationnel.

Je ne reviens pas sur la configuration de ce logiciel déjà étudiée dans ces colonnes, mais voici pour faire court un exemple de **smb.conf** immédiatement opérationnel. Pour mémoire, vu que l'on est passé par les ports, ce fichier doit être déposé dans **/usr/local/etc/samba**. Une fois cette copie faite, il faut relancer Samba avec la commande :

```
# /usr/local/etc/rc.d/samba restart
```

```
[global]
    workgroup = BIDOUILLE
    netbios name = Elrond
    server string = NAS server
    wins support = no
    wins server = 172.20.0.2

    log file = /var/log/samba/log.%m
    max log size = 1000
    syslog = 0
    panic action = /usr/share/samba/panic-action %d

    security = user
    encrypt passwords = true
    invalid users = root
    unix password sync = no
    socket options = TCP_NODELAY

[homes]
    comment = Home Directories
    browseable = yes
    writable = yes
    create mask = 0755
    directory mask = 0755

[Publications]
    comment = Publications
   browsable = yes
    path = /naspool/publications
    public = no
    writable = no
    write list = admin
    force group = films

[Videos]
    comment = Films
    browsable = yes
    path = /naspool/video
    public = no
   writable = no
```

```
    write list = admin
    force group = films

[Musique]
    comment = Musique
    browsable = yes
    path = /naspool/musique
    public = no
    writable = no
    write list = admin
    force group = films

[NasInput]
    comment = Input
    browsable = yes
    path = /naspool/Input
    public = yes
    writable = yes
    force group = films
```

Et voilà, rien de bien compliqué là non plus.

4 Deluge

4.1 Côté serveur

Il nous reste maintenant à installer et paramétrer notre client peer-to-peer dont le port se trouve dans **/usr/ports/net-p2p/deluge**. On commence par un **make config** afin de virer la partie GTK qui ne nous servira à rien puis l'incantation habituelle **make install clean**. Le daemon comme le serveur web étant écrits en Python, FreeBSD va commencer par rapatrier un Python 2.7 et le compiler avant de passer à la suite. C'est beau la résolution de dépendances, non ? :) Cependant, cela signifie qu'il vaut mieux rester devant son écran en y jetant un œil de temps en temps, car lesdites dépendances ont parfois besoin de notre opinion sur l'installation ou non de certaines options. À la suite d'un temps plus ou moins long – comptez quand même une bonne demi-heure – Deluge a fini de compiler et d'installer.

Contrairement à nos habitudes, la configuration de Deluge ne se fait pas dans un **/etc** quelconque, mais dans un sous-répertoire de l'utilisateur qui fera tourner les démons. En ce qui nous concerne, nous disposons d'un utilisateur nommé **films** qui fera parfaitement l'affaire. Avant d'aller effectuer les quelques réglages nécessaires, tournons-nous vers le fichier **/etc/rc.conf** auquel nous allons rajouter les trois lignes suivantes :

```
deluged_user="films"
deluged_enable="YES"
delugew_enable="YES"
```

Une fois ceci fait, mettons-nous sous le compte films avec un brave :

```
# su - films
```

et lançons **deluged** pour la première fois en mode debug :

```
[films@Fbnas ~]$ deluged -d -L debug
```

Profitons-en pour regarder si tout semble bien se dérouler, puis arrêtons le démon avec un Ctrl-C.

Direction maintenant le répertoire **/home/films/.config/deluge** dans lequel nous trouvons un paquet de fichiers qui ont été créés par Deluge lors de son premier lancement. Le premier fichier qui nous intéresse s'intitule **core.conf** et contrôle le fonctionnement du daemon Deluge, le bien nommé **deluged**. Commençons par aller modifier deux-trois choses :

1. La directive **move_completed_path** indique à Deluge où mettre les téléchargements terminés. Pour des raisons de simplicité, mettons-les dans **/naspool/video/Downloads**. En effet les plus gros fichiers sont les vidéos et un **mv** au sein du même filesystem – au sens ZFS du terme – est immédiat tandis qu'un **mv** vers un autre filesystem peut être long.
2. La directive **download_location** donne l'emplacement d'un répertoire temporaire où Deluge stocke les fichiers en cours de téléchargement. Pour ma part, j'ai choisi de créer un répertoire « Deluge » dans **/naspool/video** différent de **Download** dans le but d'éviter aux utilisateurs novices et/ou pressés d'essayer de regarder une vidéo pas encore totalement téléchargée.

Une fois ces modifications effectuées, nous pouvons démarrer **deluged** de manière classique (en tant que **root** bien sûr) :

```
# /usr/local/etc/rc.d/deluged start
```

puis vérifier qu'il tourne bien sous l'utilisateur **films** avec un :

```
# ps aux | grep deluge
```

Profitons donc du fait que nous sommes dans le bon répertoire pour aller préparer la suite et ajouter un compte et un mot de passe qui serviront au client lourd. Pour ce faire, il suffit d'ajouter la ligne suivante :

```
admin:deluge:10
```

dans le fichier **/home/films/.config/deluge/auth** ce qui a pour effet de créer le compte **admin** avec le mot de passe « deluge ».

4.2 Interface web

Après avoir réglé son compte au daemon principal, passons au petit serveur web. Commençons là aussi par lancer **deluge-web** à la main :

```
[films@Fbnas ~]$ deluge-web
```

puis connectons-nous sur notre NAS sur le port 8112 avec notre brouteur favori en HTTP (on pourra forcer le HTTPS dans la configuration), le mot de passe par défaut est « deluge ».

Le système nous suggère de le changer tout de suite, ce que nous refusons, car nous allons modifier quelques petites choses via l'interface graphique. Une première boîte de dialogue apparaît nous demandant à quel serveur deluge nous voulons nous connecter. Comme nous n'en avons qu'un, la réponse n'est pas trop dure à trouver. Ensuite, il faut aller cliquer sur le bouton « Preferences » dans la barre d'outils, ce qui nous ouvre une fenêtre comme ci-dessous dans laquelle l'onglet de gestion des téléchargements est activé :

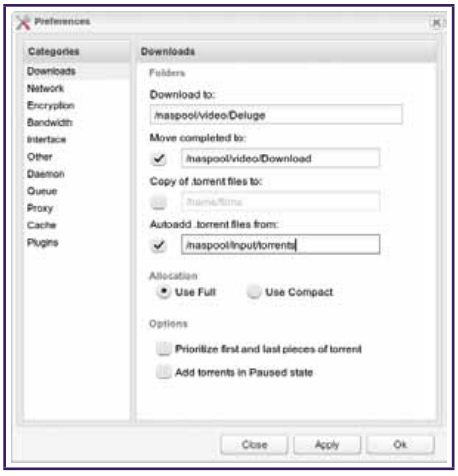


Figure 14

À cet endroit, nous devrions retrouver les chemins modifiés, si ce n'est pas le cas (certaines versions comportent un bug à ce niveau) il suffit de les re-saisir. Profitons-en pour activer l'autoadd, option très utile qui va aller scanner régulièrement le répertoire précisé à la recherche de tous les fichiers .torrent qu'il peut trouver. S'il en trouve, il les ajoutera automatiquement à la file de téléchargements.

Dans l'onglet « Network » il vous est loisible de fixer les ports d'entrée et de sortie ce qui est bien utile lorsque le serveur est derrière un firewall. Petit conseil, éviter de fixer des ports ostensiblement peer-to-peer genre 6881...

Option primordiale, la configuration d'encodage ressemble à ceci, avec l'option d'encodage qu'il est fortement conseillé de valider pour l'ensemble du flux :

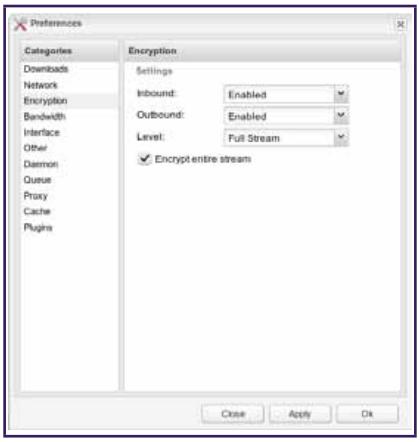


Figure 15

L'onglet « Bandwith » est simple à comprendre et il vous appartient de modifier les valeurs par défaut, ou pas.

C'est dans l'onglet « Interface » que l'on va pouvoir modifier le mot de passe par défaut, mais aussi passer toute notre connexion en HTTPS ce qui est plus prudent, surtout si vous comptez accéder à cette interface depuis l'extérieur.

Note

Si jamais vous oubliez ce mot de passe, la procédure de reset est la suivante :

- Arrêter deluge-web ;
- Supprimer web.conf ;
- Relancer deluge-web ;

et le mot de passe est remis à « deluge ».

Vous pouvez aussi modifier là le port d'écoute de **deluge-web**.

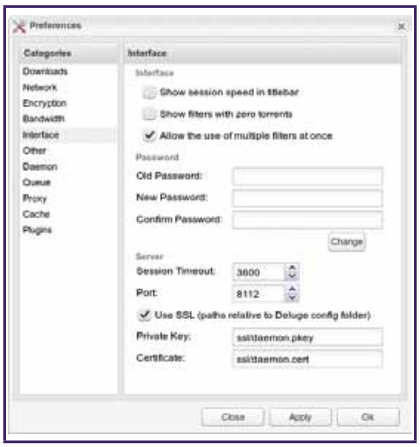


Figure 16

L'onglet « Other » regroupe des choses diverses et néanmoins variées qui n'ont pas leur place ailleurs comme la détection de l'arrivée d'une nouvelle version.

Dans l'onglet « Daemon », il est possible de modifier le port d'écoute de deluged, ce qui n'a pas grand intérêt sauf si vous déportez deluge-web sur une autre machine.

Les trois onglets « Queue », « Proxy » et « Cache » parlent d'eux-mêmes.

Pour finir l'onglet « Plugins » permet d'activer ou de désactiver les divers plugins existants. Le plugin **AutoAdd** me paraît faire double-emploi avec l'autoadd intégré, peut-être un reste d'un passé récent où cette fonctionnalité n'était pas présente par défaut ?

Une fois tous ces paramètres spécifiés, il est temps de vérifier que tout se déroule bien sans notre surveillance. Il est possible que vous n'ayez pas de script de démarrage pour **deluge-web** dans **/usr/local/etc/rc.d**. Il n'est pas très compliqué à créer, soit en se basant sur **/etc/rc.d/skeleton**, soit sur **/usr/local/etc/rc.d/deluged**. Dernière solution vous pouvez aller le recopier sur http://dev.deluge-torrent.org/wiki/UserGuide/InitScript/FreeBSD.

4.3 Client lourd Gtk+

Pour gérer son Deluge, un autre moyen consiste à installer le paquet **deluge-gtk** sur sa station de travail. Un brave :

```
# apt-get install deluge-gtk
```

devrait suffire.

Ensuite de quoi il ne reste plus qu'à lancer **deluge-gtk**, ce qui nous fait apparaître la fenêtre de connexion ci-dessous, pré-remplie avec comme présupposé que le serveur tourne sur la même machine, ce qui n'est pas notre cas ; cela explique la croix rouge à gauche de la ligne (voir Figure 17).

Figure 17

Nous allons donc ajouter le compte créé au début sur le serveur en cliquant sur « Ajouter » et en remplissant les champs avec l'adresse IP de notre serveur (192.168.0.101 chez moi) et les informations de connexion (admin/deluge) :

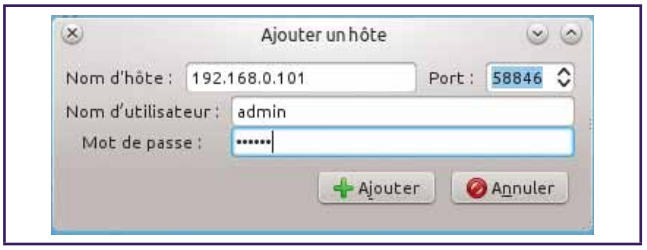


Figure 18

Inutile de changer le port par défaut, sauf si vous l'avez modifié côté serveur ou avec l'interface web. Un clic sur « Ajouter » et normalement notre serveur devrait être visible immédiatement comme ci-dessous :

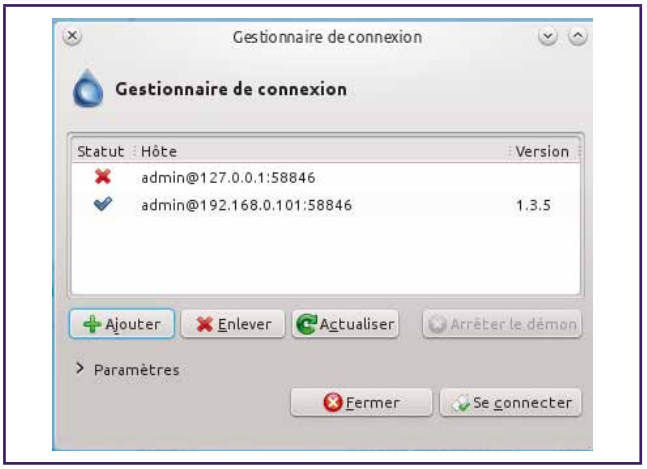


Figure 19

On voit bien que notre serveur a été trouvé grâce à la coche à gauche et à l'apparition de son numéro de version. Au besoin, un clic sur « Actualiser » permettra de rafraîchir les informations. Maintenant, on sélectionne notre serveur, on clique sur « Se connecter » et voilà.

Conclusion

Ce bref article n'a jamais eu pour vocation de faire de vous des pros de FreeBSD. Je ne suis moi-même qu'un débutant un peu éclairé et je supplie à genoux les geeks qui maîtrisent le sujet de ne pas me pendre avec un RJ45 si j'ai sorti quelques énormités. Tout ce que je peux vous dire c'est qu'un serveur de ce type fonctionne chez moi sans - trop - d'anicroches depuis plusieurs années et que son administration n'est pas si compliquée que ça. Pour les mises à jour, l'incantation est la suivante :

```
# portsnap fetch update
# portupgrade -a
```

ou pour ceux qui ont autre chose à faire qu'à rester devant l'écran, la version un poil plus risquée :

```
# portupgrade -batch -ak
```

qui évitera d'avoir à répondre aux questions concernant la configuration de certains paquets (**--batch**) et qui forcera la mise à jour des paquets même si certaines dépendances ne sont pas à jour (**-k**).

Toujours est-il qu'avec un peu de temps, de patience et d'huile de coude dans les neurones on finit par maîtriser suffisamment la bête pour y prendre un vrai plaisir et on se surprend à se demander pourquoi certaines fonctionnalités n'ont pas été (encore ?) implémentées sous Linux.

Il reste beaucoup de choses à faire pour finir ce NAS et la première que je vois ne concerne pas du tout le NAS, mais Firefox ou Chrome. Quelqu'un saurait-il développer une extension à ces navigateurs qui permettrait de télécharger les fichiers dans un répertoire différent en fonction de leur extension ? En clair, j'en ai marre de faire des :

```
$ mv ~/Download/*.torrent /naspool/Input/torrent
```

Si on pouvait dire au navigateur « Tu me télécharges tous les .torrent dans **/naspool/Input/torrent** et tous les PDF dans **~/Documents** et le reste dans **~/Download** » ça simplifierait bien la vie.

En tout cas, nous voici avec un joli NAS fonctionnel qui nous a permis d'apprendre un autre système d'exploitation libre. Vous avez toutes les billes maintenant pour lui rajouter des fonctionnalités genre IHM de gestion, support iSCSI ou autres. Bon courage... ■

BASE DE DONNÉES EMBARQUÉE DANS UN NAVIGATEUR

par p-e-g

Cela fait quelques années que je vends du logiciel et... soit je n'ai pas de chance, soit c'est le marché qui veut ça, mais pour une application client lourd (QT,Swing,GTK...) que j'écris, je vends 3 applications Web. Ces applications Web sont des applications style « desktop », mais vivantes, soit dans un navigateur Internet « standard », soit sur le navigateur d'un téléphone/tablette. Là où ça se complique, c'est qu'une application lourde interagit rapidement et « fonctionne » sans Internet... Deux trucs « difficiles » à mettre en place pour les applications Web.

Il devient assez complexe de faire comprendre à son client qu'une application Web 2.0 ne peut fonctionner sans Internet alors qu'une vieille application client lourd (Pre-Web 1.0 ?) le peut.

1 Problématique des applications Web

Comme dit dans le résumé de cet article, j'ai parfois l'impression que mon métier se résume maintenant à fabriquer des applications Web pour des clients. Le cas typique prend source dans une application client lourd écrite en C/C++ vivant sur les postes « client » dont le client souhaite une réécriture sur le Web.

Les besoins de cette réécriture peuvent être multiples et je retrouve régulièrement parmi ces derniers : la facilité de déploiement/de mise à jour. D'autres raisons moins avouables sont aussi la facilité à trouver des ressources de programmation Web ou encore l'envie d'avoir une application sur Internet pour faire « moderne ».

Pragmatiquement parlant, il semble qu'il soit assez facile de trouver des « profils » d'informaticiens assez jeunes (comprendre moins chers que des informaticiens expérimentés) programmant des sites Web que des « artistes » de l'informatique comprenant les rouages de la gestion mémoire ou du multithreading.

Là où ça cloche est dans le fait qu'une application Web est d'abord une application et non pas un site Web. C'est un fait que les deux utilisent un navigateur et du HTML, mais en pratique et surtout dans les pratiques de codage la différence s'arrête bien souvent là.

Pour essayer de construire une ligne de démarcation application/site Web, il est possible de prendre deux exemples.

Une application Web classique est par exemple le Webmail qui est le port d'un logiciel de messagerie « client lourd » (Thunderbird, Outlook...) vers une version « en ligne ».

Un autre exemple classique dans le cadre d'un « site Web » est par exemple un Intranet d'entreprise qui est d'abord un outil permettant de diffuser de l'information.

D'expérience (et faute de trouver une vraie ligne de démarcation), une des différences entre un site et une application Web est l'exigence des utilisateurs en terme de qualité d'interactivité.

Dans le cadre d'un site Web, il s'agit souvent d'une interaction assez faible ; l'utilisateur « navigue » dans l'application et passe plus de temps à lire des pages qu'à changer de page. Le temps d'un site Web est donc de l'ordre de la seconde, et un utilisateur peut attendre une seconde sans « râler ».

Pour une application Web, l'interaction est relativement forte et il y a une utilisation en continu de l'application. Les éditeurs de texte en ligne (CKEditor pour ne pas le nommer) demandent par exemple à pouvoir gérer pour un pigiste entraîné une quinzaine de caractères tapés à la seconde. Autrement dit, le temps des applications Web se doit d'être dans l'instantanéité. À titre personnel, je râle contre les Webmail qui ne permettent pas d'afficher les mails immédiatement, car mon Thunderbird lui, me permet de lire les mails instantanément.

Une autre ligne de démarcation, pour le coup évidente, est la fonctionnalité de mode déconnecté. Autant je ne peux pas lire un site web sans être connecté à Internet, autant une application (un traitement de texte par exemple) marche sans accès Internet. Il est donc légitime de penser qu'une application Web comme n'importe quelle application doit pouvoir fonctionner sans Internet.

Cette évidence est une des plus grosses difficultés de compréhension pour les clients. Avec un certain pragmatisme, les anciennes applications « client lourd » (écrites il y a entre 5 et 10 ans pour certaines) marchaient sans Internet, et la nouvelle application Web 2.0 (tadaam ! bienvenue dans le monde moderne) nécessite Internet.

En fait, contrairement à ce que pensent beaucoup de développeurs, le territoire français n'est pas complètement couvert de réseau 4G/Wifi permettant de se passer d'un mode déconnecté !

Un des challenges techniques des applications Web est donc de répondre positivement à ces deux critères (haute exigence de l'interactivité et mode déconnecté). En prenant en compte les différents navigateurs du marché (Firefox, Chrome, Opéra, IE), une bonne application Web se doit d'être réactive et déconnectée. C'est dans cette double contrainte que naît la technique que je décris dans cet article.

Le cas pratique provient d'une application Web fortement chargée et déployée avec succès qui sert des navigateurs Web se trouvant sur la France entière. Dit comme cela ça a l'air simple, un bon serveur bien configuré, et normalement ça marche. Néanmoins, la France c'est grand, et cela couvre aussi des régions mal connectées à Internet (débit faible) et des régions lointaines (débit élevé, mais latence réseau forte).

Pour la petite histoire, le serveur applicatif est sur Brest et le problème est apparu lorsque des navigateurs d'outremer ont commencé à utiliser l'applicatif. La configuration serveur/réseau/machine cliente provoque un délai de 2 secondes entre le navigateur et le serveur.

Or deux secondes entre un serveur et un navigateur sont dans le cadre des applications Web une éternité et même si certaines latences peuvent être cachées avec des techniques Ajax, cela est insuffisant.

2 Le HTML5

Donc dans le cas d'une application Web, avant le HTML5, la gestion d'un mode déconnecté ainsi que le respect des latences entre le navigateur et un serveur Internet étaient quasi impossibles. En effet, afin de pouvoir gérer ces contraintes, il est nécessaire de pouvoir sauvegarder des données sur le poste client. Ces données sauvegardées servant de cache applicatif pour éviter des appels serveur (et respecter des latences correctes).

Le pattern d'une telle technique est trivial : si mon application cherche à calculer un résultat alors :

1. Je regarde dans mon cache applicatif si le résultat de ma fonction n'est pas déjà connu. Si oui, je renvoie le résultat. Dans ce cas, je reste dans le navigateur et l'applicatif est rapide.
2. Sinon, je fais une requête au serveur Web (et c'est lent !), je mets le résultat de la requête dans mon cache applicatif et je renvoie le résultat.

L'idée générale étant que je ne fais des requêtes (potentiellement avec de la latence) au serveur Web que si c'est nécessaire. D'un point de vue de génie logiciel, le serveur héberge une base de données avec un schéma relationnel et un ensemble de requêtes SQL. Il est assez utile que le cache applicatif contienne à peu près le même schéma de base de données et globalement les mêmes requêtes SQL. Ainsi, le cache applicatif et le serveur possèdent à peu près le même code, ce qui permet de mutualiser, et l'effort de développement et l'effort de test.

Le hic c'est qu'avant le HTML5, la création d'un cache applicatif était assez complexe à mettre en place/ou limiter à certains navigateurs. Une des promesses de cette norme est/était de répondre à cette problématique. C'est à ce niveau que pourrait se finir cet article avec un truc du genre avant le HTML5 c'était l'enfer et après c'est le bonheur... ils se marièrent et eurent beaucoup d'enfants.

En fait, non pas de bol, avant le HTML5, c'était l'enfer et après c'est un peu le bordel, :(car il y a plusieurs techniques pour faire un cache applicatif.

Le Web Storage (parfois décrit comme DOM Storage) **[1]** qui est un tableau associatif et qui fonctionne dans la plupart des navigateurs. Deux versions de cette fonctionnalité sont accessibles : soit un Web Storage de session (dont la durée de vie est celle de la fenêtre de l'application Web), soit un Web Storage local qui est persistant même lorsque le navigateur est fermé.

En pratique, il est possible de sauvegarder des couples clefs/chaînes de caractères dans ce tableau associatif. L'avantage de cette technologie est qu'elle est présente et avec une implémentation quasi identique dans la plupart des navigateurs. Néanmoins, la taille de ce tableau associatif diffère entre les navigateurs et ne dépasse pas la dizaine de mégas. Or, en pratique la construction d'un cache applicatif pour mes applications nécessite plus de 10 mégas et de ce fait, il est possible d'utiliser le Web Storage pour sauvegarder des « petits » trucs, mais pas pour construire un cache complet. L'autre désavantage est que les données sont sauvegardées sous la forme d'un dictionnaire. Or en pratique, côté serveur, les données sont sous forme relationnelles et il serait pratique que le cache applicatif respecte le schéma relationnel et en particulier le « requêtage » SQL.

L'autre solution est d'utiliser les API Web SQL Database. Cette solution est disponible nativement sur Chrome, Opéra et Safari (ou via un add-on sur Firefox, ce qui implique « non présente » nativement).

Le premier défaut de cette solution est son manque de disponibilité. Il est en effet difficile de bâtir une application Web sans prendre en compte Firefox et Internet Explorer (ce qui représente à minima 50% du marché). En outre, l'organisme de standardisation W3C a cessé son travail sur le Web SQL Database en novembre 2010 **[2]**. L'argument avancé est que la totalité des implémentations est basée sur la base de données SQLite. Ce manque d'implémentations différentes empêche le W3C de fabriquer un standard. J'avoue à titre personnel ne pas avoir compris l'argument, mais ce qui est sûr, c'est que si la normalisation de cette API est abandonnée... elle ne risque pas de survivre longtemps à l'épreuve du temps.

Néanmoins, ce jeu d'API permet de faire des requêtes SQL côté navigateur dans la base de données embarquée. Il est évident que cette solution est donc la « mieux » adaptée pour fabriquer des caches applicatifs.

Enfin, la dernière solution est IndexedDB **[3]** qui est aussi une solution à base de tableaux associatifs. Comparée à la solution de Web Storage qui propose un tableau associatif, la solution IndexedDB offre autant de tableaux associatifs (appelés ObjectStore) que voulu, organisés en « base de données ». Nous avons donc la potentialité pour une application de créer une base de données et dans cette base nous pouvons créer autant de tableaux associatifs que nous voulons. Enfin IndexedDB propose deux jeux d'API, une synchrone et une asynchrone. La solution IndexedDB semble séduisante (bien qu'ici aussi nous n'ayons pas de modèle relationnel), mais est insuffisante. Le jeu d'API synchrone n'est pas implémenté et surtout, Safari n'est pas supporté. Le non support de ce navigateur est gênant pour les plateformes Mac OS dont c'est le navigateur par défaut.

Si nous prenons ces trois technologies, nous pouvons nous amuser à tracer ce petit tableau fonctionnel afin de regarder ce qu'il est possible d'utiliser pour faire un cache applicatif :

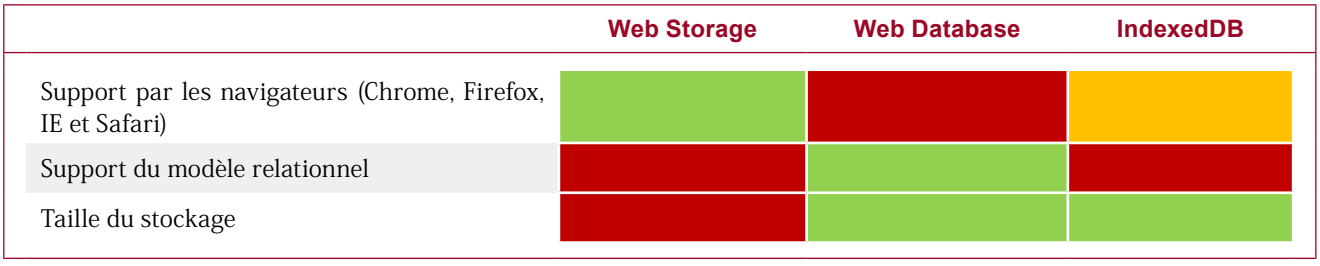
Les différentes couleurs doivent se lire comme cela :

- **Rouge** : la fonctionnalité n'est pas suffisante pour mes applications afin de faire un bon cache applicatif ;
- **Orange** : la fonctionnalité est envisageable pour faire un cache applicatif, mais il va falloir ramer pour que cela marche ;
- **Vert** : la fonctionnalité est bonne pour un cache applicatif.

De ce tableau, il est visible que la moins mauvaise des solutions semble être IndexedDB. Néanmoins, le choix de cette méthode pour construire un cache applicatif n'est pas enchanteur. Il faudrait pour cela prendre en compte que Safari n'a pas d'implémentation et le fait qu'il va falloir construire un cache sans modèle relationnel. Néanmoins, dans nos rêves les plus fous, une technologie de cache applicatif :

- Doit être supportée sur la plupart des navigateurs du marché. Ceci est essentiel afin que l'application Web puisse fonctionner « partout ».
- Supporter le modèle relationnel et si possible le langage SQL. Cela permettrait d'utiliser quasiment le même code côté serveur que côté client. Ceci est essentiel pour mutualiser l'effort de développement/test entre le cache et le serveur.
- Pouvoir stocker des tailles de données relativement importantes. Un cache de données doit contenir des données du serveur, et il s'agit donc d'une « miniature » des données serveur. Dans mon cas, il est essentiel de pouvoir sauvegarder plus de 10 mégas octet de données dans le cache.

C'est pour répondre à ce triple besoin que la technique suivante a été mise en place.



3 Emscripten, SQLite et GWT

Emscripten **[4]** est un compilateur de code C/C++ vers la machine virtuelle Javascript. Cet utilitaire assez magique permet via LLVM **[5]** de réutiliser les librairies C/C++ dans un navigateur. Emscripten fournit l'équivalent de make et configure (respectivement emmake et emconfigure) ainsi qu'un certain nombre de librairies dont SDL qui peuvent être utilisées directement en JavaScript.

En particulier, parmi les ports d'applicatifs faits avec Emscripten, on trouve la base de données SQLite **[6]**. Nous avons donc un moteur de base de données écrit en JavaScript et utilisable sur tous les navigateurs.

Les APIs de ce moteur sont simplissimes et synchrones, injectent en JavaScript un objet de type SQL autorisant :

1. Une méthode **Open** qui renvoie soit une nouvelle instance de base de données (si aucun paramètre n'est passé), soit ouvre une base de données (si un tableau d'octets est passé en paramètre) ;
2. Une fonction **Exec** qui prend une requête SQL en paramètre et renvoie un objet JSON comme résultat ;
3. Enfin une méthode **Close** qui ferme la base de données ;
4. Une fonction **ExportData** qui renvoie un tableau d'octets non signé représentant la base de données.

Créant mes applications Web avec GWT, je me suis permis d'encapsuler la librairie Sqlite version JavaScript dans du code Java GWT. Pour ceux pour qui l'encapsulation du JavaScript dans le GWT est un mystère, je laisse ici quelques lignes.

3.1 JSNI et l'encapsulation de Sqlite dans GWT

Les applications Web et GWT (Google Web Toolkit) sont en pratique intimement liées. En effet, GWT, SmartGWT, GXT ou encore Vaadin sont des librairies très utilisées pour créer des applications Web du fait :

- Qu'elles autorisent la programmation des applications Web dans le langage Java ;
- Sont indépendantes des navigateurs utilisés ;
- Permettent l'utilisation aisée des outils de développement Java (débuggeur, test unitaire) ;
- Offre des performances élevées.

Néanmoins, il est assez évident que même si GWT est principalement utilisé avec le langage Java, il est légitime de vouloir utiliser dans son programme Java des librairies JavaScript.

L'interface JavaScript Native Interface **[7]** sert justement à cela.

Cette interface consiste à définir une classe Java, par exemple SQLiteFactory qui permet de créer une instance de SQL.

Ces classes finales doivent se doter d'un constructeur non public et définissent les méthodes « embarquant » du JavaScript comme étant natives.

Nous avons écrit que la version Javascript de Sqlite injecte dans la fenêtre de l'application un objet SQL. Cet objet de méthode **open** prenant respectivement soit un paramètre vide (nous construisons une nouvelle base), soit un tableau d'octets (nous lisons une base de données).

Notre classe SQLiteFactory a donc pour but d'encapsuler ces deux méthodes JavaScript (la seule complexité à voir est que l'objet fenêtre est dans JNSI représenté par la variable **$wnd**) :

```
public final class SQLiteFactory {

  protected SQLiteFactory()
  {

  }
  public static native  SQLite createSQLite(Uint8ArrayNative s)
  /*-{

    return $wnd.SQL.open(s);
  }-*/;
  public static native  SQLite createSQLite()
  /*-{
    return $wnd.SQL.open();
  }-*/;
}
```

Nos deux fonctions **createSQLite** se contentent donc d'invoquer les méthodes JavaScript **open** qui renvoient des objets JavaScript représentant la base de données SQLite. C'est pour cela que nous renvoyons comme objet une instance de classe SQLite étant l'objet GWT **JavaScriptObject**. Ce dernier est « l'astuce » utilisée pour communiquer entre le monde des objets JavaScript et le monde des objets Java.

De ce fait, nous encapsulons dans notre objet SQLite, et de la même manière que pour l'objet **SQLiteFactory**, le reste des fonctions JavaScript à savoir :

- **Close** qui permet de fermer une base de données ;
- **Exec** qui renvoie un objet JavaScript (que nous transformons en objet JSON) ;
- **ExportData** qui renvoie la base sous forme d'un tableau d'octets.

```
public final class SQLite extends JavaScriptObject  {

  protected SQLite()
  {


  public native final void close()
  /*-{
  this.close();
}-*/;
```

```
  public final JSONObject exec(String sql)
  {
    return new JSONObject(exec0(sql));
  }
    private native final JavaScriptObject exec0(String s)
  /*-{
   return this.exec(s);
}-*/;
  public native final Uint8ArrayNative getDatabase()
  /*-{
   return this.exportData();
}-*/;

}
```

Ce jeu de deux classes nous permet donc d'écrire en Java un peu de code de test :

```
//Création d'une base de données vide
SQLite sqlite=SQLiteFactory.getSQLite();
// Création d'une table A avec une clef primaire et une valeur
sqlite.exec("CREATE TABLE A (KEY INTEGER PRIMARY KEY AUTOINCREMENT, VAL
INTEGER)");
for (int i=0;i<2;i++) sqlite.exec("insert into  A(VAL) values(3)");
//Impression dans le navigateur des valeurs insérées dans A
RootPanel.get().add(new Label((sqlite.exec("select * from A ")).toString()));

Ce qui donne bien dans le navigateur deux tuples (0 et 1) :
{
  "0": [{"column": "KEY",  "value": "1"}, {"column": "VAL","value": "3"}],
  "1": [{"column": "KEY", "value": "2"}, {"column": "VAL","value": "3"}]
}
```

3.2 Une base de données embarquée

Nous avons maintenant une base de données écrite en JavaScript que nous pouvons embarquée dans nos applications Web. Cette base de données a pour principal intérêt :

- D'être par construction portable sur tous les navigateurs ;
- Supportant, et le modèle relationnel, et le « requêtage » SQL (il s'agit d'une base de données SQLite).

Néanmoins, cette base de données n'est pas persistante. Autrement dit, si la fenêtre est fermée, alors la base de données est perdue. Nous allons donc maintenant travailler sur le fait de pouvoir rendre persistante cette base de données en utilisant respectivement pour les navigateurs Safari/Opera le jeu d'API WebSQLDatabase et pour les navigateurs Chrome/Firefox/IE la technologie IndexedDB. En effet, cette version de Sqlite marche assez bien, mais nous souhaitons éviter la destruction de la base lors de la fermeture du navigateur.

L'idée générale est d'utiliser ces techniques pour sauvegarder la base SQLite en utilisant la méthode **`getDatabase`** et de la restaurer en passant via la méthode **`createSqlite`**.

4 Couche de stockage

Dans cette partie, nous allons définir le jeu de méthodes qui va nous permettre d'implémenter un service afin de stocker notre base SQlite aussi bien avec les API WebSQLDatabase que les API IndexedDB.

Afin de rester simple, nous définissons 4 méthodes :

- **`dropAllDatabase()`** qui détruit toutes les bases de données Sqlite que nous aurions pu créer sur le navigateur ;
- **`dropDatabase(String name)`** qui détruit une base de données Sqlite **`name`** du poste client ;
- **`getDatabase(String name,DatabaseOpenHandler handler)`** qui charge une base de données **`name`** depuis le navigateur. L'interface **`DatabaseOpenHandler`** est une interface **`CallBack`** offrant une méthode **`onSucceed`** prenant un événement **`DatabaseOpenEvent`**. Cet événement autorise la récupération d'une instance de notre classe SQLite ;
- **`InsertOrCreate(String name,SQLite base)`** sauvegarde dans la couche de stockage une base de données Sqlite avec un nom « Name ».

Enfin, nos deux implémentations s'appelleront simplement **`WebSQLDatabaseStoreService`** et **`IndexedDBStoreService`**.

Ces différentes méthodes peuvent être utilisées comme dans le petit code exemple suivant :

```
IndexedDBStoreDataService dataService=new IndexedDBStoreDataService();
dataService.getDatabase("MyDatabase",new DatabaseOpenHandler(){
@Override
  public void onSucceed(DatabaseOpenEvent event) {
    SQLite sqlite=event.getSQLite();
    //Nous pouvons travailler ici avec la base de données SQLite "sqlite"
    dataService.insertOrCreateDatabase("MyDatabase", sqlite);
    }
  });
```

4.1 WebSQLDatabase

L'API WebSQLDatabase va donc nous permettre de sauvegarder notre base de données sur les navigateurs Opera, Safari, voire Chrome. Par chance, les WebSQLDatabase disposent d'une encapsulation assez propre dans la librairie gwt-mobile **[8]**.

Enfin, d'après le W3C, les implémentations de cette API utilisent de façon sous-jacente un moteur SQLite, chose dont nous allons abuser par la suite. Notre abus va consister à créer une base de données SQLite appelée **`MyDatabase`** contenant une table **`MyDatabaseSQLite`** composée d'un identifiant entier, d'un nom et d'un champ BLOB (pour Binary large Object).

Le truc général est le suivant : lorsqu'il y aura la volonté de lire/écrire une de nos bases SQLite JavaScript (appelons la **`MySqliteJS`** pour plus de lisibilité), nous allons lire/insérer dans **`MyDatabaseSQLite`** :

- Le hashCode du nom de la base comme identifiant de ligne ;
- Le nom de la base ;
- La base de données elle-même transformée en Base64.

La situation finalement revient à utiliser un moteur SQLite mis à disposition de la technologie WebSQLDatabase pour embarquer des bases de données SQLite JavaScript.

La seule question qui reste en suspens est pourquoi nous transformons la base de données en Base64. En fait, je ne sais pas s'il s'agit d'un bug ou pas, mais le passage de JavaScript (pour Sqlite) à GWT (pour gwt-mobile) provoque des dysfonctionnements sur la base de données codée sur un tableau d'octets. La transformation du tableau d'octets en Base64 a surtout pour but de manipuler la structure sous forme de chaîne de caractères.

4.1.1 Comment Insérer une ligne dans une table en WebSQLDatabase

Commençons simplement en définissant la méthode **insertOrCreateDatabase** qui prend en paramètre le nom de la base de données et la base de données telle que définie dans la spécification suivante :

```
public  void insertOrCreateDatabase(final String name,final SQLite sqlite)
```

Il est possible d'ouvrir une base de données Sqlite avec gwt-mobile en utilisant la méthode **openDatabase** :

```
Database db = Database.openDatabase("MyDatabase", "1.0", "", 50000);
```

Cette méthode prend en paramètres le nom de la base SQlite à ouvrir, un numéro de version de cette base (ici, la version 1) ainsi que la taille estimée de la base. Il est à noter que la notion de base estimée n'est pas très claire dans les spécifications de la W3C, mais nous supposons que notre base **MyDatabase** utilisera 50 Mo.

Enfin, il est possible d'appliquer à un objet **Database** des transactions SQL pilotées par des méthodes callBacks **onTransactionStart**, **onTransactionFailure** et **onTransactionSuccess** (représentant le fait qu'une transaction démarrée, a échoué ou est un succès).

Commençons donc une transaction :

```
db.transaction(new TransactionCallback(){
     @Override
     public void onTransactionStart(SQLTransaction transaction) {
     //La transaction démarre
     }
     @Override
     public void onTransactionSuccess() {
     //La transaction est un succès
                                  }
     @Override
     public void onTransactionFailure(SQLError error) {
     //la transaction a échoué
                                  }});
```

La méthode **onTransactionStart** fournit une instance d'objet **SQLTransaction** autorisant une méthode exec prenant une chaîne de caractères SQL, des paramètres potentiels de la requête et ici aussi une callback permettant de piloter la méthode **executeSql**.

Un premier exemple simple est l'appel suivant permettant de créer notre table **MyDatabaseSQLite**. Nous ne passons pas de paramètres à la requête SQL (la valeur est nulle), et nous ne cherchons pas à piloter le résultat de cette requête.

```
tx.executeSql("CREATE TABLE IF NOT EXISTS MyDatabaseSQLite ("
                  + "id INT8 NOT NULL PRIMARY KEY AUTOINCREMENT,"
                  + "name VARCHAR, value BLOB)",null);
```

Enfin, nous insérons dans la table **MyDatabaseSQLite** notre base de données SQLite JavaScript sqlite.

Afin de faire cette action, nous transformons la base de données SQLite JavaScript en tableau d'octets, puis le tableau d'octets est transformé en chaîne de caractères et enfin, nous encodons la chaîne de caractères en Base64 :

```
  String toInsert=Base64Coder.encodeString(JsUtils.
stringFromArrayBuffer(sqlite.getDatabase().buffer()));
```

La classe Base64Coder permet d'encoder/décoder une chaîne de caractères en Base64 et n'est pas une classe native de GWT. Néanmoins, de nombreuses implémentations existent telle que **[9]**.

Nous allons maintenant utiliser la transaction afin d'insérer notre base Sqlite JavaScript Base64 (ouf !) dans la table **MyDatabaseSQLite** :

```
tx.executeSql("INSERT INTO MyDatabaseSQLite values(?,?,?)"
,new Object []{name.hashCode(),name,toInsert},
new StatementCallback(){
@Override
public void onSuccess(SQLTransaction transaction, SQLResultSet
resultSet) {
//L'insertion est un succès
}
@Override
public boolean onFailure(SQLTransaction transaction, SQLError error)
{
//L'insertion est un echec
return false;
 }
});
```

4.1.2 Comment ligne une ligne dans une table en WebSQLDatabase

Maintenant que nous avons pu sauvegarder une base de données Sqlite JavaScript en WebSQLDatabase, il serait heureux de pouvoir restaurer cette base de données.

C'est le but de la méthode :

```
public  void getDatabase(final String name,final DatabaseOpenHandler handler)
```

Cette méthode prend en paramètre le nom de la base et la restaure via une interface CallBack handler.

Ici aussi, nous devons ouvrir la base de données **MyDatabase** et potentiellement créer la table **MyDatabaseSQLite**

```
Database db = Database.openDatabase("MyDatabase", "1.0", "", 50000);
b.transaction(new TransactionCallback() {
       public void onTransactionStart(SQLTransaction tx) {
          tx.executeSql("CREATE TABLE IF NOT EXISTS MyDatabaseSQLite ("
                  + "id INT8 NOT NULL PRIMARY KEY ,"
                  + "name VARCHAR, value BLOB)",null);
```

L'objet **tx** permettant de faire des requêtes SQL, nous l'utilisons pour faire un sélect dans la table :

Ledit **select** étant paramétré avec le hashCode du nom de la base, et dans cet exemple nous allons faire quelque chose d'intelligent dans la méthode **onSuccess**... à savoir récupérer un résultat potentiel

```
tx.executeSql("select value from  MyDatabaseSQLite where id=?",new
Object []{name.hashCode()},
             new StatementCallback<JavaScriptObject>(){ @Override
        public void onSuccess(SQLTransaction transaction,
           SQLResultSet<JavaScriptObject> resultSet) {
```

Le **ResultSet** de la méthode **onSuccess** contient la séquence des objets JavaScript représentant potentiellement la base de données SQlite JavaScript-Base64. Il y a deux cas de figure, soit cette séquence est vide, soit elle contient un élément.

En fonction de ces deux cas, nous invoquons la bonne méthode de SQLiteFactory :

```
for (JavaScriptObject row : resultSet.getRows()) {
// Nous sommes dans le cas où une
// base de données a été trouvée
// Nous récupérons alors une base dans
// la colonne " value " au format Base64.
// Il faut alors récupérer la valeur de cette
// colonne en chaîne de caractères, décoder la
// chaîne en base64 et transformer cette chaîne
// en tableau d'octets.
Uint8ArrayNative read=
 =Uint8ArrayNative.create(JsUtils.arrayBufferFromString(Base64Coder.
decodeString((new JSONObject(row).get("value").isString().stringValue()))));
handler.onSucceed(new SQLiteDatabaseOpenEvent(SQLFactory.createSQLite(read)));
return; }
 if (resultSet.getRows().getLength()==0)
handler.onSucceed(new SQLiteDatabaseOpenEvent(SQLFactory.createSQLite()));
```

4.2 IndexedDB

IndexedDB est donc notre moteur de stockage pour les navigateurs Firefox, IE et Chrome. Cette norme assez récente ne dispose que d'API asynchrones, et manque de chance, ne dispose pas d'encapsulation GWT récente et fonctionnelle. Nous allons donc principalement travailler en JavaScript et nous encapsulerons nos méthodes JavaScript dans du code GWT.

IndexedDB, comme dit en introduction, est orienté autour de deux concepts :

- la base de données en elle-même qui est un regroupement de dictionnaires ;
- le dictionnaire proprement dit appelé ObjectStore dans les spécifications du W3C.

Nos acteurs étant présentés, nous allons procéder dans la même idée qu'avec WebSQLDatabase à savoir :

- créer une base de données ;
- créer un ObjectStore MyDatabaseSqlite et enfin, lire/écrire des bases de données.

4.2.1 Création de la base de données

La création de la base de données **MyDatabaseSqlite** ainsi que l'ObjectStore n'est pas aussi simple que pour les WebSQLDatabase. L'histoire commence par l'injection dans la fenêtre du navigateur d'un objet IndexedDB qui peut être préfixé soit « rien », soit **webkit** (pour Chrome), soit **moz** (pour Mozilla), soit **ms** (pour Microsoft).

Nous rationalisons le tout dans un objet **indexedDB** via :

```
$wnd.indexedDB = $wnd.indexedDB || $wnd.mozIndexedDB || $wnd.
webkitIndexedDB || $wnd.msIndexedDB;
```

Enfin, nous pouvons appeler la fonction **open** qui prend trois paramètres, le nom de la base de données et un numéro de version de base. Le numéro de version de la base est une information importante qui va conditionner l'appel ou pas de la fonction javascript **onupgradeneeded**. Cette fonction est un bon endroit pour créer notre ObjectStore et est appelée si la dernière version de base connue est inférieure au paramètre passé de version de base.

Enfin, il est possible et même souhaité de fournir lors de l'ouverture de la base une fonction JavaScript **onsucess** qui est appelée, soit directement si la version de la base passée en paramètre est identique au numéro de version de base connue, soit juste après la fonction **onupgradeneeded**.

Intéressons-nous à la méthode **onupgradeneeded** qui va nous autoriser à construire et définir un ObjectStore. Afin de le construire, nous avons besoin de deux informations, à savoir le nom de l'ObjectStore mais aussi la structure au format JSON contenant :

- le nom d'une clef primaire qui est appelée keypath ;
- une information pour dire si cette keypath est auto-générée par IndexedDB ou gérée par le programmeur.

Enfin, lors de la création de l'ObjectStore il est possible de définir un certain nombre d'index sur les objets que nous stockerons.

Dans notre cas, nous voulons sauvegarder dans un ObjectStore **MyDatabaseSQLite** qui possède trois attributs :

- **id** qui est notre keypath ;
- **name** qui est le nom de la base SQlite JavaScript ;
- **value** qui est la base de données Sqlite proprement dite.

Ces fonctions sont donc enchaînées ainsi :

```
//Ouverture de la base MyDatabase
var req = $wnd.indexedDB.open("MyDatabase",1);
req.onupgradeneeded = function (evt) {
  //evt.currentTarget.result contient l'objet permettant de créer un ObjectStore
// via la fonction createObjectStore
  var store = evt.currentTarget.result.createObjectStore(MyDatabaseSQLite,
{ keyPath: 'id', autoIncrement: false });
  // le nom de base Sqlite JavaScript
store.createIndex('name', 'name', { unique: false });
      //La base proprement dite
store.createIndex('value', 'value', { unique: true });
   };
```

4.2.2 Comment lire une base de données

La lecture d'une base de données SQLite JavaScript dans IndexedDB se fait dans la fonction JavaScript onsuccess lors de l'ouverture de la base MyDatabase.

La démarche est de récupérer l'ObjectStore **MyDatabaseSQLite**, d'utiliser la valeur de haschage du nom de la base SQLite pour récupérer un tableau d'octets représentant la base.

En premier lieu, il faut récupérer un objet IndexedDB Transaction qui va permettre de récupérer l'objet ObjectStore. Une des particularités de cet API est que pour obtenir une « Transaction », il faut préciser sur quel ObjectStore on souhaite travailler.

Notre cas est simple, car nous travaillons sur **MyDatabaseSQLite** et nous souhaitons récupérer l'ObjectStore du même nom :

```
req.onsuccess = function (evt) {
       var db = this.result;
       var trans = db.transaction("MyDatabaseSqlite", "readwrite");
    var store = trans.objectStore("MyDatabaseSqlite");
```

Enfin, il est possible d'appeler la méthode **get** sur l'objet store permettant de construire un objet requête. Comme avec WebSQLDatabase, nous nous servons du code hashage (méthode **hashCode** de la classe Object) du nom **name** de la base. Ce qui donne en JNSI :

```
var request = store.get( name.@java.lang.Object::hashCode());
```

L'objet **request** peut lui-même avoir une fonction **onsucceed** lui permettant de récupérer le résultat et d'invoquer en JNSI **SQLFactory.createSQLite** avec ou sans tableau d'octets :

```
  request.onsuccess = function(event) {
// si la requête contient quelque chose
// alors on invoque la méthode onSucceed de
// l'interface DatabaseOpenHandler avec
// une instance SQLiteDatabaseOpenEvent prenant
// en paramètre l'invocation de SQLFactoy avec un
// tableau d'octet.
      if (request.result!=undefined)

{
handler.@com.neuresys.database.databaseapi.DatabaseOpenHandler::onSucceed(Lc
om/neuresys/database/databaseapi/DatabaseOpenEvent;)
  (@com.neuresys.database.databaseimpl.jssqlite.SQLiteDatabaseOpenEvent::new(
Lcom/google/gwt/typedarrays/client/Uint8ArrayNative;)
  (@com.neuresys.database.databaseimpl.jssqlite.SQLiteFactory::createSQLite(L
com/google/gwt/typedarrays/client/Uint8ArrayNative;)(request.result.value)));
}else
handler.@com.neuresys.database.databaseapi.DatabaseOpenHandler::onSucceed(Lc
om/neuresys/database/databaseapi/DatabaseOpenEvent;)
 (@com.neuresys.database.databaseimpl.jssqlite.SQLiteDatabaseOpenEvent::new(L
com/google/gwt/typedarrays/client/Uint8ArrayNative;)
  (@com.neuresys.database.databaseimpl.jssqlite.SQLiteFactory::createSQLite()
()));
}
```

Nous n'expliciterons pas l'implémentation de **insertOrCreateDatabase** permettant d'écrire une base de données SQLite avec IndexedDB. En effet, l'écriture d'une base se fait selon le même pattern que celle de la lecture d'une base, si ce n'est que l'appel à la fonction « get » prenant un entier et renvoyant une base est remplacé par l'appel à la fonction « put ». Cette dernière prend un entier (le code de hachage du nom de la base), une base SQLite et écrit cette dernière dans l'ObjectStore.

Conclusion

Nous avons mis en place une librairie permettant d'embarquer côté navigateur une base de données SQLite écrite en JavaScript. Cette base obtient une propriété de persistance via, soit WebSQLDatabase, soit IndexedDB.

Ces deux propriétés assurent un support important pour la plupart des navigateurs du marché. Enfin, fonctionnellement, le but de la manipulation est d'autoriser à pouvoir mettre en place un cache applicatif côté client possédant toutes les propriétés d'un cache base sur le modèle relationnel (capacité de faire des requêtes, structure homogène à une structure serveur...).

Enfin, pour les plus curieux, le code de la librairie est disponible sur **[10]**. ■

Références

[1] WebStorage : en.wikipedia.org/wiki/Web_storage

[2] WebSQLDatabase : en.wikipedia.org/wiki/Web_SQL_Database

[3] IndexedDatabase : en.wikipedia.org/wiki/Indexed_Database_API

[4] Emscripten : https://github.com/kripken/emscripten

[5] LLVM: llvm.org

[6] SqliteJS : https://github.com/kripken/sql.js

[7] JSNI : https://developers.google.com/eclipse/docs/gwt_jsni

[8] gwt-mobile : code.google.com/p/gwtmobile/

[9] Base64 : http://code.google.com/p/gwt-crypto/

[10] code de l'article : code.google.com/p/gwt-storage-html5/

INTERCEPTION DE SIGNAL AVEC DUMP DE LA PILE D'APPEL

par Thierry GAYET

Juste avant qu'un programme se termine « anormalement », il peut être intéressant d'afficher la pile d'appel. Nous allons voir comment mettre cela en œuvre.

1 Définition

Une « backtrace » est une liste des appels de fonctions qui sont actuellement actifs dans un thread ou un processus donné. La façon habituelle d'inspecter la trace d'un programme est d'utiliser un débogueur externe tel que gdb.

Exemple de backtrace obtenu à partir d'un core dump généré après un segfault :

```
$ ulimit -c unlimited
./test_core
Erreur de segmentation (core dumped)
$ file core
core: ELF 32-bit LSB core file Intel 80386, version 1 (SYSV), SVR4-style, from
'./test_core'
$ gdb ./test core
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/tgayet/workspace/toto...(no debugging symbols
found)...done.
[New LWP 22423]

warning: Can't read pathname for load map: Erreur d'entrée/sortie.
Core was generated by `./test_core'.
Program terminated with signal 11, Segmentation fault.
#0  0x080483c4 in main ()
(gdb) bt
#0  0x080483c4 in main ()
```

Cependant, il est parfois utile d'obtenir une trace analogue à partir d'un programme via programmation, par exemple à des fins de trace ou de diagnostic.

2 Détail des fonctions de l'API execinfo

Le header **execinfo.h** déclare trois fonctions qui permettent de manipuler la backtraces du thread courant.

- **int backtrace(void **buffer, int size)** :

  La primitive **backtrace** permet d'obtenir la liste d'appel courante pour le thread actuel, comme une liste de pointeurs et place ces informations dans un buffer.

  La taille de l'argument doit être le nombre d'éléments **void*** qui sont dans le buffer. La valeur de retour est le nombre réel d'entrées de tampon qui sont retournés.

  Les pointeurs placés dans le buffer contiennent des adresses de retour obtenu via inspection de la pile, une adresse de retour par élément.

  Il est à noter que certaines optimisations du compilateur **gcc** peuvent interférer avec l'obtention d'une backtrace valide. En effet, les fonctions inline ne génèrent pas de traces et restent invisibles. L'optimisation remplace un élément de la liste par un autre.

- **char ** backtrace_symbols (void *const *buffer, int size)** :

  La primitive **backtrace_symbols** traduit l'information obtenue à partir de la primitive backtrace dans un tableau de chaînes de caractères. Le buffer en argument doit être un pointeur vers un tableau d'adresses obtenues via la fonction de backtrace et la taille est le nombre d'entrées dans ce tableau (la valeur de retour de backtrace).

  La valeur de retour est un pointeur sur un tableau de chaînes de caractères. Chaque chaîne contient une représentation imprimable de l'élément correspondant

au buffer. Il comprend le nom de la fonction (si elle peut être déterminée), l'offset dans la fonction et l'adresse de retour réelle (en hexadécimal).

Actuellement, le nom de la fonction et son offset peuvent être uniquement obtenus sur des systèmes qui utilisent le format binaire ELF pour les programmes et les bibliothèques.

Sur d'autres systèmes, seule l'adresse de retour en hexadécimal sera présente.

En outre, il est parfois nécessaire de passer des arguments supplémentaires à l'éditeur de liens pour que les noms de fonctions soient disponibles pour le programme. Par exemple, sur les systèmes utilisant GNU **ld**, vous devez passer (**-rdynamic**).

La valeur de retour de **backtrace_symbols** est un pointeur obtenu via la fonction **malloc** et il est de la responsabilité de l'appelant de libérer ce pointeur.

La valeur de retour est **NULL** s'il n'y a pas suffisamment pouvant être obtenu.

- **void backtrace_symbols_fd (void *const *buffer, int size, int fd)** :

La primitive **backtrace_symbols_fd** effectue la même traduction que la fonction **backtrace_symbols**. Au lieu de retourner directement des chaînes de caractères à l'appelant, il les écrit dans le descripteur de fichier **fd**, une par ligne.

Il n'utilise pas la fonction **malloc** et peut donc être utilisé dans des situations où cette fonction peut échouer.

Notez que le tableau pour contenir des adresses de retour renvoyés par la primitive **backtrace** allouée sur la pile.

C'est pourquoi ce type de code peut être utilisé dans des situations où la gestion de la mémoire via **malloc** ne fonctionne plus (dans ce cas, **backtrace_symbols** doit être remplacé par un appel **backtrace_symbols_fd** également).

Le nombre d'adresses de retour n'est généralement pas très grande. Même les programmes compliqués, plutôt rares, ont un niveau d'imbrication de plus que, disons, 50 et avec 200 entrées possibles probablement tous les programmes devraient être couverts.

3 Exemple simple d'affichage d'une trace d'appel

Exemple de code générant une trace d'appel sur stdout. Cela pourrait être afficher dans les traces si cela est sollicité explicitement ou bien généré juste avant un crash :

```
#include <execinfo.h>
#include <stdio.h>
#include <stdlib.h>

/* Génération de la trace d'appels sur stdout */
void print_trace(void)
{
  void* array[10];
    size_t size;
    char** strings;
    size_t i;

    size = backtrace(array, 10);
    strings = backtrace_symbols(array, size);

    printf ("Obtention de %zd fonctions dans la pile d'appel : \n", size);

    for (i = 0; i < size; i++)
    {
    printf ("%s \n", strings[i]);
    }

    free (strings);
}

/* Fonction factice pour rentre la backtrace plus fournie */
void dummy_function(void)
{
    print_trace();
}

int  main(void)
{
    dummy_function();
    return 0;
}
```

Exemple :

```
$ gcc bt01.c -o bt01
$ ./bt01
Obtention de 5 fonctions dans la pile d'appel
./bt01() [0x80484ad]
./bt01() [0x8048522]
./bt01() [0x804852f]
/lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf3) [0xb759f4d3]
./bt01() [0x8048401]
```

Cela fonctionne bien sauf... qu'il manque des symboles. Cela peut être solutionné facilement en rajoutant le flag **-export-dynamic** qui sera transmis à l'éditeur de lien ELF (**ld**). Ce flag demande d'ajouter tous les symboles et non pas juste ceux utilisés dans la table dynamique des symboles :

```
$ gcc bt01.c -export-dynamic -o bt01
$ ./bt01
Obtention de 5 fonctions dans la pile d'appel :
./bt01(print_trace+0x19) [0x804867d]
./bt01(dummy_function+0xb) [0x80486f2]
./bt01(main+0xb) [0x80486ff]
/lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf3) [0xb76154d3]
./bt01() [0x80485d1]
```

Et voilà, c'est déjà beaucoup mieux.

4 Dump en cas de crash

Faisons évoluer le code pour qu'au moment d'un crash d'un binaire via un segfault par exemple, une trace d'appel soit afficher sur stdout. Cela peut en effet être utile lorsqu'une exception est déclenchée.

Le code suivant montre comment associer un handle de fonction à certains signaux de façon a afficher la callstack courante avant de quitter le processus :

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>
#include <execinfo.h>
#include <sys/times.h>
#include <sys/ucontext.h>
#include <asm/unistd.h>

static int _pid = 0;

/* fonction d'affichage de la pile d'appels courante */
static void _show_callstack()
{
   void*  trace[16];
   char** messages      = (char **)NULL;
   int    i, trace_size = 0;

   trace_size = backtrace(trace, 16);
   messages   = backtrace_symbols(trace, trace_size);

   printf("CALLSTACK:\n");
   for (i=1; i<trace_size; ++i)
      printf("%s\n", messages[i]);

   free (messages);
}

/* Fonction de gestion des signaux */
static void _sigdemux(int sig, struct siginfo* si, void *v)
{
   int si_code = si->si_code & 0xffff;

   if (sig != SIGINT)
   {
      fprintf(stderr, "\n=============================================\n");
      fprintf(stderr, "Signal %s (#%d) reçu par le processus ayant le PID
%ld\n",
    strsignal(sig),
    (int)sig,
    syscall(__NR_gettid));
      fprintf(stderr, "Errno\t%i\nPID\t%i\naddr\t0x%08x\n\n",
    (int)si->si_errno,
    (int)si->si_pid,
    (int)si->si_addr);
      _show_callstack();
      fprintf(stderr, "=============================================\n");

      kill(_pid, SIGINT);
      exit(-1);
    }

    if (getpid() == _pid)
    {
          exit(0);
    }

    return;
}

/* Fonction déclarant un handle sur certain signaux */
int sig_init(int pid)
{
   struct sigaction si_segv;
   int              rc;

   si_segv.sa_sigaction = _sigdemux;
   si_segv.sa_flags     = SA_SIGINFO;
   _pid                 = pid;

   rc = sigaction(SIGINT, &si_segv, NULL);
   if (rc<0) {
      goto sig_error;
   }
   rc = sigaction(SIGSEGV, &si_segv, NULL);
   if (rc<0) {
      goto sig_error;
   }
   rc = sigaction(SIGILL, &si_segv, NULL);
   if (rc<0) {
      goto sig_error;
   }
   rc = sigaction(SIGFPE, &si_segv, NULL);
   if (rc<0) {
      goto sig_error;
   }
   rc = sigaction(SIGBUS, &si_segv, NULL);
   if (rc<0) {
      goto sig_error;
   }
   rc = sigaction(SIGPIPE, &si_segv, NULL);
   if (rc<0) {
      goto sig_error;
   }

sig_error:
  return rc;
}

/* Point d'entrée du binaire */
int main(int argc, char* argv[], char* env[])
{
  /* Initialisation du catcher de signal */
  sig_init(getpid());

  /* Pause avant le crash 'programmé' */
  sleep(5);

  /* Code générant l'envoie d'un signal par le noyau */
  /* de façon à simuler un crash                     */
  char *s = "crashme";
  *s = 'H';

  return 0;
}
```

Génération du binaire :

```
$ rm -f bt02 ; gcc bt02.c -export-dynamic -o bt02
```

Exemple de résultat :

```
$ ./bt02

================================================
Signal Segmentation fault (#11) reçu par le processus ayant le PID 22086
Errno0
PID 134515929
addr 0x08048cd9

CALLSTACK:
./bt02() [0x8048966]
[0xb76f040c]
./bt02(main+0x2e) [0x8048b25]
/lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf3) [0xb753e4d3]
./bt02() [0x8048791]
```

5 Autre exemple de backtrace

Voila un autre exemple de backtrace plus fournis :

```
========================
SIG SIGSEGV (#11) received in PID 2794
Errno   0
PID     0
addr    0x00000000

Got signal faulty address is (nil), from 0xb750bca4
[bt] Execution path:
[bt] /usr/lib/libsystem_api_adapter_configuration_repository.so.1(_ZN26DbusLocalStorageReposit
ory7getItemERKSs+0x3a) [0xb750bca4]
[bt] [0xffffe40c]
[bt] /usr/lib/libsystem_api_adapter_configuration_repository.so.1(_ZN26DbusLocalStorageReposit
ory7getItemERKSs+0x3a) [0xb750bca4]
[bt] /usr/lib/libsystem_api_adapter_configuration_repository.so.1(_
ZN3com11Technicolor23ConfigurationRepository30LocalStorageRepository_adaptor13_getItem_stubERK
N4DBus11CallMessageE+0x6d) [0xb750cac9]
[bt] /usr/lib/libsystem_api_adapter_configuration_repository.so.1(_
ZNK4DBus8CallbackIN3com11Technicolor23ConfigurationRepository30LocalStorageRepository_adaptorE
NS_7MessageERKNS_11CallMessageEE4callES8_+0x60) [0xb750f23c]
[bt] /usr/lib/libdbus-c++-1.so.0(_ZN4DBus16InterfaceAdaptor15dispatch_methodERKNS_11CallMessag
eE+0x135) [0xb6f669f5]
[bt] /usr/lib/libdbus-c++-1.so.0(_ZN4DBus13ObjectAdaptor14handle_messageERKNS_7MessageE+0xd0)
[0xb6f6d0a0]
[bt] /usr/lib/libdbus-c++-1.so.0(_ZN4DBus13ObjectAdaptor7Private21message_function_stubEP14DBu
sConnectionP11DBusMessagePv+0xb9) [0xb6f6b509]
[bt] /usr/lib/libdbus-1.so.3(+0x1822f) [0xb6f3122f]
[bt] /usr/lib/libdbus-1.so.3(dbus_connection_dispatch+0x365) [0xb6f244cd]
[bt] /usr/lib/libdbus-c++-1.so.0(+0x1c280) [0xb6f73280]
[bt] /usr/lib/libdbus-c++-1.so.0(_ZN4DBus10Dispatcher16dispatch_pendingEv+0x4e) [0xb6f7b94e]
[bt] /usr/lib/libdbus-c++-1.so.0(_ZN4DBus13BusDispatcher12do_iterationEv+0x23) [0xb6f82653]
[bt] /usr/lib/libdbus-c++-1.so.0(_ZN4DBus13BusDispatcher5enterEv+0x40) [0xb6f82370]
[bt] /usr/lib/libsystem_api_adapter_dbus.so.1(+0x6e0f) [0xb703ce0f]
[bt] [0x12]
[bt] /lib/libc.so.6(clone+0x5e) [0xb6cec8de]
```

On remarque que cette callstack permet d'identifier la fonction étant à l'origine du crash.

Conclusion

Voila donc un moyen permettant de récupérer une pile d'appels de fonction à un instant t, que ce soit au moment d'une erreur ou non. Une possibilité serait d'inclure et d'activer ce code uniquement en phase de développement. En effet, tout comme avec les assertions, vaut mieux ne pas les activer en production. ■

Liens

- http://www.delorie.com/gnu/docs/glibc/libc_665.html
- http://linux.die.net/man/3/dladdr
- http://linux.die.net/man/1/gcc
- http://linux.die.net/man/8/ld.so
- http://linux.die.net/man/8/ldconfig
- http://linux.die.net/include/dlfcn.h
- http://fossies.org/dox/cairo-1.12.14/backtrace-symbols_8c_source.html
- http://retropaganda.info/~bohan/work/psycle/branches/bohan/tests/opengl/gles/o3d/googleclient/third_party/cairo/util/cairo-trace/lookup-symbol.c
- https://github.com/Distrotech/cairo/blob/master/util/backtrace-symbols.c
- https://github.com/dhylands/projects/blob/master/host/lib-backtrace/backtrace.c
- http://www.sourceware.org/ml/libc-hacker/2009-08/msg00000.html
- http://stackoverflow.com/questions/11731229/dladdr-doesnt-return-the-function-name
- http://stackoverflow.com/questions/6948438/stacktrace-and-functions-in-namespaces
- http://scaryreasoner.wordpress.com/2007/11/17/using-ld_preload-libraries-and-glibc-backtrace-function-for-debugging/
- http://man7.org/linux/man-pages/man7/signal.7.html
- http://lists.gnu.org/archive/html/autoconf/2011-03/msg00024.html
- http://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html
- http://en.wikipedia.org/wiki/Binary_File_Descriptor_library
- http://www.skyfree.org/linux/references/bfd.pdf
- http://sourceware.org/binutils/docs/bfd/
- http://gcc.gnu.org/onlinedocs/gcc/Link-Options.html

ANATOMIE D'UNE DES PARTICULARITÉS DE LA LIBC

par Thierry GAYET

L'une des particularités de la librairie glibc, en plus d'être une librairie tampon au noyau GNU/linux, est aussi exécutable comme un processus standard.

1 Analyse des faits

Peu de personnes connaissent l'une des particularités de la bibliothèque GNU libc qui consiste à être exécutable. En effet, il est possible de l'exécuter tel que nous le ferions avec un binaire standard. Anciennement localisé dans le répertoire **/lib**, elle est désormais placée dans un sous-répertoire **/lib/i386-linux-gnu** comme l'indique l'inspection avec **ldd** d'un binaire :

```
$ ldd /bin/grep
  libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2 (0xb76a8000)
  libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb74ff000)
  /lib/ld-linux.so.2 (0xb76e2000)
$ ls -alH /lib/i386-linux-gnu/libc.so.6
-rwxr-xr-x 1 root root 1730024 janv. 28 13:30 libc.so.6
$ file /lib/i386-linux-gnu/libc-2.15.so
/lib/i386-linux-gnu/libc-2.15.so: ELF 32-bit LSB shared object, Intel 80386,
version 1 (SYSV), dynamically linked (uses shared libs), BuildID[sha1]=0x006e
0da0b3591a57f8cc9013d07dd253b8e81f2d, for GNU/Linux 2.6.24, stripped
```

On peu aussi observer un point d'entrée localisé à l'adresse **0x19630** mais aussi la section **interp** dont nous parlerons plus tard :

```
$ readelf -l libc.so.6

Elf file type is DYN (Shared object file)
Entry point 0x19630
There are 10 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00140 0x00140 R E 0x4
  INTERP         0x16b368 0x0016b368 0x0016b368 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x00000000 0x00000000 0x1a2dd0 0x1a2dd0 R E 0x1000
  LOAD           0x1a31c0 0x001a31c0 0x001a31c0 0x02d1c 0x0591c RW  0x1000
  DYNAMIC        0x1a4d7c 0x001a4d7c 0x001a4d7c 0x000f0 0x000f0 RW  0x4
  NOTE           0x000174 0x00000174 0x00000174 0x00044 0x00044 R   0x4
  TLS            0x1a31c0 0x001a31c0 0x001a31c0 0x00008 0x00040 R   0x4
  GNU_EH_FRAME   0x16b37c 0x0016b37c 0x0016b37c 0x076ec 0x076ec R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x1a31c0 0x001a31c0 0x001a31c0 0x01e40 0x01e40 R   0x1

 Section to Segment mapping:
 Segment Sections...
  00
  01     .interp
  02     .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.
version .gnu.version_d .gnu.version_r .rel.dyn .rel.plt .plt .text __libc_
freeres_fn __libc_thread_freeres_fn .rodata .interp .eh_frame_hdr .eh_frame
.gcc_except_table .hash
  03     .tdata .init_array __libc_subfreeres __libc_atexit __libc_thread_
subfreeres .data.rel.ro .dynamic .got .got.plt .data .bss
  04     .dynamic
  05     .note.gnu.build-id .note.ABI-tag
  06     .tdata .tbss
  07     .eh_frame_hdr
  08
  09     .tdata .init_array __libc_subfreeres __libc_atexit __libc_thread_
subfreeres .data.rel.ro .dynamic .got
```

Il est donc possible de l'exécuter, ce qui provoque l'affichage de sa version ainsi que d'un message statique :

```
$ ./libc.so.6
GNU C Library (Ubuntu EGLIBC 2.15-0ubuntu10.4) stable release version 2.15,
by Roland McGrath et al.
Copyright (C) 2012 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 4.6.3.
Compiled on a Linux 3.2.35 system on 2013-01-28.
Available extensions:
  crypt add-on version 2.1 by Michael Glad and others
  GNU Libidn by Simon Josefsson
  Native POSIX Threads Library by Ulrich Drepper et al
  BIND-8.2.3-T5B
libc ABIs: UNIQUE IFUNC
For bug reporting instructions, please see:
<http://www.debian.org/Bugs/>.
```

Avec une librairie « standard » cela n'est pas possible et donne l'erreur suivante :

```
$ ./libpam.so.0
bash: ./libpam.so.0: Permission non accordée
```

Enfin presque puisqu'il s'agit d'un lien symbolique :

```
$ ls -al libpam.so.0
lrwxrwxrwx 1 root root 16 févr.  9  2012 libpam.so.0 -> libpam.so.0.83.0

$ ls -al libpam.so.0.83.0
-rw-r--r-- 1 root root 50760 févr.  9  2012 libpam.so.0.83.0
```

Il est à noter que les bibliothèques ne possèdent pas le flag permettant leur exécution. On peut remédier à cela facilement :

```
$ sudo chmod +x libpam.so.0.83.0
$ ls -al libpam.so.0.83.0
-rwxr-xr-x 1 root root 50760 févr.  9  2012 libpam.so.0.83.0
```

Il est désormais possible de lancer une exécution :

```
./libpam.so.0.83.0
Erreur de segmentation (core dumped)
```

SIGSEGV est un signal envoyé à un processus lorsque celui-ci fait référence à une zone de mémoire invalide, par exemple parce qu'elle ne lui appartient pas. Une interruption est alors déclenchée et interrompt le programme. **SIG** désigne le mot signal, **SEG** segmentation et **V** violation. Sous Linux, il correspond au signal numéro 11 (vérifiable par la commande **kill -l**).

2 Explications

Alors comment cela se fait-il ?

Étudions dans un premier temps le code de la bibliothèque eglibc. Pour disposer des sources, il faut installer sous Debian ou Ubuntu le package **eglibc-source**.

```
sudo apt-get install  eglibc-source
```

Cela installe les sources de la eglibc sous la forme d'une archive :

```
ls -al /usr/src/glibc/eglibc-2.15.tar.xz
-rw-r--r-- 1 root root 11680952 janv. 28 13:24 /usr/src/glibc/eglibc-2.15.tar.xz
```

Parmi les sources, deux fichiers sont intéressants :

- **Makerules** à la racine des sources :

```
(...)
# Don't try to use -lc when making libc.so itself.
# Also omits crti.o and crtn.o, which we do not want
# since we define our own `.init' section specially.
LDFLAGS-c.so = -nostdlib -nostartfiles
# But we still want to link libc.so against $(libc.so-gnulib).
LDLIBS-c.so += $(libc.so-gnulib)
# Give libc.so an entry point and make it directly runnable itself.
LDFLAGS-c.so += -e __libc_main
# If lazy relocation is disabled add the -z now flag.
ifeq ($(bind-now),yes)
LDFLAGS-c.so += -Wl,-z,now
endif
(...)
```

Ce fichier détaille les règles qui seront utilisés pour rajouter le point d'entrée à la librairie dans les flags de l'éditeur de liens.

- **version.c** dans le répertoire **csu/** :

```
(...)
#ifdef HAVE_ELF
/* This function is the entry point for the shared object.
   Running the library as a program will get here.  */

extern void __libc_main (void) __attribute__ ((noreturn));
void
__libc_main (void)
{
  __libc_print_version ();
  _exit (0);
}
#endif
(...)
```

Ce fichier ajoute le point d'entrée qui est appelé lorsque la glibc est exécutée comme un processus. En effet, l'éditeur de liens de la suite GNU/GCC permet le remplacement de la fonction d'entrée d'un exécutable. Cela se fait via le paramètre **-e** ou **--entry**. Le man de **ld** en donne l'explication suivante :

```
-e entry
--entry=entry
Use  entry  as  the explicit symbol for beginning
execution of your program, rather than the default
entry point. If there is no  symbol  named  entry,
the linker will try to parse entry as a number and
use that as the entry address (the number will  be
interpreted in  base  10;  you may use a leading 0x
for base 16, or a leading 0 for base 8).
```

3 Ecriture d'un binaire avec point d'entrée personnalisé

Je propose de commencer par la modification du point d'entrée personnalisé d'un binaire :

Code source du fichier **mainexec.c** :

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>

extern void print_message(void);

void print_message(void)
{
  printf("hello world !!");
}

extern void my_main(void) __attribute__ ((noreturn));

void my_main(void)
{
  print_message();
  exit(0);
}
```

Compilation :

```
$ gcc -Wall -fPIC -c mainexec.c -o mainexec.o
```

Édition de liens :

```
$ gcc -nostartfiles -fPIC -e my_main mainexec.o  -o mainexec
```

Vérification :

```
$ ls -al mainexec
-rwxrwxr-x 1 tgayet tgayet 5722 juil. 21 20:02 mainexec
$ file mainexec
mainexec: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), BuildID[sha1]=0xf6c8cbf026a5ef
1f8df2b5875a2e1cafbe98cf5a, not stripped
$ ldd mainexec
```

```
  libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7517000)
  /lib/ld-linux.so.2 (0xb76f5000)
$ nm mainexec
08049f54 d _DYNAMIC
08049ff4 d _GLOBAL_OFFSET_TABLE_
0804a008 A __bss_start
080482a9 T __i686.get_pc_thunk.bx
0804a008 A _edata
0804a008 A _end
         U exit@@GLIBC_2.0
08048286 T my_main
08048260 T print_message
         U printf@@GLIBC_2.0
```

Execution du binaire :

```
$ ./mainexec
hello world!!
```

4 Ecriture d'une librairie exécutable comparable à la libc :

Maintenant que l'on a plus de détails, passons à l'étape suivante à savoir une bibliothèque dynamique exécutable. Créer une librairie exécutable n'est pas si trivial qu'il n'y paraît. En effet, une bibliothèque est compilé avec un **LDFLAG** positionné à **-shared -o libfoo.so -Wl,-soname,lifoo.so**.

Commençons avec le code source **service.c** suivant :

```
#include <stdio.h>

void lib_service(void)
{
  printf("Fonction fake de la librairie\n");
}

void lib_entry(void)
{
  printf("Point d'entrée de la librairie\n");
}
```

Pour compiler :

```
$ gcc -shared service.c -o libservice.so -Wl,-soname,libservice.so -Wl,-e,lib_entry
```

L'exécution échouera :

```
$ ./libservice.so
Illegal instruction
```

En réalité, il est nécessaire que le point d'entrée soit une fonction qui ne retourne rien et cela est possible avec l'appel système **_exit()**. Le code source évolue de la façon suivante :

```
#include <stdio.h>
#include <unistd.h>

void lib_service(void)
{
  printf("Fonction fake de la librairie\n");
}

void lib_entry(void)
{
  printf("Point d'entrée de la librairie\n");
  _exit(0);
}
```

```
$ gcc -shared service.c -o libservice.so -Wl,-soname,libservice.so -Wl,-e,lib_entry
$ ./libservice.so
Illegal instruction
```

Le tableau était presque complet sans compter que le linker a besoin que l'on lui spécifie l'interpréteur de fichier qui, sous Linux, est **/lib/ld-linux.so.2**.

5 Solution du problème

En tenant compte de tous les éléments précédents, le code source de la librairie est le suivant :

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>

const char service_interp[] __attribute__((section(".interp"))) = "/
lib/ld-linux.so.2";

/* --- Main entry point --- */

extern void print_message(void);

void print_message(void)
{
  printf("I am an executable library !!");
}

extern void my_main(void) __attribute__ ((noreturn));

void my_main(void)
{
  print_message();
  _exit(0);
}

/* --- At least one fake function for the dynamic library --- */

const char* getState(void)
{
  return "1.0";
}
```

Makefile :

```
MAJOR=1
MINOR=0
VERSION=${MAJOR}.${MINOR}
NAME=exec
LIBRARY=lib${NAME}.so.${VERSION}

all: ${LIBRARY}

clean:
  echo "[RM] Cleaning. "
  rm -f *.o
  rm -f *~
  rm -f ${LIBRARY}

${LIBRARY}: libexec.o
  echo "[LD] $^ -> $@ "
  gcc -shared $^ -o $@ -Wl,-soname,$@ -Wl,-e,my_main

libexec.o:libexec.c
  echo "[CC] $^ -> $@ "
  gcc -Wall -fPIC -c $^ -o $@

test: ${LIBRARY}
```

```
  echo "[CHK] Library"
  ls -al      ./${LIBRARY}
  file        ./${LIBRARY}
  readelf -l ./${LIBRARY}
  nm          ./${LIBRARY}
  ldd         ./${LIBRARY}

run: ${LIBRARY}
  echo "[RUN] Library."
  ./${LIBRARY}
```

Démonstration :

```
$ make clean && make &&  make test && make run
[RM] Cleaning.
[CC] libexec.c -> libexec.o
[LD] libexec.o -> libexec.so.1.0
[CHK] Library
-rwxrwxr-x 1 tgayet tgayet 7201 juil. 23 21:40 ./libexec.so.1.0
./libexec.so.1.0: ELF 32-bit LSB shared object, Intel 80386, version 1
(SYSV), dynamically linked (uses shared libs), BuildID[sha1]=0xbe6c7bc2c2c
f1500bd40af04670d5ca898ec52eb, not stripped

Elf file type is DYN (Shared object file)
Entry point 0x60e
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00120 0x00120 R E 0x4
  INTERP         0x0006e0 0x000006e0 0x000006e0 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x00000000 0x00000000 0x00834 0x00834 R E 0x1000
  LOAD           0x000ee4 0x00001ee4 0x00001ee4 0x00138 0x00140 RW  0x1000
  DYNAMIC        0x000f00 0x00001f00 0x00001f00 0x000e8 0x000e8 RW  0x4
  NOTE           0x000154 0x00000154 0x00000154 0x00024 0x00024 R   0x4
  GNU_EH_FRAME   0x0006f4 0x000006f4 0x000006f4 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x000ee4 0x00001ee4 0x00001ee4 0x0011c 0x0011c R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.
version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .interp .eh_
frame_hdr .eh_frame
   03     .init_array .fini_array .ctors .dtors .jcr .dynamic .got .got.
plt .data .bss
   04     .dynamic
   05     .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .ctors .dtors .jcr .dynamic .got
00001f00 a _DYNAMIC
00001ff4 a _GLOBAL_OFFSET_TABLE_
         w _Jv_RegisterClasses
00001ef0 d __CTOR_END__
00001eec d __CTOR_LIST__
00001ef8 d __DTOR_END__
00001ef4 d __DTOR_LIST__
00000830 r __FRAME_END__
00001efc d __JCR_END__
00001efc d __JCR_LIST__
0000201c A __bss_start
         w __cxa_finalize@@GLIBC_2.1.3
00000650 t __do_global_ctors_aux
000004e0 t __do_global_dtors_aux
00002018 d __dso_handle
         w __gmon_start__
00000597 t __i686.get_pc_thunk.bx
00000647 t __i686.get_pc_thunk.cx
0000201c A _edata
00002024 A _end
00000688 T _fini
00000438 T _init
```

```
000005c2 t bar
0000201c b completed.6159
00002020 b dtor_idx.6161
         U exit@@GLIBC_2.0
0000059c t foo
00000560 t frame_dummy
00000631 T getState
0000060e T my_main
000005e8 T print_message
         U printf@@GLIBC_2.0
         U puts@@GLIBC_2.0
000006e0 R service_interp
  libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0x4005a000)
  /lib/ld-linux.so.2 (0x40000000)
[RUN] Library.
I am executable09:40:07 ~/workspace/libexec$ ls
libexec.c  libexec.o  libexec.so.1.0  mainexec  mainexec.c  Makefile
09:42:38 ~/workspace/libexec$ ./libexec.so.1.0
I am an executable library
```

Et voilà le tour est joué !

6 Exemple d'utilisation

Imaginons une bibliothèque dynamique disposant d'une API de fonctions. Ne disposant que d'un fichier avec une extension **.so**, on se pose souvent la question de la version qui sous GNU/Linux est souvent indiqué par un suffixe arbitraire dans le nommage du fichier. Mis à part cette convention, il peut être compliqué de la trouver rapidement.

Imaginons que le fait de disposer d'une entrée exécutable dans une bibliothèque dynamique puisse permettre de lui passer des options comme **-v** pour obtenir sa version ou bien **-t** pour lancer les tests unitaires de non-régression en cas de compilation avec un flag **TEST** donné.

Voila un exemple d'utilisation pragmatique.

Conclusion

Bon, je ne sais pas si cela vous sera utile, mais peut être que l'on pourra trouver des cas d'utilisation qui nous rendrons service au delà de ceux que je viens d'exposer.

Nous sommes cependant rentré dans le détail des bibliothèques dynamiques, de l'éditeur de liens, des exécutables, du format ELF et de la librairie C, ce qui est passionnant non ;-) !!

A bon entendeur, salut ! ■

Liens

- http://linux.about.com/library/cmd/blcmdl2_exit1.htm
- http://linux.about.com/library/cmd/blcmdl1_ld.htm
- http://linux.about.com/library/cmd/blcmdl1_readelf.htm
- http://linux.about.com/library/cmd/blcmdl1_objdump.htm
- http://linux.about.com/library/cmd/blcmdl1_ldd.htm
- http://linux.about.com/library/cmd/blcmdl1_nm.htm

PHPEXCEL :

LA SOLUTION ULTIME POUR ÉCHANGER DES DONNÉES ENTRE PHP ET VOTRE TABLEUR

par Stéphane Mourey

Il arrive fréquemment dans la vie d'un développeur qu'on lui demande d'importer ou d'exporter des données depuis ou vers un format de fichier ayant une orientation tableur. Le plus souvent, il opte pour une solution à la base de fichier CSV sortant du standard pour pouvoir être ouvert sans difficulté par MS Excel. Une autre solution existe pourtant et j'avoue avoir été impressionné par sa puissance et sa facilité d'utilisation : PHPExcel.

1 Un projet PHPOffice

PHPExcel est un projet membre de la famille PHPOffice (https://github.com/PHPOffice) qui comporte également PHPWord, PHPPowerPoint, PHPProject et PHPVisio. De tous ces projets, il est sans aucun doute le plus abouti et le plus dynamique. PHPWord m'a paru bien abouti également, bien que je n'ai pas encore eu l'occasion de le tester aussi profondément que PHPExcel, peut-être en reparlerons-nous dans ces colonnes.

Vous aurez remarqué que les noms de ces librairies rappellent étrangement ceux de produits Microsoft simplement préfixés de « PHP ». Rien d'étonnant à cela, puisque ces projets ont pris naissance sur CodePlex (http://www.codeplex.com), le site d'hébergement de projets OpenSource de Microsoft, une sorte de SourceForge à sauce Redmon. Sans doute que les développeurs de PHPOffice n'en étaient pas complètement satisfaits, puisque le 19 juin 2012, le projet a migré vers GitHub.

Pour autant, il ne faut pas croire que les capacités de communication de PHPExcel sont limitées à MS Excel, sinon, je crois bien que je n'aurai pas écrit cet article. Au contraire, cette librairie a des capacités impressionnantes en terme de formats de fichier supportés à l'import et à l'export, ce qui en fait, à mes yeux, un outil d'excellence pour ce genre d'exercice. Toutefois, ne rêvez pas, cette agilité a un coût, notamment en terme de mémoire et si vous n'avez pas le contrôle sur la configuration de votre serveur, vous risquez de passer quelques moments pénibles à chercher des solutions pour minimiser l'impact de PHPExcel. Je vous donnerai quelques pistes ici, que j'ai expérimentées en production mais dans tous les cas, si vous êtes sur un serveur partagé, si vous voulez utiliser PHPExcel sur un site très fréquenté ou encore si vous voulez utiliser de manière intensive cette bibliothèque, vous ne pourrez sans doute pas régler toutes les difficultés au niveau du code.

2 Dans quels cas utiliser PHPExcel ?

PHPExcel se révélera idéal pour travailler avec des classeurs de tableaux complexes. Il a été étudié pour être apte à imiter toutes les fonctionnalités d'un bon tableur, tant en terme de calcul que de mises en forme. Il se révèle capable d'importer, de modifier et d'exporter des fichiers complexes qui n'auront pas à rougir à côté de ceux que vous aurez vous-même produits dans LibreOffice.

À l'usage, il n'en reste pas moins vrai qu'il est souvent plus agréable de cliquer sur un bouton pour mettre un texte en gras que de taper une ligne de code. Aussi, l'utilisation la plus pratique de PHPExcel, lorsqu'il s'agit de produire un document, est de construire un modèle

de classeur dans votre tableur préféré, de l'importer pour le modifier et l'exporter. Mais, dans ce cas, il vous faudra avoir au moins une bonne idée du fonctionnement global de PHPExcel.

3 Principe de fonctionnement

PHPExcel ne travaille pas en mode fichier. Il ne se préoccupe de fichiers qu'au moment d'importer ou d'exporter les données. Hors de ces deux actions, PHPExcel produit une représentation interne de votre classeur et ne travaille qu'avec elle. Il la construit au moment de la création ou de l'importation et la transforme en fichier au moment de l'exportation.

D'où les problèmes d'utilisation de mémoire qui peuvent survenir lorsque vous voulez utiliser des fichiers comportant des milliers de lignes réparties sur des dizaines de feuilles comportant des mises en forme complexes. Tout devra être chargé jusqu'à ce que vous ayez terminé. D'où les problèmes de mémoire qui peuvent survenir. Votre premier réflexe, si vous rencontrez de tels problèmes sera sans doute d'augmenter la mémoire disponible pour un script PHP, directive **memory_limit** du fichier **php.ini**. Sa valeur par défaut est de 128 mégaoctets, (voire 16 ou même 8 si vous avez une version ancienne de PHP...).

4 Installation

Une fois n'est pas coutume, je vais vous présenter deux manières d'installer cette librairie. La première méthode suivra le cheminement habituel du téléchargement, la seconde utilisera Composer, le gestionnaire de dépendances de PHP.

4.1 Pré-requis

Pour que PHPExcel fonctionne, il vous faudra avoir installé PHP en version 5.2 au moins avec l'extension **php_xml**. L'extension **php_zip** est requise pour manipuler des fichiers **.xlsx** et **.gnumeric**. L'extention **php_gd2** est recommandée pour une détermination exacte des largeurs automatiques de colonnes.

Si vous souhaitez faire des exports en PDF, vous aurez également besoin d'une librairie sachant les générer, au choix tcPDF (http://www.tcpdf.org), DomPDF (https://github.com/dompdf/dompdf) ou mPDF (http://www.mpdf1.com) au choix.

4.2 Par téléchargement

Rendez-vous sur la page GitHub du projet et cliquez sur le bouton « ZIP » pour lancer le téléchargement.

Ou, si vous préférez, vous pouvez utiliser Git pour cloner le projet localement :

```
$ git clone git://github.com/PHPOffice/PHPExcel.git
```

4.3 Avec Composer

Voici le fichier **composer.json** à placer à la racine de votre projet (ou le texte à y ajouter si vous utilisez déjà Composer) :

```
{
  "require": {
     "phpexcel/phpexcel": "v1.7.7"
  }
}
```

Il n'y a plus qu'à lancer l'installation :

```
$ composer install
```

si vous avez installé Composer globalement (voir notre article sur Composer dans le GNU/Linux Magazine N° 162 du mois d'août dernier).

5 Utilisation

Nous ne ferons pas ici une description exhaustive de toutes les fonctionnalités de PHPExcel, mais nous nous arrêterons sur les plus importantes et nous étudierons comment manipuler des classeurs, des valeurs et des cellules, ce qui suffira à résoudre la plupart des problèmes que vous pourriez rencontrer dans une utilisation ordinaire de PHPExcel. Pour ce qui est des mises en forme avancées, de la génération de graphiques, de tableaux croisés dynamiques, etc, vous pourrez vous référer à la documentation officielle, incluse dans la librairie.

5.1 Un peu de configuration

Avant de créer votre premier classeur, quelques éléments de configuration peuvent se révéler utiles : l'utilisation d'un cache pour stocker les cellules de votre classeur et la localisation.

5.1.1 Le cache

Par défaut, PHPExcel stocke toutes les cellules dans la mémoire. Étant donné qu'en moyenne, 1 Ko est nécessaire pour stocker une cellule, il se peut que des problèmes ne tardent pas à surgir, même avec des tableaux de dimensions relativement modestes. La première solution à mettre en œuvre est d'utiliser un cache. Pour activer cette fonctionnalité, il faut appeler la méthode **setCacheStorageMethod**. Celle-ci accepte un paramètre indiquant la méthode de cache à utiliser et renvoie **true** en cas de succès :

```
$cacheMethod = PHPExcel_CachedObjectStorageFactory::cache_to_phpTemp;
$cacheSettings = array( 'memoryCacheSize'  => '8MB');
PHPExel_Settings::setCacheStorageMethod($cacheMethod,$cacheSettings);
```

Parmi les méthodes proposées, les plus rapides consistent à stocker les objets en mémoire sous une forme sérialisée plutôt qu'instanciée, en les gzippant optionnellement, les plus lentes utilisent un cache disque tel que APC ou memcache ou même SQLite. La méthode illustrée ci-dessus est

celle qui a ma préférence : elle utilise le flux **php://temp**. Ce flux est très similaire à **php://memory**, qui permet d'accéder à un système de fichier virtuel pouvant se révéler très pratique, à ceci près que **php://memory** réside exclusivement en mémoire et que **php://temp** n'y réside qu'à concurrence d'une certaine taille, par défaut 1 Mo mais qui peut être modifier, ce que nous avons fait ici avec le paramètre **$cacheSettings**.

Le choix de cette méthode de cache me paraît un bon compromis entre rapidité et capacité de traitement : tant que votre classeur ne dépasse pas une certaine taille, vous disposez de toute la rapidité de la mémoire ; puis, si cela devient nécessaire, une partie des données est déportées sur le disque dur, ce qui vous permettra de continuer à travailler, au prix d'un certain ralentissement, certes, mais qui n'interviendra que si nécessaire.

5.1.2 Localisation

Excel présentant des fonctionnalités liées à la localisation, il est normal que PHPExcel fasse de même. Pour créer un classeur utilisant le canadien français, vous procéderez ainsi :

```
$locale = 'fr_ca';
$validLocale = PHPExcel_Settings::setLocale($locale);
if (!$validLocale)
  {
    echo "Locale $locale unavailable, switching to en_us<br />\n";
  }
```

Si les données pour le français du Canada ne sont pas disponibles, alors PHPExcel utilisera celles pour le français de France... Et si ces dernières ne sont pas non plus disponibles, il basculera sur l'anglais américain et la méthode **setLocale** renverra **false**.

5.2 Manipulations de classeurs

5.2.1 Créer un nouveau classeur

Un classeur PHPExcel est un objet PHP. Pour en créer un nouveau, il suffit d'utiliser l'instruction **new** :

```
$monClasseur = new PHPExcel() ;
```

5.2.2 Importer un classeur

Pour charger un classeur depuis un fichier existant, ce n'est pas beaucoup plus compliqué :

```
$monFichier = '/chemin/vers/mon/fichier.xls';
$monClasseur = PHPExcel_IOFactory::load($monFichier);
```

Les formats Excel 2007, 2003, 5, CSV, ODS, SYLK et HTML sont supportés en lecture.

5.2.3 Enregistrer un classeur

Pour enregistrer votre classeur, vous devrez passer par un objet de type **PHPExcel_Writer**. Il en existe plusieurs classes, selon le format que vous voulez utiliser. Attention, certains formats ne sont supportés qu'en lecture et vous ne trouverez donc pas de classe **PHPExcel_Writer** correspondante. C'est le cas, on le regrettera, pour le format des classeurs OpenDocument (**.ods**).

Il est possible d'appeler directement la classe désirée. Voici un exemple utilisant le format recommandé, celui d'Excel 2007 (SpreadsheetML) :

```
$monSauveur = new PHPExcel_Writer_Excel2007($monClasseur);
$monSauver->save();
```

Mais on préférera un code s'appuyant sur la fabrique dédiée, plus souple :

```
$monFormat = "Excel2007";
$nomDeFichier = "monExcel.xlsx";
$monSauveur = PHPExcel_IOFactory::createWriter($monClasseur, $monFormat);
$monSauveur->save($nomDeFichier);
```

Il sera ainsi plus facile de laisser le choix du format de sortie à l'utilisateur. Les autres formats supportés en écriture sont : Excel 5, Excel 2003 (à travers une option de compatibilité à appliquer sur le writer Excel 2007), CSV, HTML et PDF (si vous avez installé les bibliothèques nécessaires).

5.2.4 Effacer un classeur de la mémoire

Vues les difficultés d'occupation de mémoire que PHPExcel peut poser, vous aurez sans doute besoin de supprimer les classeurs chargés en mémoire lorsque vous en aurez terminé avec eux. Mais il faudra prendre une précaution auparavant et ne pas utiliser **unset()** sans réfléchir. En effet, un objet PHPExcel contient lui-même des objets. Ceux-ci font entre eux de nombreuses références circulaires, qui posent problème à PHP lorsqu'il essaie de les détruire, ce qui peut mener à des fuites de mémoire et vous n'aurez pas gagné grand chose. Il faut d'abord envoyer au classeur l'instruction de déconnecter ses feuilles :

```
$monClasseur->disconnectWorksheets();
unset($monClasseur);
```

Pourquoi les développeurs de PHPExcel n'ont pas utilisé la méthode magique **__destruct**, je l'ignore.

5.3 Manipulation de feuilles

La possibilité de manipuler des feuilles est sans doute le point le plus motivant pour l'utilisation de PHPExcel à la place d'un export CSV, même compatible avec Excel, avec la manipulation des formules que nous verrons plus loin.

5.3.1 Accéder à une feuille

Pour cela, plusieurs méthodes sont disponibles.

Tout d'abord, pour accéder à la feuille actuellement active, on utilisera **getActiveSheet()** :

```
$monClasseur->getActiveSheet();
```

Vous pouvez ensuite accéder aux autres feuilles par leurs index à partir de 0 :

```
$monClasseur->getSheet(1);
```

Ou encore par son nom :

```
$monClasseur->getSheetByName('Feuille d\'automne');
```

La feuille actuellement active, il y en a toujours une, est celle qui sera active lorsque vous ouvrirez votre classeur dans votre tableur. Il est possible de la modifier en indiquant le nom ou l'index de la nouvelle feuille active respectivement en appelant les méthodes **setActiveSheetIndexByName** ou **setActiveSheetIndex**.

5.3.2 Ajouter une feuille

La méthode à utiliser pour cela est **createSheet**. Cette méthode accepte un argument optionnel entier qui indique l'index à utiliser pour son emplacement. La feuille qui se trouverait à cet index et toutes celles qui se trouveraient sur des index plus élevés verront leurs positions augmentées de 1.

Si aucun index n'est précisé, la nouvelle feuille trouvera sa place après la dernière.

La nouvelle feuille aura pour nom **Worksheet<n>** où **<n>** aura la plus petite valeur possible pour garantir l'unicité du nom.

5.3.3 Copier des feuilles

Il est possible de copier des feuilles à l'intérieur d'un classeur ou encore d'un classeur à l'autre. La façon de procéder est à peine différente, cela tient à ce que, d'un classeur à l'autre, des éléments de style doivent également être copiés, ce qui n'est pas le cas à l'intérieur d'un même classeur. En tous les cas, il est de la responsabilité du développeur de garantir que la copie n'aura pas le même nom qu'une autre feuille du classeur, auquel cas une exception serait levée.

Pour copier une feuille dans un même classeur, on commence par cloner la feuille, changer son nom et ajouter la nouvelle feuille au classeur :

```
$maFeuilleCopie = clone $monClasseur->getSheetByName('Feuille d\'automne');
$maFeuilleCopie->setTitle('Feuille de printemps');
$monClasseur->addSheet($maFeuilleCopie);
```

Avec un peu de chance, l'étape de renommage ne devrait pas être nécessaire lors d'une copie d'un classeur à l'autre :

```
$maFeuilleCopie = clone $monClasseur1->getSheetByName('Feuille
d\'automne');
$monClasseur2->addExternalSheet($maFeuilleCopie);
```

Notez que, cette fois-ci, nous n'avons pas utilisé **addSheet()** mais **addExternalSheet()** : cette dernière prend également en charge la copie des styles requis d'un classeur à l'autre.

5.3.3 Suppression d'une feuille

Une seule méthode est disponible pour cela, **removeSheetByIndex()** :

```
$monClasseur->removeSheetByIndex();
```

D'ordinaire en PHP, on se préoccupe peu de la destruction des objets en mémoire, le langage faisant en principe tout le nettoyage nécessaire à la fin du script. Avec PHPExcel, vu son occupation de la mémoire et les problèmes qui peuvent en résulter, n'hésitez surtout pas à éliminer les éléments qui ne vous sont plus nécessaires au fur et à mesure.

5.4 Manipulation de données

5.4.1 Accéder aux cellules

Pour modifier la valeur d'une cellule en connaissant ses coordonnées, on utilise la méthode **setCellValue** de la feuille active. Cette méthode prend deux arguments : tout d'abord les coordonnées de la cellule dans le format habituel aux tableurs, puis la valeur à attribuer :

```
$monClasseur->getActiveSheet()->setCellValue('B8','Une valeur');
```

Une méthode alternative est d'utiliser **setCellValueByColumnAndRow()**, qui permet d'utiliser des coordonnées numériques, ce qui peut se révéler pratique. Remarquez que la première colonne a pour coordonnée 0 :

```
$monClasseur->getActiveSheet()->setCellValueByColumnAndRow(1,8,'Une valeur');
```

Pour lire sa valeur, il faut passer par l'intermédiaire de la méthode **getCell** pour récupérer la cellule, puis utiliser la méthode **getValue** :

```
$value = $monClasseur->getActiveSheet()->getCell('B8')->getValue();
```

Une autre méthode est à utiliser si la valeur à récupérer est le résultat d'une formule affectée à la cellule, mais je ne la recommande pas, nous en reparlerons plus loin dans le paragraphe dédié aux formules.

De même que pour affecter une valeur à une cellule, il est également possible de la lire en utilisant des coordonnées numériques :

```
$value = $monClasseur->getActiveSheet()->getCellByColumnAndRow(1,8)->getValue();
```

Si l'accès aux cellules par coordonnées numériques permet de simplifier les algorithmes permettant d'y accéder en

évitant la conversion en coordonnées alpha-numériques, il existe des méthodes plus simples pour y accéder en boucle.

Il est possible d'utiliser un itérateur. Ci-dessous, un exemple permettant d'afficher toutes les valeurs des cellules d'une feuille dans un tableau HTML :

```
<?php
$objReader = PHPExcel_IOFactory::createReader('Excel2007');
$objReader->setReadDataOnly(true);

$monClasseur = $objReader->load("test.xlsx");
$maFeuille = $monClasseur->getActiveSheet();

echo '<table>' . "\n";
foreach ($maFeuille->getRowIterator() as $row) {
  echo '<tr>' . "\n";

  $cellIterator = $row->getCellIterator();
  $cellIterator->setIterateOnlyExistingCells(false);
  foreach ($cellIterator as $cell) {
    echo '<td>' . $cell->getValue() . '</td>' . "\n";
  }

  echo '</tr>' . "\n";
}
echo '</table>' . "\n";
```

Ici, nous avons chargé un fichier Excel en lecture seule. **getRowIterator()** nous permet de boucler sur toutes les lignes du tableau, tandis que **getCellIterator** nous permet de boucler sur toutes les cellules de chaque ligne. **setIterateOnlyExistingCell()** avec **false** comme argument nous permet de prendre en compte les cellules sans contenu défini et, ainsi, d'afficher un tableau respectant les positions de toutes les cellules sans qu'il soit perturbé par une cellule vide qui aurait été ignorée sans cela.

Comme nous l'avons dit, une méthodologie différente est également possible, en utilisant les coordonnées numériques. La difficulté est que les dimensions de la feuille ne sont pas nécessairement connues. Heureusement, PHPExcel fournit des méthodes permettant de les connaître. Reprenons l'exemple précédent en utilisant cette méthodologie :

```
$plusHauteLigne = $maFeuille->getHigestRow();
$plusHauteColone = $maFeuille->getHigestColumn();

$plusHauteColoneIndex = PHPExcel_Cell::columnIndexFromString($plusHauteColone);

echo '<table>' . "\n";
for ($row = 1; $row <= $plusHauteLigne; ++$row) {
  echo '<tr>' . "\n";

  for ($col = 0; $col <= $plusHauteColoneIndex; ++$col) {
    echo '<td>' . $maFeuille->getCellByColumnAndRow($col, $row)->getValue() .
'</td>' . "\n";
  }

  echo '</tr>' . "\n";
}
```

getHigestRow() et **getHigestColumn()** nous renvoient les coordonnées respectivement de la ligne et de la colonne les plus élevées. Notez que **getHigestColumn()** utilisera une lettre, qu'il faudra donc convertir par **PHPExcel_Cell::columnIndexFromString()** pour obtenir une valeur numérique utilisable pour notre boucle.

5.4.2 Formules

L'affectation de formule à une cellule s'effectue de la même façon que l'affectation de valeur. De même que dans votre tableur préféré, il suffit que la chaîne de caractère commence par le signe égal :

```
$monClasseur->getActiveSheet()->setCellValue('B8','=SUM(B2:B4)');
```

Ensuite, la valeur calculée peut être récupérée par la méthode **getCalculatedValue** :

```
$value = $monClasseur->getActiveSheet()->getCell('B8')->getCalculatedValue();
```

J'ai écrit plus haut que je ne recommandais pas d'utiliser cette méthode. Pourquoi ? Tout d'abord, cette valeur calculée ne devrait pas être utile à ce niveau : lorsque vous ouvrirez votre classeur dans votre tableur, celui-ci prendra automatiquement en charge ce calcul et si d'autres cellules en dépendaient, il devrait être possible de le gérer par d'autres formules sans s'appuyer sur la fonctionnalité de calcul de PHPExcel ; si vous avez réellement besoin d'un tel résultat, il vaudrait mieux l'effectuer en PHP pur.

Une telle recommandation est rendue nécessaire par l'existence d'un problème et d'une difficulté inhérente à PHPExcel : les formules sont converties en code PHP et il se trouve que la précédence des opérateurs n'est pas tout à fait la même dans Excel et dans PHP. À en croire la documentation, seuls les opérateurs **+** et **&** sont affectés mais cela suffirait déjà pour éviter d'utiliser cette fonctionnalité...

Mais il y a encore un autre problème, lié également à une différence de comportement entre PHP et Excel : les formules mêlant texte et valeurs numériques ne seront pas traitées de la même façon. Pour Excel, une formule telle que **=3+"Hello"** affichera une erreur sous la formule du texte **#VALEUR!**. PHP, lui, suivant son principe habituel de conversion de type automatique, remplacera la chaîne **"Hello"** par la valeur numérique **0** et donnera **3** comme résultat de la formule.

Ces différences de comportement ne paraîtront peut-être pas considérables et seront corrigées par le tableur lors de l'ouverture du classeur. Mais, outre qu'il est facile de ne pas utiliser cette fonctionnalité, imaginez le temps qui sera peut-être nécessaire pour découvrir un bug lié à ces problèmes pour peu que vous ayez l'idée, juste une fois, d'attribuer une valeur à une cellule à partir de données calculées d'autres cellules sans maintenir le lien entre elles par le biais d'une formule ?

Il me semble que la meilleure méthodologie à adopter est de s'appuyer autant que faire ce peut sur les capacités de traitement du tableur et utiliser PHPExcel pour alimenter votre classeur en données brutes et éventuellement en formules, en tâchant le plus possible de produire un document qui se suffise à lui-même, qu'il soit possible de comprendre entièrement sans connaître le processus de sa génération.

Enfin, pour conclure positivement sur la question des formules, remarquons que PHPExcel est capable, tout comme un tableur ordinaire, de faire en sorte que les formules s'adaptent à leur déplacement de façon à conserver le lien avec les cellules d'origine. Par exemple, imaginons que vous effectuez dans une cellule la somme des cellules **E4** à **E9** par l'intermédiaire de la fonction **SUM**, votre formule est **=SUM(E4:E9)** ; imaginons ensuite que vous rajoutiez une ligne après la cinquième ligne en utilisant la méthode **insertNewRowBefore** ; la formule de votre somme est alors automatiquement corrigée **=SUM(E4:E10)**.

6 Optimiser l'utilisation de la mémoire

Nous avons déjà vu, au chapitre consacré à la configuration comment utiliser un cache couplant mémoire et disque pour limiter les problèmes liés à la mémoire. Il se peut que cette solution ne convienne pas ou ne suffise pas. Si vous avez déjà augmenté la mémoire disponible pour PHP autant que possible, voici d'autres pistes à explorer pour vous aider à les résoudre.

6.1 Readers et Writers inégaux

Nous l'avons vu, PHPExcel s'appuie sur des objets de type Reader ou Writer pour respectivement importer ou enregistrer un classeur. Selon le format de fichier que vous utilisez, les différents types d'objets ont une occupation de la mémoire toute différente. Vous trouverez à cette adresse http://phpexcel.codeplex.com/discussions/234150 quelques tests rapportant l'utilisation de la mémoire faite par ces différents objets en fonction du format de fichier utilisé (de même que, plus bas, des statistiques sur l'utilisation de la mémoire en fonction de la méthode de cache utilisée pour les cellules). Le grand gagnant est, oh surprise, CSV... à privilégier donc pour l'import de données si vous en avez la possibilité.

6.2 Ne chargez que les feuilles utiles

Si vous devez charger un classeur comprenant de nombreuses feuilles mais dont seulement certaines vous seront utiles, vous pouvez utiliser la méthode **setLoadSheetsOnly** pour lui indiquer les feuilles que vous souhaitez charger et laisser les autres :

```
$reader = PHPExcel_IOFactory::createReader('Excel5');
$reader->setLoadSheetsOnly('Feuille d\'automne');
$reader->load('monFichier.xls');
```

setLoadSheetsOnly() accepte soit une chaîne de caractères, soit un tableau de chaînes, contenant respectivement le nom de la ou des feuilles à charger.

6.3 Ignorez les styles

Dans bien des cas, vous n'aurez pas à modifier les styles utilisés par votre classeur, mais seulement les données qu'il contient. PHPExcel vous permet dès lors de ne pas les charger à l'aide de la méthode **setReadDataOnly**, ce qui peut représenter un gain de mémoire significatif :

```
$reader = PHPExcel_IOFactory::createReader('Excel5');
$reader->setReadDataOnly(true);
$reader->load('monFichier.xls');
```

6.4 Des solutions plus élaborées encore

Nous n'avons ici examiné que les solutions les plus simples à mettre en œuvre, il en existe encore d'autres, comme la création d'un filtre de lecture, qui ne chargera que les cellules remplissant des conditions que vous aurez définies ou encore de fragmenter la lecture de votre classeur en chargeant une seule partie à la fois. Vous trouverez tous les détails à cette adresse : http://phpexcel.codeplex.com/discussions/242712

Ces méthodes ne sont pas très complexes à mettre en œuvre mais si vous en arrivez là, c'est que vous aurez déjà fait largement connaissance avec PHPExcel.

Conclusion

Nous n'avons abordé dans cet article qu'une petite partie des possibilités offertes par PHPExcel, en nous concentrant d'abord sur la manipulation des données, en laissant de côté d'autres aspects, en particulier la mise en forme de ces données. Ce n'est pas que PHPExcel soit limité de ce côté-là, bien au contraire, mais il nous a semblé que, dans un premier temps, l'argument le plus pertinent pour l'utilisation de PHPExcel était l'utilisation d'un classeur pré-existant avec une belle mise en forme pour y inclure des données issues d'un serveur de production. Si vous vous lancez dans la réalisation d'une application s'appuyant sur cette librairie, vous aurez tout le loisir de découvrir comment mettre un texte en gras, définir une bordure de cellule, indiquer les sauts de pages, insérer une image, rendre une URL cliquable, indiquer un format de date ou de nombre... Bref, tout ce que vous êtes en droit d'attendre d'un bon tableur. ■

GOOGLE APP ENGINE ACCUEILLE LES APPLICATIONS PHP

par Stéphane Mourey [Taohacker]

Depuis la mi-mai, Google a annoncé lors du Google I/O que PHP était maintenant supporté par le Google App Engine, rejoignant ainsi Java, Python et Go. Il s'agissait de l'évolution la plus demandée par les utilisateurs depuis la création de App Engine.

1 Google App Engine

Google App Engine est la plate-forme d'hébergement d'applications web de Google. Créée en 2008, elle ne supportait d'abord que les applications écrites en Python, avant de s'élargir à Java, Go et maintenant PHP. Google y propose gratuitement, par application, un espace de stockage de 1Go et les capacités nécessaires tant en termes de bande passante que de capacité de calcul pour servir cinq millions de pages par mois, ainsi que nombre d'autres services dédiés (sauvegarde, cron, base de données MySQL Google Store Engine...). Vous pouvez avoir jusqu'à dix applications par développeur et il est possible d'obtenir plus de capacités pour vos applications contre facturation.

Ce service repose sur une architecture complexe, ce qui impose certaines limitations, en particulier, les accès en écriture au système de fichier local sont impossibles. Aussi, ne vous attendez pas à installer votre application et à ce qu'elle fonctionne sans problème immédiatement. Outre le fait qu'il faut lui adjoindre une configuration spécifique pour fonctionner sur le Google App Engine, il vous faudra la tester en profondeur pour détecter les problèmes éventuels liés à la nouvelle plate-forme. Il peut être préférable d'écrire des applications dédiées ou au moins d'isoler les codes dépendants de cette plate-forme pour faciliter une éventuelle migration.

Car, pourquoi utiliser Google App Engine ? À cela, Google vous donnera de bonnes réponses : facilités de démarrage de développement et de mise en œuvre, capacité à changer d'échelle automatiquement et surtout la fiabilité, les performances et la sécurité de l'infrastructure de Google. Et tout cela n'est pas rien. Imaginez : vous développez dans votre garage une application web géniale qui pourrait intéresser des millions d'utilisateurs mais n'avez ni les moyens, ni les compétences pour développer une infrastructure de cette ampleur. Vous vous retrouvez avec un joli morceau de code à peu près sans utilité, cherchant à convaincre des partenaires pour trouver un financement... et vous n'êtes peut-être pas doué pour cela. Google App Engine est alors fait pour vous : vous pouvez mettre votre application à la disposition de tous et commencer à voir votre projet évoluer tout de suite, sans avoir besoin d'un apport financier initial. Autrement, App Engine peut intéresser des entreprises qui souhaitent minimiser leurs coûts d'infrastructure informatique tout en gagnant en fiabilité. Ou encore des associations aux moyens limités peuvent s'offrir ainsi une solution de première classe.

Pourquoi Google a-t-il mis si longtemps à proposer PHP parmi les langages supportés par App Engine ? À ma connaissance, Google ne s'est pas expliqué là-dessus. On peut spéculer sur le fait que le porteur du projet, Guido van Rossum, est également le fondateur du langage Python, ou que Google n'utilise pas PHP en interne... Toujours est-il que lors du Google I/O, ils ont pris acte d'un fait qu'ils sont bien placés pour connaître : 75 % des sites web fonctionnent avec PHP. Une part de marché qu'il était difficile de continuer à ignorer.

Avertissement : au moment où j'écris ces lignes, les demandes de création d'applications sur le Google App Engine sont soumises à approbation (cette mesure devrait être supprimée lorsque l'équipe Google estimera que ce projet sera arrivé à maturité...). Après plus d'un mois d'attente, je n'ai pas encore reçu de réponse et les demandes semblent traitées au compte-goutte. Aussi, hors la partie concernant le serveur de développement proprement dite, et ce qui est testable dans cet environnement, cet article a été en partie réalisé en s'appuyant sur la documentation officielle. Celle-ci et

d'ailleurs l'ensemble du support de PHP sont décrits comme étant encore à un stade précoce, susceptibles d'évoluer rapidement. Cet article n'est donc pas à considérer comme un guide d'utilisation de Google App Engine mais comme un aperçu approfondi qui vous aidera à déterminer si cet outil saura répondre à vos besoins.

Pour être inscrit sur la liste blanche des applications autorisées, il faut en faire la demande motivée en remplissant un formulaire accessible à l'adresse suivante : https://gaeforphp.appspot.com/register

Il faut ensuite attendre un mail vous indiquant que votre demande a été acceptée ou refusée. Cette démarche devrait être provisoire et, avec un peu de chance, vous n'aurez plus à l'effectuer lorsque ce magazine arrivera dans vos mains. Si vous obtenez une erreur 400 « Invalid runtime or the current user is not authorized to use it », lors de l'upload de votre application (voir plus loin), il se peut que la cause en soit là. Ou alors que vous n'avez pas utilisé le mail d'un utilisateur autorisé à faire cette action.

2 Votre environnement de développement

La première chose à faire est d'écrire une première version de votre application. Pour ce faire, il convient de mettre en place un environnement de développement qui permette de reproduire en local le comportement de Google App Engine. Mais comment ? Google vous propose de le télécharger.

Mais avant de le mettre en place, il faut vous assurer de quelques prérequis qui ne présentent normalement pas de difficultés particulières : il vous faut avoir installé Python 2.7, PHP 5.4 et MySQL 5.5. Prenez garde toutefois à bien disposer de l'exécutable **php-cgi**, qui peut se trouver dans un autre paquet de votre distribution que l'habituel paquet **php5**. Sous Ubuntu et Debian, il s'agit du paquet **php5-cgi**. Une fois vous être assuré de tout cela, ouvrez un shell et placez-vous à l'endroit où vous voulez installer le kit de développement (Software Development Kit ou SDK) et lancez les commandes :

```
$ wget http://commondatastorage.googleapis.com/appengine-php/
appengine-php-sdk-1.8.0.zip
$ unzip appengine-php-sdk-1.8.0.zip
```

Vous verrez alors apparaître un répertoire **google_appengine** contenant tout le nécessaire. À partir de là, je dirais, développez votre application « normalement ». Mais ce serait sans doute trop simple et il y a un certain nombre de précautions à prendre. Pour vous montrer les particularités d'un développement destiné à App Engine, le mieux est de vous en montrer un exemple.

2.1 Hello, World !

Et pour suivre notre bonne vieille tradition, lançons dans une nouvelle implémentation de cette application dont le succès laisse rêveur, Hello, World. Pour le plaisir, voici les deux lignes nécessaires à sa réalisation, que nous allons placer dans un fichier **helloworld.php** que nous placerons où vous voulez, dans un répertoire **helloworld** créé pour l'occasion :

```
<?php
print 'Hello, World !';
```

Maintenant pour que notre application fonctionne avec App Engine, il faut lui adjoindre un fichier de configuration, **app.yaml**. Celui-ci est rédigé dans le langage Yaml (http://www.yaml.org), un format de représentation de données sérialisées relativement simple à utiliser. Pour notre exemple, voici quel est son contenu :

```
application: helloworld
version: 1
runtime: php
api_version: 1
handlers:
- url: /.*
  script: helloworld.php
```

La première ligne indique l'identifiant de votre application au sein de Google App Engine. Vous devrez le choisir au moment où vous enregistrerez votre application auprès de leurs services, où il devra être unique parmi toutes les applications déjà présentes. Tant que vous n'en n'êtes pas là, vous pouvez choisir ce qui vous chante.

La seconde ligne indique le numéro de version de votre application. Si, lorsque vous téléchargez votre code sur les serveurs de Google vous modifiez ce numéro, les versions antérieures seront conservées à l'abri et vous pourrez éventuellement les restaurer depuis votre console d'administration.

Les deux lignes suivantes indiquent que votre application utilise le runtime PHP en sa version 1. D'autres valeurs possibles sont **`python27/1`**, **`go/go1`**, mais elles n'entrent pas dans le cadre de cet article.

Les trois dernières lignes permettent de créer un handler définissant que le script **`helloworld.php`** doit traiter toutes les requêtes correspondantes à l'expression régulière **`/.*`**, c'est-à-dire absolument toutes. Cela est bien rudimentaire, mais nous verrons plus tard comment faire mieux.

Une fois cela fait, vous pouvez lancer le serveur de développement du Google App Engine grâce à la commande :

```
$ /chemin/d/installation/google_engine/dev_appserver.py \
--php_executable_path=/usr/bin/php-cgi \
/chemin/vers/votre/projet/helloworld/
```

Dès lors, vous pouvez ouvrir votre navigateur vers l'url http://localhost:8080 et, si tout s'est bien passé, vous devriez voir votre application fonctionner. Vous pouvez également consulter votre serveur local sur le port 8000 et vous y découvrirez une interface d'administration.

Selon l'utilisation de votre machine, vous pourriez avoir besoin d'utiliser d'autres ports. Pour cela, il faut ajouter quelques arguments à la commande ci-dessus :

- **`--admin_port`** : permet de modifier le port de la console d'administration,
- **`--port`** : à utiliser pour régler le port de votre application.

```
$ /chemin/d/installation/google_engine/dev_appserver.py --php_executable_path=/usr/bin/php-cgi
--admin_port=8001 --port=8081 /chemin/vers/votre/projet/helloworld/
```

Pour arrêter ce serveur, un Control+C dans le terminal où vous l'avez lancé est la méthode recommandée.

La documentation complète sur le serveur de développement, en anglais, est consultable à l'adresse : https://developers.google.com/appengine/docs/php/tools/devserver.

3 Écrivez votre application

3.1 La configuration de l'application

Contrairement aux habitudes, App Engine ne sert pas directement les fichiers de l'arborescence de votre projet tant que l'application n'a pas été configuré pour. Évidemment, vous voudrez le faire pour un certain nombre de ressources comme les feuilles de styles, les images, les vidéos, etc. Pour ce faire, il est nécessaire de modifier le fichier **`app.yaml`** et de lui indiquer quels répertoires contiennent des ressources statiques et à quelles urls les associer. Cela vous obligera à bien définir les répertoires contenant ces ressources.

Prenons un exemple : supposons que vous avez placé vos fichiers de styles dans un répertoire **`/css`**. Voici comment il faudra modifier le **`app.yaml`** du projet **`helloworld`** :

```
application: helloworld
version: 1
runtime: php
api_version: 1

handlers:
- url: /css
  static_dir: css

- url: /.*
  script: helloworld.php
```

En fait, il s'agit de définir un nouvel handler pour les url commençant par **`/css`** et de le paramétrer pour utiliser les ressources statiques de ce répertoire. Attention, vous devrez bien placer ce nouvel handler avant celui que nous avons créé précédemment. En effet, lorsque l'App Engine doit traiter une requête, il s'arrêtera au premier handler qui lui correspond. Si vous ne le mettez pas dans le bon ordre, alors vous n'aurez jamais que votre script qui sera accessible.

La référence complète, en anglais, est consultable à l'adresse : https://developers.google.com/appengine/docs/php/config/appconfig.

3.2 Restrictions

La plus grosse restriction concerne le système de fichiers local. Il est impossible à votre application d'y écrire. Pour stocker des données, il vous faudra recourir à la base de données ou au Google Cloud Storage (**`https://developers.google.com/appengine/docs/php/googlestorage/overview`**) mais, attention, ce dernier service n'est pas gratuit. Le plus gros problème qui est soulevé ici est celui du traitement des fichiers uploadés par les utilisateurs. La procédure recommandée

(https://developers.google.com/appengine/docs/php/googlestorage/overview#uploading-to-gcs) utilise le Google Cloud Storage...

Lors de la définition d'un nouvel handler dans le fichier **app.yaml**, sachez que Google a réservé certaines url qu'il ne vous faudra en aucun cas utiliser à savoir **/_ah/** et **/form**.

Certaines fonctions ont été désactivé, en particulier celles effectuant des appels systèmes. Vous en trouverez la liste complète ici : **https://developers.google.com/appengine/docs/php/runtime#Function-Support**

Certaines autres, désactivées par défaut (**https://developers.google.com/appengine/docs/php/runtime#Functions-That-Must-Be-Manually-Enabled**), peuvent être réactivées en utilisant un **php.ini** placé à la racine de votre projet grâce à la directive **google_app_engine.enable_functions**. Ce fichier peut d'ailleurs être utilisé pour effectuer les modifications des variables PHP qui seraient nécessaires à votre application.

Certains types de flux sont désactivés également : **ftp://**, **data://**, **phar://**, **rar://**, **ssh2://**...

Il est toutefois possible de contacter des serveurs externes depuis votre application mais pour cela vous devrez vous appuyer sur l'Url Fetch PHP API (**https://developers.google.com/appengine/docs/php/urlfetch/overview**).

3.3 Stocker des données

Pour les autres langages,le Google App Engine fournit au moins une solution gratuite pour stocker des données, il s'agit de Datastore, que vous verrez dans la console d'administration. Malheureusement, cette solution n'est pas disponible (encore?) pour PHP. Les deux autres solutions existantes sont toutes deux payantes. Il existe toutefois une alternative très expérimentale : il s'agit d'utiliser le Datastore depuis PHP, mais il s'agit du travail d'un développeur isolé. Mais vous voudrez peut-être y jeter un coup d'œil.

Remarquons au passage que la facturation est calculée selon l'usage que vous aurez de ces services, ce qui implique que si vous avez su correctement monétiser votre projet, le coût devrait être proportionnel au gain... Évidemment, si votre projet est un service gratuit sans publicité respectant la vie privée... il vous faudra passer votre chemin.

3.3.1 Google Cloud SQL

Google Cloud SQL est à comprendre comme une base de données MySQL fonctionnant dans le nuage de Google. Il n'y a donc pas lieu de communiquer autrement avec elle que vous ne le faites d'ordinaire. Après avoir créé l'instance, la seule précaution à prendre est dans la manière de se connecter à la base et au serveur.

Pour créer l'instance, il faut passer par la console de gestion. Tout d'abord, il faut activer le service : dans le menu de gauche, cliquez sur « Services », trouvez « Google Cloud SQL » dans la section principale, basculez l'interrupteur, indiquez tout ce qu'il faut pour que Google puisse récupérer ce que vous lui devez, etc.

Une fois cela fait, une nouvelle entrée apparaît dans le menu latéral : Google Cloud SQL. Cliquez-la, puis cliquez sur « New instance ». Vous indiquez alors le nom de votre instance, en minuscule exclusivement. Ce nom sera automatiquement préfixé par le nom de votre projet, ce qui vous donnera un nom complet ressemblant à **mon_projet:instance1**. Indiquez si vous voulez localiser votre instance en Europe ou aux États-Unis (selon l'emplacement de vos utilisateurs, pas de votre domicile). Choisissez une taille pour votre instance et un plan de facturation. Choisissez un mode de synchronisation, synchrone ou asynchrone : le premier est plus lent mais vous ne prenez aucun risque de perte de données, celles-ci étant immédiatement synchronisées entre les différents datacenters ; le second est à peine moins sûr mais les requêtes de synchronisation sont traitées en arrière-plan sans bloquer le fonctionnement de l'application, il y a donc un (faible) risque de (petite) perte de données en cas d'incident majeur. À moins d'avoir des raisons majeures, choisissez le premier. Votre instance est par ailleurs sauvegardée une fois par jour, à moins que vous ne modifiez cette planification.

Si vous avez plusieurs projets et que vous souhaitez qu'ils partagent un accès à cette base, vous pouvez les ajouter dans la section « Authorized applications », accessible également plus tard.

Enfin, cliquez sur « Create ». Voilà, il ne reste plus qu'à vous y connecter.

Pour cela, vous pouvez utiliser les trois grandes API MySQL de PHP, soit via un objet PDO, via **mysql_connect()** ou encore via un objet mysqli. (Attention ! **mysql_connect()** faisant partie d'une API dépréciée, il se pourrait que son support par le Google App Engine disparaisse un jour ! Évitez de l'utiliser.)

Pour vous connecter en utilisant PDO, vous utiliserez la syntaxe suivante :

```
$db = new PDO('mysql:unix_socket=/cloudsql/mon_projet:instance1;charset=utf8',
  'utilisateur_google',
  'mot_de_passe_google'
);
```

Et pour mysqi :

```
$bd = new mysqli(null, 'utilisateur_google','mot_de_passe_google',
null, null, '/cloudsql/mon_projet:instace1') ;
```

3.3.2 Google Cloud Storage

Il s'agit du système de fichier « dans les nuages » de Google, seul emplacement que vous pourrez utiliser pour effectuer un accès en écriture à un système de fichier. En ce sens, il complète Google Cloud SQL, les deux ensembles vous permettant de faire fonctionner une application d'une manière presque classique.

Pour utiliser le Google Cloud Storage, vous devez l'activer depuis la console de gestion de la même façon que vous avez procédé pour Google Cloud SQL. Une fois cela effectué, vous devrez créer ce que Google appelle un bucket, qu'on pourrait traduire littéralement par « seau »...

Pour la création d'un bucket, le mieux est de procéder en ligne de commande à l'aide de l'utilitaire **gsutil** que vous trouverez dans le SDK. gsutil comprend un certain nombre de commandes que vous utiliser couramment telles que **cp**, **mv**, etc. Vous en trouverez la référence complète sur cette page : https://developers.google.com/storage/docs/gsutil?hl=fr.

Pour créer un bucket, vous devrez utiliser la commande **mb** :

```
$ gsutil mb -l EU -p project_id gs://mon_bucket
```

Cette commande crée un bucket en Europe pour mon projet ayant pour id **project_id** à l'emplacement gs://mon_bucket.

Alternativement, il est possible de parcourir et gérer son espace sur le Google Cloud Storage avec le navigateur en ligne à l'adresse : https://developers.google.com/storage/docs/gsmanager?hl=fr.

Ensuite, il vous faudra autoriser votre application à utiliser cet espace. Pour cela, dans la console d'administration du Google App Engine, allez dans Administration / Application settings. Là, copiez le service account name. Il est de la forme application-id@appspot.gserviceaccount.com. Ce service account name, vous devrez l'utiliser dans la console Google API pour l'ajouter comme membre de l'équipe du projet qui contient le bucket.

Si cela ne fonctionnait pas en passant par la console Google API, il vous faudrait vous en remettre à nouveau à gsutil, tout d'abord pour récupérer le fichier des ACL, puis pour les renvoyer une fois modifié.

Pour les récupérer dans un fichier local **myAcl.txt** :

```
$ gsutil getacl gs://mon_bucket > myAcl.txt
```

Éditez le fichier dans votre éditeur de texte préféré et ajoutez les lignes suivantes :

```
<Entry>
  <Scope type="UserByEmail">
    <EmailAddress>
      your-application-id@appspot.gserviceaccount.com
    </EmailAddress>
  </Scope>
  <Permission>
    WRITE
  </Permission>
</Entry>
```

Puis enregistrez les nouvelles ACL à l'aide de la commande :

```
$ gsutil setacl myAcl.txt gs://mon_bucket
```

La documentation indique également que certaines fonctionnalités requièrent un objet ACL par défaut pour fonctionner et qu'il vous faut donc le créer. Pour cela, il vous faut rééditer le fichier myAcl.txt, et dans la portion que vous avez ajoutée, remplacer **WRITE** par **FULL_CONTROL**. Et enregistrer ce nouveau fichier dans le cloud avec la commande :

```
$ gsutil setdefacl myAcl.txt gs://mon_bucket
```

Pour terminer, il vous faudra créer un contexte de flux dans PHP, avant de pouvoir simplement y utiliser le protocole **gs:** pour accéder à vos fichiers :

```
<?php
$options = [ "gs" => [ "Content-Type" => "text/plain" ]];
stream_context_set_default($options)
file_put_contents('gs://mon_bucket/monfichier.txt', 'Hello, world!');
```

Attention ! Il est important de savoir qu'une fois le fichier refermé (ce qui est le cas ici, ou si vous appelez **fclose()**), vous n'aurez plus la possibilité d'ajouter ou d'y modifier des données ! Il vous faudra créer un nouveau fichier de même nom qui l'écrasera si vous voulez en modifier le contenu.

Reste une difficulté encore : la gestion des fichiers uploadés par les utilisateurs. Le Google App Engine ne vous permet pas de procéder de la façon habituelle, il vous faudra faire appel à son API. L'avantage est que votre application n'est pas sollicitée pendant l'upload, seulement quand il est terminé, ce qui permet aux utilisateurs d'envoyer des fichiers bien plus grands que d'ordinaire (la documentation parle d'upload de 100TB!).

Imaginons que le script qui doit traiter l'upload terminé s'appelle **upload_handler.php**. Mais vous ne pourrez pas l'utiliser directement dans votre formulaire, il vous faudra créer une url temporaire d'upload (que vous devez commencer à utiliser dans les dix minutes, que vous ne devez en aucun cas modifier) que vous allez utiliser dans votre formulaire en valeur de la propriété **action**.

Voici comment obtenir cette url :

```
<?php
require_once 'google/appengine/api/cloud_storage/CloudStorageTools.php';
use google\appengine\api\cloud_storage\CloudStorageTools;
```

```
$options = [ 'gs_bucket_name' => 'my_bucket' ];
$upload_url = CloudStorageTools::createUploadUrl('/upload_handler.php',
$options);
```

Et voici comment l'utiliser dans votre formulaire HTML !

```
<form action="<?php echo $upload_url?>" enctype="multipart/form-data" method="post">
<p>
Fichiers à télécharger: <br>
<input type="file" name="uploaded_files" size="40">
</p>
<input type="submit" value="Envoyer">
</form>
```

À partir de là, dans votre fichier **upload_handler.php**, la variable **$_FILES** sera alimentée comme elle l'aurait été par un formulaire ordinaire.

3.3.3 pQg

Il existe en Hollande, pays de la Liberté parfois plus que le nôtre, un doux malade du nom de Herbert Groot Jebbink qui a eu l'idée de permettre à PHP d'utiliser le Datastore gratuit de Google depuis l'App Engine en suivant une syntaxe de type SQL. Je ne l'ai pas testé, je ne sais pas comment ça marche et je ne suis pas même sûr de vouloir le savoir. Je crains une alchimie diabolique. Le projet n'a pas de site dédié à l'heure où j'écris et il vous faudra parcourir le blog, heureusement peu fourni, de son auteur pour découvrir les pages qui vous intéressent. La plus récente pour le moment est : http://herbert-groot-jebbink.blogspot.fr/2009/06/pqg-new-release-new-name.html. (Il semble bien qu'il y ait un problème d'année, il s'agit bien d'un article paru en juin 2012 et non 2009!)

À vos risques et périls !

3.4 Sécuriser des URL

Vous aurez certainement besoin de sécuriser certaines url de votre application, soit qu'il s'agisse de fonctionnalités nécessitant un utilisateur identifié, soit qu'il s'agisse d'une interface d'administration qui vous est propre ou encore de crypter des données confidentielles durant leur transit sur le réseau. Naturellement, vous pouvez gérer cela de manière interne à votre application (ce que je vous recommande), mais le fichier **app.yaml** permet de le faire avec une simplicité déroutante.

Vos handler d'url acceptent effectivement deux paramétrages permettant de déterminer le comportement au regard du cryptage de la communication et de l'identification de l'utilisateur :

- le paramètre **login** qui peut prendre trois valeurs : **optionnal** (par défaut), tous les utilisateurs peuvent y accéder ; **required**, accès restreint aux utilisateurs identifiés ; ou **admin**, seuls les utilisateurs mentionnés comme administrateurs dans la console d'administration de l'application peuvent y accéder ;
- le paramètre **secure** qui peut également prendre trois valeurs : **optionnal** (par défaut), **never** ou **always** ; le comportement par défaut est d'autoriser l'accès aux url en http ou en https, **never** redirige le protocole https vers le protocole http et inversement pour **always**.

Par ailleurs, il est possible d'intégrer certaines pages directement à la console d'administration de Google App Engine, ce qui rend la chose plus confortable. Pour cela, il faut ajouter la directive **admin_console**. Celle-ci accepte comme paramètres une liste **pages**, contenant pour chaque entrée deux paramètres **name** et **url**, définissant respectivement le nom à faire apparaître dans l'interface d'administration et la page correspondante à l'intérieur de votre site :

```
admin_console:
  pages:
  - name: Administration avancee
  url: /admin
```

Les pages en question seront affichées par l'intermédiaire d'une iframe. Attention, le nom de la page ne doit contenir que des caractères ASCII.

Remarquez que le fait de faire apparaître une page dans la console d'administration ne la sécurise pas. Les indications que nous avons données dans le paragraphe juste précédent doivent tout de même être suivies.

4 Publiez votre application

4.1 Créer le projet

La première chose dont vous aurez besoin pour pouvoir publier votre application est un compte Google valide mais cela est certainement déjà le cas et, de toute façon, trivial.

Ensuite, rendez-vous sur la page de démarrage de Google App Engine : https://appengine.google.com/start.

Cliquez sur « Create Application ». Là, il vous sera certainement demandé de valider votre compte par un appel téléphonique ou un SMS.

Ensuite, vous aurez à renseigner un identifiant et un titre pour votre application. Sur la même page, vous devrez indiquer quels utilisateurs pourront accéder à votre application : soit toutes les personnes ayant un compte Google, soit les personnes ayant un compte sur un domaine Google Apps (les utilisateurs d'une entreprise ayant souscrit au service en ligne Google App for Business), soit à tout utilisateur disposant d'une identification OpenID. Pour une application publique, ma préférence personnelle va à cette dernière option, bien qu'elle soit encore expérimentale à l'heure où j'écris ces lignes : c'est la seule qui vous laisse, vous et vos utilisateurs, libres de Google. À partir de là, vous avez accès à une interface d'administration de votre application. Mais celle-ci est encore limitée dans la mesure où vous n'avez pas encore uploadé votre projet. Nous y reviendrons donc plus tard.

4.2 Déployer

L'upload de votre application sur les serveurs de Google s'effectue par cette commande :

```
$ /chemin/d/installation/google_engine/appcfg.py update -R --runtime=php /
chemin/vers/votre/projet/
```

Le chemin indiqué vers votre projet doit contenir immédiatement le fichier **config.yaml**. Il vous sera demandé de vous identifier à l'aide de votre compte Google. Ce ne sera plus le cas par la suite, **appcfg.py** créant une sorte de cookie sur votre machine permettant la persistance de votre identification. Mais auparavant, n'oubliez pas de mettre à jour le champ application de votre fichier config.yaml si vous n'avez pas pu utiliser le même que celui que vous avez défini lors de la configuration initiale de votre projet (chapitre 2.1).

4.3 Tester en environnement réel

Il vous est possible de tester votre application déployée sur les serveurs de Google sans la rendre publique. En effet, le Google App Engine conserve plusieurs versions de votre application au fur et à mesure que vous la mettez à jour, à condition que l'indication de version du fichier **app.yaml** vu plus haut ait été modifié. Il faut vous connecter à la console d'administration pour indiquer quelle version doit être utilisée par défaut. Les autres versions restent accessibles par leurs propres URL qui ont la forme : http://**version_id**-dot-latest-dot-**your_app_id**.appspot.com.

Testez avec cette url, puis, lorsque les tests sont concluants, utilisez la console d'administration pour basculer la version à utiliser par défaut.

5 Intégration des services Google

Le Google App Engine vous offre la possibilité d'intégrer des fonctionnalités des autres services Google, ce qui est très vaste. Cela peut se révéler très intéressant, mais cela rendra votre code largement dépendant de Google, ce qui peut se révéler gênant si un jour vous souhaitez changer de crèmerie (ou si la crème change de goût, ou si la crémière vous met dehors...) J'ai donc décidé de ne pas explorer cette possibilité ici. La plus intéressante est largement documentée par Google, il s'agit de l'utilisation de l'identification Google Account : https://developers.google.com/appengine/docs/php/gettingstarted/usingusers.

Conclusion

Bénéficier des infrastructures à toute épreuve (ou presque) de Google pour sa propre application web écrite en PHP, voilà qui relevait encore il y a peu du rêve. Maintenant que cela est devenu une réalité, on ne peut que s'en réjouir, avec un regret toutefois : il est dommage que Google ne propose pas un minimum d'espace de stockage SQL et fichiers gratuitement, cela fait paraître son offre d'hébergement d'application un peu moins gratuite et la fait ressembler à un amorçage, en même temps que cela freine et, sans doute dans certains cas, interdit le développement d'applications à but non-lucratif sur cette plate-forme. Ajoutons encore quelques grammes de prudence et de vigilance pour nuancer notre enthousiasme : tout d'abord, la tentation peut être forte d'écrire un code fortement lié aux autres services Google et construire une entreprise sur de telles fondations est toujours risqué ; enfin, le scandale autour de l'affaire PRISM montre qu'il faut traiter avec prudence les données collectées auprès de vos utilisateurs. ■

Aruba Cloud, les solutions IaaS qui répondent à chacun de vos besoins.

CLOUD COMPUTING

- Créez, activez et gérez vos VM.
- Choisissez parmi nos 3 hyperviseurs.
- Maîtrisez et planifiez vos ressources CPU, RAM et espace disque.
- Uptime 99.95% garanti par SLA.

CLOUD OBJECT STORAGE

- Créez vos espaces et stockez vos données en toute sécurité.
- Une solution qui s'adapte à vos besoins : Pay as you Go, ou formule prête à l'emploi.
- Bande passante et requêtes illimitées.

LE CLOUD PAR ARUBA

- Ubiquité : choisissez votre pays et datacenter.
- Interopérabilité : API et connecteurs.
- Agnosticisme : choisissez votre hyperviseur.
- Scalabilité : étendez votre infrastructure à l'infini.
- Transparence : pas de coûts d'activation, ni coût caché.
- Pay as you Go : ne payez que ce que vous consommez.

Aruba, le bon partenaire pour bénéficier de la puissance d'un acteur majeur qui considère que chaque client, dans chaque pays, est unique. **MY COUNTRY. MY CLOUD.**

aruba CLOUD

arubacloud.fr | TÉL : 0810 710 300 (COÛT D'UN APPEL LOCAL)

L'EXCELLENCE DU DÉVELOPPEMENT OPEN SOURCE

LinStudio

POUR VOS APPLICATIONS WEB

CHOISIR LinStudio C'EST CHOISIR :

- **LE SPÉCIALISTE LOGICIELS LIBRES POUR LA MISE EN ŒUVRE DE VOS APPLICATIONS WEB ET MOBILES**

DEPUIS PLUS DE 10 ANS, LINSTUDIO A DÉVELOPPÉ DES CENTRES DE COMPÉTENCE DE HAUT NIVEAU SUR LES TECHNOLOGIES DU WEB LES PLUS PERFORMANTES: DRUPAL, SYMFONY.. NOS INGÉNIEURS METTENT EN PLACE DES PLATE-ORMES WEB « INDUSTRIELLES » ET S'APPUIENT SUR DES MÉTHODES ET OUTILS PERFORMANTS. ÉPAULÉ PAR L'EXPÉRIENCE ET LE SAVOIR FAIRE DES EXPERTS DE LINAGORA, LINSTUDIO EST UN EXPERT RECONNDU POUR L' INTÉGRATION FORTE DES APPLICATIONS WEB DANS LES SYSTÈMES D'INFORMATION.

- **UN ACCOMPAGNEMENT PROJET « SUR-MESURE »**

ACCOMPAGNEMENT DE VOS PROJETS DIGITAUX, QUE CE SOIT LORS DE MISSIONS D'ÉTUDES, D'ASSISTANCE OU DE FORMATIONS, EST ARTICULÉ AUTOUR DE PROCESSUS QUI INTÈGRENT SYSTÉMATIQUEMENT UNE GESTION DES RISQUES ADAPTÉE, QUI SONT LES GARANTS DE LA QUALITÉ DES LIVRABLES ET DES LIVRABLES

UNE EXPÉRIENCE CENTRÉE AUTOUR DE L'EXPÉRIENCE UTILISATEUR

NOTRE STRATÉGIE CONSTITUE LA LIGNE DIRECTRICE DE NOS ARCHITECTES DÈS LA PHASE DE CONCEPTION DE VOS PROJETS

DANS LE WEB TOUT LE MONDE DIT QU'IL EST LE MEILLEUR ...

ALORS AUTANT TRAVAILLER AVEC LES PLUS SYMPAS !

- **UN ACTEUR MAJEUR DU MARCHE AVEC UN IMPORTANT RÉSEAU DE** ... AVEC LES **COMMUNAUTES OPEN SOURCE**. LINSTUDIO PARTICIPE ET SOUTIENT DES ÉVÉNEMENTS ASSOCIATIFS COMME DRUPALMEETUP, DRUPALCAMP... ET EST UN MEMBRE ACTIF DE DRUPAL ASSOCIATION ET COMPTE PARMI SES INGÉNIEURS PLUSIEURS CONTRIBUTEURS AUX CMS OPEN SOURCE

- **DES RÉFÉRENCES PRESTIGIEUSES DE GRANDS CLIENTS À LA RECHERCHE DE SERVICE « HAUTE-COUTURE »** COMME DES ADMINISTRATIONS OU COLLECTIVITÉS LOCALES (RÉGION ÎLE DE FRANCE, SERVICE D'INFORMATION DU GOUVERNEMENT, DÉFENSEUR DES DROITS, VILLE DE TOULON...), DES ACTEURS DANS LES DOMAINES DES MÉDIAS (FRANCE 24, L'HUMANITÉ, REPORTEURS SANS FRONTIÈRES), DU LUXE OU DE LA « GRANDE CONSO »..... ;

AVEC L'OPEN SOURCE, PASSEZ À LA VITESSE SUPÉRIEURE SUR LE WEB !

LINAGORA

www.linagora.com

www.linstudio.com