ANEXO 6: AFS

Índice general

Capítulo 1

Sistema de Ficheros Distribuído con OpenAFS

Contents

- Anotación: adelantamos ya que el módulo del kernel de OpenAFS puede ser cargado una vez pero no recargado: corrompe la memoria y el ordenador se para (así nos ha ocurrido tanto en máquinas reales como bajo Qemu). Desgraciadamente, de hecho, ese módulo no se carga sólo sino que lo carga el cliente AFS cuando se inicia, pero éste a veces mostraba problemas al arranque que necesitaban reiniciarlo... parando la máquina por tanto. Para evitar de raíz esa posibilidad desafortunada, nosotros configuraremos el cliente AFS para que no se arranque automáticamente al inicio sino que podremos hacerlo a mano cuando necesitemos usar a éste. Para ello tendremos en cuenta las dependencias que tiene en ambos sentidos: de un lado el cliente AFS depende de MIT Kerberos, luego no lo iniciaremos hasta que éste esté cargado; y de otro sistemas como el de correo dependerán (cuando los abordemos en otros capítulos) del cliente AFS así que no se arrancan si éste no lo está, por lo que deberemos identificar así la situación y arrancarlos manualmente también.

1.1. Arquitectura de OpenAFS

Muy esquemáticamente, podemos desplegar los siguientes grupos de servicios:

- Grupo "Simple File Server", comprende:

Programa	Función
fileserver	El servidor de ficheros propiamente
volserver	Permite la gestión remota de los volúmenes
salvager	Permite comprobaciones de integridad remotas de los volúmenes

- Grupo "Database Server", comprende:

Programa	Función
ptserver	Base de datos para el sistema de ACL (id de usuarios, permisos...)
vlserver	Base de datos que mapea ficheros a los servidores y vol donde se encuentran
buserver	Gestor de la base de datos relacionado con respaldos a cinta.
kaserver	Relacionado con autenticación. No usado en favor de KERBEROS.

- Otros no desplegados: binary distribution (actualizador binarios), system control machine (actualizador de configuraciones)...

- Cliente:

Programa	Función
afsd	Es el cliente AFS y administrador de la caché, en concreto es la parte en espacio de usuario, pues existe otra en el kernel (módulo openafs).

Nótese que la base de datos vlserver es absolutamente fundamental. En un sistema de ficheros distribuído no podemos acceder directamente al servidor de ficheros para encontrar un recurso pues no sabemos en cuál está. Hace falta una acción de resolución previa que el cliente afsd AFS lleva a cabo contra vlserver.

A bajo nivel, la información se organiza de la siguiente forma: se cede una partición de algún disco duro y se monta bajo un directorio que en general ha de ser /vicepa. En esa partición, el proceso volserver (a petición nuestra), puede crear unos ficheros con un formato especial y que llamamos volúmenes. Dentro de los volúmenes se almacena la información que ve el usuario. El usuario puede ver o manipular información en AFS cuando accede al punto de montaje de AFS en su sistema operativo y que es, por convenio, /AFS/.

Administrativamente, AFS entiende que un despliegue de servidores y clientes para una misma organización es una "celda" o "célula" AFS. Para que celdas de varias organizaciones puedan interactuar, bajo /AFS/ se encuentran directorios que representan el raiz, donde empiezan, los recursos de ficheros y directorios de cada celda. Así los recursos de una organización cuelgan bajo /AFS/<nombre celda1>y los de otra bajo /AFS/<nombre celda2>.

1.2. DKLAB1

1.2.1. Instalación del soporte en el kernel Linux

El cliente y administrador de caché AFS del proyecto OpenAFS está implementado como un módulo en el kernel. Debido a incompatibilidades entre las licencias de Linux y el módulo, Debian no puede redistribuirlo compilado, sólo como código fuente. Al menos, Debian dispone de paquetes configurados para que la compilación sea un proceso trivial a través de los sistemas "module-assistant" y, recientemente, "DKMS".

La diferencia entre ambos es que si el módulo lo registramos y compilamos con DKMS, cada vez que instalemos una nueva versión del kernel Linux el sistema lo detectará y recompilará los módulos para él, sin intervención del administrador; por su lado "module-assistant" requeriría hacerlo manualmente. Esta automatización puede ser o no ser adecuada, de forma que se debe elegir entre un sistema u otro.

Con module-assistant

```
apt-get install openafs-modules-source module-assistant linux-headers-‘uname -r‘


m-a prepare
m-a a-i -t -l ‘uname -r‘ openafs


ls /lib/modules/‘uname -r‘/fs/openafs.ko


# Ya podemos cargarlo:
modprobe -v openafs
```

No es necesario editar /etc/modules para cargar el módulo al arranque, ya que el script de incio (/etc/init.d/openafs-client) comprobará la situación y actuará acorde.

Con DKMS

Si no hemos elegido la vía de module-assisatant y preferimos los automatismos de DKMS:

```
apt-get install build-essential dkms linux-headers-‘uname -r‘
apt-get install openafs-modules-dkms


ls /lib/modules/‘uname -r‘/fs/openafs.ko


# Ya podemos cargarlo:
modprobe -v openafs
```

No es necesario editar /etc/modules para cargar el módulo al arranque, ya que el script de incio (/etc/init.d/openafs-client) comprobará la situación y actuará acorde.

1.2.2. Instalación de OpenAFS

Un servidor de archivos y bases de datos AFS necesita a openafs-fileserver y openafs-dbserver. Un cliente necesita a openafs-krb5 y a openafs-client (que utilizará, además, el módulo del kernel openafs.ko ya instalado).

Para ilustrar más en detalle el despliegue, se instala soporte para ambos roles, todos los paquetes:

```
apt-get install openafs-client openafs-krb5
```

```
-"AFS cell this workstation belongs to:"
casafx.dyndns.org (se suele hacer coincidir con el realm KERBEROS en minu'sculas)
-"Size of AFS cache in kB:"
50000
-"Run Openafs client now and at boot?"
No                  (al no estar la celda afs creada au'n)
-"Look up AFS cells in DNS?"
Yes                 (usara' SRV y AFSDB)
-"Encrypt authenticated traffic with AFS fileserver?"
Yes
-"Dynamically generate the contents of /afs?"
No
-"Use fakestat to avoid hangs when listing /afs?"
Yes
```

- Anotación respecto al tamaño de la caché: en AFS, el cliente anuncia al servidor aquello que cachea, y el servidor entonces se compromete a avisar al cliente cuando algo de esa caché cambie. Claramente, aumentar el tamaño de caché es el recurso del lado del cliente para mejorar las prestaciones (habiendo un límite máximo razonable de 20GB, y mínimo de 10MB), por tanto se debe plantear si es posible declarar un valor superior a esos 50000 kB por defecto. A la vez, es fundamental tener en cuenta que, residiendo la caché en /var/cache/openafs:
  - OpenAFS no maneja nada bien quedarse sin el espacio libre asignado para su caché.
  - Algunos sistemas de archivos no son convenientes como espacio para caché afs, la solución es utilizar una partición libre o crearla virtual desde un fichero: ext2/3/4 parecen ser los formatos soportados [1] y hay reportes favorables sobre btrfs a partir de la versión 1.6[2] de OpenAFS (la que instalaremos aquí). No confúndanse los formatos soportados para la caché del cliente AFS (ext2/3/4) y los soportados para las particiones del servidor AFS: cualquiera POSIX. Recalcaremos ésto más tarde.

[1] http://wiki.openafs.org/AdminFAQ/#3.29%20What%20underlying%20filesystems
[2] https://lists.openafs.org/pipermail/openafs-info/2012-November/038950.html

- Anotación respecto a la posibilidad de encriptar el tráfico AFS: el desempeño no se ve apenas afectado por lo que es conveniente activarlo; de cualquier forma, la recomendación es usarlo en clientes que utilicen canales inseguros. El comando

  ```
  fs getcrypt
  ```

  reporta si está o no activada y

  ```
  fs setcrypt
  ```

  lo activa.

Respecto al rol de servidor de archivos y gestores de bases de datos AFS, podemos proceder a su instalación:

```
apt-get install openafs-fileserver openafs-dbserver
```

```
-"Cell this server serves files for (fileserver):"
casafx.dyndns.org
```

- Con la respuesta a esa pregunta, debconf hace que el servidor forme parte de la celda indicada, almacenando dicha información en /etc/openafs/server/ThisCell (también en /etc/openafs/server/CellServDB del que se hablará más adelante), y equivale a lanzar el comando

  ```
  bos setcellname -server dklab1.casafx.dyndns.org. casafx.dyndns.org -localauth
  ```

Sólo el componente de OpenAFS bosserver (de "Basic Overseer Server", un watchdog o monitorizador) queda en ejecución por ahora; más tarde tras crear la celda se creará un /etc/openafs/BosConfig que le dirá a bosserver qué otros componentes debe arrancar y monitorizar.

```
ps aux | (sleep 1; egrep \
"(bosserver\
|fileserver\
|volserver\
|vlserver\
|ptserver\
|salvage\
|buserver\
|upclient\
|upserver\
|afsd)")
```

```
root 1338  0.0  0.1   5140  3808 ? Ss   22:13 0:00 /usr/sbin/bosserver
```

También, puesto que por un lado debconf, al instalar la parte de cliente AFS, mencionó el descubrimiento de servicio a través de DNS y por otro lado acabamos de instalar el software que ofrece ese servicio, nos parece un buen momento para recordar que en DNS no están registrados los servidores de ficheros (componente fileserver del servicio AFS) sino los gestores de base de datos AFS (componentes ptserver y vlserver del servicio AFS). Al ser un sistema distribuído, con ellos ha de iniciarse la comunicación: no podemos acceder a un recurso sin antes preguntar a vlserver dónde se encuentra (en qué fileserver) dicho recurso[3]. Se volverá a insistir sobre ésto.

1.2.3. Administración de las IP de los servidores y de sus entradas en la VLDB

En OpenAFS se establecen diversas comunicaciones entre sus componentes:

- los clientes contactan con el gestor de bases de datos de volúmenes (vlserver) para resolver en qué servidor de archivos se encuentra un recurso. El fileserver también necesita contactar con el los gestores de db en ocasiones.
- los clientes contactan con el fileserver para leer o escribir en el recurso.
- los gestores de bases de datos contactan entre sí para sincronizarse.

[3]Además sin perjuicio, sabemos, de que esté dónde esté físicamente, cambie de servidor o no, será montado en un lugar fijo del espacio de nombres único /afs.

- los fileserver contactan con el vlserver para que tenga noticia de su presencia y de los volúmenes que aloja.

Todas esas comunicaciones son posibles utilizando una dirección IP que permita llegar de unos a otros en la red. Relacionado con ésto, no es infrecuente encontrar servidores que tienen no sólo una, sino varias interfaces y por tanto varias direcciones IP. Inmediatamente ésto nos da la posibilidad de contactar con el equipo incluso aunque algunas de sus interfaces/rutas tengan problemas, pero también, arroja el problema de que el administrador tendría que estar al tanto para seleccionar qué IP's deben usar los componentes para comunicarse, dado que en un escenario típico no podrá probablemente resolverse el problema estrictamente a nivel III (protocolos de enrutamiento etc).

Ante ésto, OpenAFS implementa un sistema para que se puedan utilizar varias IP's y se administre de forma casi automática. También ayuda en los problemas que pueden encontrar fileservers tras NAT. La clave está en que los componentes escanean qué interfaces de red hay activas y las registran unos en los otros, si bien el administrador puede intervenir e indicar qué utilizar y qué no a través de los ficheros NetInfo y NetRestrict. El formato de estos ficheros es, de hecho, una IP por línea, no importa el orden. Veamos:

- NetInfo: cada grupo de componentes (cliente y servidor) lo busca en una localización distinta y lo interpretan también diferentemente:

  - Cliente (en /etc/openafs): registra esas IP en el fileserver para que éste sepa qué posibilidades tiene de acceder a al cliente (pings, callback para avisar de cambios...). Si no hay NetInfo pide al SO la lista de ifaces activas excepto localhost. Véase con: fs getclientaddr
  - Fileserver (en /var/lib/openafs/local): lo usa para saber qué ifaces puede registrar en el vl db server (este "campo" de la db es dinámico, pues) y que se le pueda indicar así a los cache manager cómo acceder al fs que guarda el volumen por el que preguntan; el fileserver server deja esta info en formato binario en /var/lib/openafs/local/sysid. Si no hay NetInfo pide al SO la lista de ifaces activas excepto localhost. Si el fileserver está tras NAT, prefijando la IP de la lista con "f " se puede hacer que se anuncie esa ip aunque el servidor no tenga iface activa con ésta. Véanse las IP registradas en la vldb con: vos listaddrs
  - Db server (en /var/lib/openafs/local, el mismo que usa fileserver): lo usa para anunciar qué interfaces locales puede utilizar ubik para comunicarse con él desde otros db servers. Si no hay NetInfo pide al SO la lista de interfaces activas excepto la de retorno.

- NetRestrict: tanto la información NetInfo como, en su ausencia, la lista que devuelva el SO se puede enmascarar si se crea junto a él un fichero NetRestrict con IP expuestas en NetInfo. El administrador debiera usarlo cuando esté seguro de que todas las demás partes involucradas pueden alcanzarte por el resto de interfaces[4].

[4] https://lists.openafs.org/pipermail/openafs-info/2011-January/035279.html

De los 3 puntos anteriores sobre NetInfo se deduce que el mecanismo NetInfo/NetRestrict se aplica a las comunicaciones cliente-fileserver, cliente-fileserver de nuevo, y db-db. ¿Y las comunicaciones cliente-db o fileserver-db? Podemos comentarlo en este momento de nuestra exposición también:

Primero hemos de puntualizar que las comunicaciones cliente-db pueden implicar a celdas AFS propias o ajenas: un cliente puede acceder a un fichero en nuestra celda o en celdas ajenas si tiene permisos para ello allí. Sin embargo las comunicaciones filserver-db son entre componentes de una misma celda. Entonces, diremos que existe un fichero CellServDB que anuncia dónde están localizados los servidores de bases de datos vlserver y ptserver, y que de nuevo según su localización es leído e interpretado de forma distinta, y puede ser suplementada o no la información no disponible allí con DNS:

- Cuando está localizado bajo /etc/openafs es leído por procesos cliente AFS e indica dónde están los db server de toda celda conocida. Y claro, adicionalmente cuando se pasa el flag -afsdb a afsd, buscará en registros SRV/AFSDB del sistama DNS si la consulta a CellServDB falló. Éste es nuestro caso y como venimos repitiendo debe quedar claro que para acceder a un recurso en una celda AFS propia o ajena, el cliente antes hace una consulta a /etc/openafs/CellServDB y (si yerra ésta) a DNS, así encuentra las bases de datos de esa celda, consulta por la localización del fileserver que almacena el recurso y por fin contacta con aquel. Si el mecanismo NetInfo/NetRestrict está configurado correctamente en esa celda, las IP que ha recibido el cliente son aquellas que le permiten acceder al fileserver (no cuál cree el fileserver que es la suya porque podría estar tras NAT etc; de igual manera ocurre con las IP declaradas en CellServDB para vlserver y ptserver, claro).

- Cuando el fichero CellServDB está bajo /etc/openafs/server, indica inicialmente qué otros gestores de db hay en la celda de la que se ocupa este servido, en nuestro caso "casafx.dyndns.org", y no de otras ya. Efectivamente al comentar el mecanismo NetInfo en relación a la comunicación fileserver-db, hemos dicho que fileserver puede registrar su lista de IP en algún vlserver de su celda, ¿pero cómo accede uno al otro? los db server de su celda están registrados en ese CellServDB.

Pueden consultarse la página man de NetInfo, NetRestrict y CellServDB para más detalles. Por nuestra parte estamos en condiciones ya de tomar algunas decisiones. Con objeto de mostrar el uso del sistema, supondremos que en nuestro despliegue todos los componentes y clientes acceden, por ejemplo, a través de la VPN. Nos interesa entonces enmascarar[5]:

```
printf "192.168.1.10\n10.0.2.15\n" > /var/lib/openafs/local/NetRestrict
```

Respecto a los CellServDB, en este momento no tienen información sobre nuestra celda porque aún no hemos registrado db server alguno, pero tras ello nuestra política será: en el caso del CellServDB de servidor dejaremos registradas las IP (así ocurrirá por defecto),

[5]Nótese que se añade la IP 10.0.2.15 porque, si trabajamos bajo el emulador Qemu, es la IP virtual que éste utiliza en su sistema "user-network", véase capítulo sobre el testbed.

puesto que en ese caso no hay posibilidad de usar DNS, y en el de cliente también pues ayudarán al proceso de registro de nuestra celda, como veremos. Podemos, además, añadir que en cualquier otra máquina hipotética que fuese a hacer de cliente AFS de nuestra celda (es decir, ni dklab1 ni dklab2), obligaríamos a usar DNS y por tanto no haríamos en su CellServDB de cliente referencia alguna a nuestra celda.

Administración IP de los fileserver en las VLDB

Como dijimos, las IP que asocian la base de datos de volúmenes para cada volumen son dinámicas, puesto que el fileserver registra en cada inicio las que considera sus IP, según la presencia o no de NetInfo y NetRestrictions. Más adelante, tras crear nuestra celda y recursos asociados, podremos utilizar comandos como los que se mencionan a continuación para consultar las IP registradas, comprobar su uso o hacer modificaciones:

```
vos listaddrs -localauth # listar'ia todas las direcciones IP
                         # registradas por los fileserver
```

Cómo utilizaría esa información, consultando dónde está un volumen dado:

```
vos listvldb <volumename>
```

Cómo hacer modificaciones directamente en la VLDB (sin parar fileserver y editar Netinfo etc):

```
bos listusers dklab1 -localauth #indicar'ia que' identidades afs tendri'an
                                #privilegios para las modificaciones
vos changeaddr -remove -oldaddr <original IP address>
vos changeaddr -oldaddr <original IP address> -newaddr # precaucio'n, ve'ase
                                                       # su pa'gina man
```

1.2.4. Configuración de la autenticación KERBEROS previamente a las tareas de creación de partición, celda y volúmenes

Todos los fileserver de la célula comparten una llave de principal, que por ser compartida no se llamará afs/<FQDN>@<realm>, sino afs@<realm>; sin embargo la documentación indica[6] que también se puede incluir el nombre de la célula: afs/<cell>@<realm>, útil cuando el nombre de la célula no coincide con el del reino kerberos; puesto que se

[6] http://docs.openafs.org/QuickStartUnix/#HDRWQ53.html

han reportado situaciones [7] en que la forma corta dio problemas no del todo aclarados, usamos la larga aunque cellname y realmname coincidan:

"*Entry for AFS server processes, called either afs or afs/cell. The key is used to encrypt the server tickets that granted to AFS clients for presentation to server processes during mutual authentication, (...) when they need to decrypt server tickets.*"

Procedemos entonces a crear el principal correspondiente:

```
kadmin.local
 > addprinc -policy service -randkey -e des-cbc-crc:normal \
            afs/casafx.dyndns.org@CASAFX.DYNDNS.ORG
 > ktadd -k /etc/keytab.d/openafs.keytab -e des-cbc-crc:normal \
            afs/casafx.dyndns.org@CASAFX.DYNDNS.ORG


klist -tKek /etc/keytab.d/openafs.keytab
```

- El par <tipo encriptación>:<tipo de salt>pasados con -e son obligatorios dadas las limitaciones de la interfaz de OpenAFS a KERBEROS. En en el futuro las llaves serán AES, mientras tanto con la versión 1.4 y, que sepamos, la 1.6 debemos usar DES e indicar a MIT Kerberos que la acepte en el krb5.conf (así lo hicimos, con la línea allow_weak_crypto).

Una vez creado el principal y exportado, en afs el tratamiento del keytab no es anunciarlo (por ejemplo en la variable KRB5_KTNAME como se hizo con slapd) sino cargarlo permanentemente. Es un requerimiento especial que tiene OpenAFS a diferencia de slapd, pero la idea es la misma: permitir a OpenAFS usar el protocolo KERBEROS como servicio. Así:

```
asetkey add 2 \
        /etc/keytab.d/openafs.keytab afs/casafx.dyndns.org@CASAFX.DYNDNS.ORG
```

...donde ese "2" debe ser valor del campo "Key" en la salida de "kadmin.local -q 'getprinc afs/casafx.dyndns.org'" (otra posibilidad, pero requiere autenticación, es haciedo

```
kvno afs/casafx.dyndns.org
```

).

Como resultado, el siguiente fichero se ha creado y la llave puede listarse local o remotamente:

[7] https://lists.openafs.org/pipermail/openafs-info/2009-May/031361.html

1.2.5. Creación de la partición vicepa

La partición estará formateada con algún sistema de archivos POSIX habitual tal como ext3, y montada en el directorio /vicepxx, donde xx va de de a-z y de aa-iv (por tanto afs puede manejar hasta 256 particiones distintas). Entonces, una vez montada la partición ext3 siguiendo esa convención de nombres, OpenAFS creará sobre el nivel de sistema de archivos unos ficheros que representen sus recursos (volúmenes...) con una estructura interna particular que el fileserver sabe interpretar. Nosotros crearemos una partición ext3 y un directorio /vicepa donde montarla. Además, simularemos una partición a través de un fichero, pero en un despliegue real debiera reservarse una partición real.

```
dd if=/dev/zero of=/var/local/vicepa bs=1024 count=76800 #75MB
mknod /dev/loop255 b 7 255 #Usamos la 255 por si contenedores (comparten con SOhos
losetup /dev/loop255
losetup /dev/loop255 /var/local/vicepa
losetup /dev/loop255
mkfs -t ext3 -m 1 -v /dev/loop255   # o ext4
mkdir /vicepa
mount -t ext3 /dev/loop255 /vicepa
ls /vicepa
```

Si no está vacío, entonces fue todo bien y podemos hacer los cambios permanentes entre arranques a través de (/lib/udev/devices/ en nuestro caso particular de montar un fichero, y, ya en general) /etc/fstab:

```
umount /vicepa
losetup -d /dev/loop255


cd /lib/udev/devices/ && mknod loop255 b 7 255


vim /etc/fstab
```

```
...
####fx:
/var/local/vicepa  /vicepa  ext3  defaults,loop=/dev/loop255,rw,auto  0  0
#Puede interesar an~adir noatime para despenalizar el desempen~o;
#por su lado rw y auto ya forman parte de "defaults" y pueden omitirse.
####endfx
```

```
mkdir -p /vicepa
mount /vicepa
ls /vicepa
```

1.2.6. Creación de nuestra celda AFS

Seis llamadas más a las utilerías de openafs son necesarias; no obstante se provee un script en perl que realiza además una batería de comprobaciones para afianzar el proceso.

El script afs-newcell necesita que /etc/openafs/CellServDB esté editado apuntando a los servidores de nuestra futura celda. Por tanto, añadiremos las siguientes dos líneas en la parte superior de dicho fichero:

```
vim /etc/openafs/CellServDB
```

```
>casafx.dyndns.org
10.168.1.1 # dklab1.casafx.dyndns.org
...
```

Vamos allá:

```
afs-newcell
```

```
-"Do you meet these requirements? [y/n]"
y
-"What administrative principal should be used?"
root/admin
...
Now, get tokens as root/admin in the casafx.dyndns.org cell.
Then, run afs-rootvol.
```

Estos son los comandos ejecutados por afs-newcell:

- Primero añade una identidad que, preexistiendo en MIT Kerberos y la db PTS de OpenAFS, puede llevar a cabo acciones privilegiadas de comandos que afectan al servidor local o sus volúmenes (bos; vos, backup). Nótese cómo mapeamos el principal "root/admin" de Kerberos a su nombre equivalente en OpenAFS (según un mapeo preconfigurado ya en OpenAFS): "root.admin". El '.'("period") se utiliza en nombres administrativos especiales[8], de ahí el cambio.

```
bos adduser dklab1.casafx.dyndns.org. root.admin -localauth
# Informacio'n que queda almacenada en UserList:
cat /etc/openafs/server/UserList
```

- Luego registra el servidor relacionado con las ACL (el db server ptserver), el de la localización de los volúmenes (el db server vlserver) y el de archivos (fileserver) y tras ellos configura el monitorizador bosserver (configuración que puede verse en /etc/openafs/BosConfig):

[8] http://docs.openafs.org/AdminGuide/#HDRWQ502.html

```
bos create dklab1.casafx.dyndns.org. ptserver \
    -type simple -cmd /usr/lib/openafs/ptserver -localauth
bos create dklab1.casafx.dyndns.org. vlserver \
    -type simple -cmd /usr/lib/openafs/vlserver -localauth
bos create dklab1.casafx.dyndns.org. dafs \
    -type dafs -cmd '/usr/lib/openafs/dafileserver -p 23 -busyat 600 -rxpck 400
                                                   -s 1200 -l 1200 -cb 65535 -b 240
                                                   -vc 1200' \
            -cmd /usr/lib/openafs/davolserver \
            -cmd /usr/lib/openafs/salvageserver \
            -cmd /usr/lib/openafs/dasalvager -localauth
# En el comando anterior, dafs es el nombre que se da al
# triplete dafileserver, davolserver, salvageserver+dasalvager; efectivamente
# definidos despue's con los "-cmd" precedentes.

bos setrestart dklab1.casafx.dyndns.org. \
    -time never -general -localauth
```

Desde la versión de OpenAFS número 1.6, fileserver, volserver y salvager implementan la funcionalidad "Demand Attach"[9] para permitir tiempos de reinicio cortos y casi independientes del número y tamaño de los volúmenes alojados. Debian distribuye conjuntamente binarios compilados sin y con esta característica, prefijando el nombre del binario con "da" en este último caso. La recomendación en nuevos despliegues es utilizar la implementación con "Demand Attach" y es la que hemos configurado, pero si se busca la máxima estabilidad debiera usarse los binarios tradicionales; consúltese /usr/share/doc/openafs-fileserver/NEWS.Debian.gz.

- Por último asigna el volumen raiz de afs (por convención llamado "root.afs") a la partición a (la "a" de "/vicepa" como se verá); sólo declara, pues tanto el volumen como la partición son recursos ambos que deberemos crear más tarde:

[9] http://wiki.openafs.org/DemandAttach/

```
vos create dklab1.casafx.dyndns.org. a root.afs -localauth
# la "a" hace referencia a la "a" que sufija "/vicep" en /vicepa,
# es decir hace referencia a la particio'n donde se creara' el volumen.
```

Han sido lanzados varios componentes OpenAfs: ptserver, vlserver, fileserver... pero OpenAFS provee otros; por ejemplo implementa un sistema propio de backups de volúmenes a cinta (tape), y que tiene a /usr/lib/openafs/buserver como su componente de gestión de base de datos; vamos a registrarlo a modo de ejemplo:

```
bos create dklab1.casafx.dyndns.org. buserver \
           -type simple -cmd /usr/lib/openafs/buserver -localauth
```

Podemos también ilustrar cómo sería revertir la situación. Efectivamente tendríamos que ejecutar los opuestos a los comandos anteriores. Además de, por supesto, ejecutar apt-get purge sobre los paquetes instalados, deberíamos en ese hipotético caso:

```
# NO ejecu'tese, es so'lo para ilustrar los opuestos de los
# anteriores comandos ejecutados por afs-newcell.
vos remove dklab1.casafx.dyndns.org. a root.afs -localauth
bos shutdown dklab1.casafx.dyndns.org. -localauth -wait
bos delete dklab1.casafx.dyndns.org. fs -localauth
bos delete dklab1.casafx.dyndns.org. vlserver -localauth
bos delete dklab1.casafx.dyndns.org. ptserver -localauth
rm /var/lib/openafs/db/prdb* # O quiza's todo;
                             # tambie'n /vicepa/AFSIDat, Lock, *vol
bos removeuser dklab1.casafx.dyndns.org. root.admin -localauth
#
```

En cualquier caso, podemos comprobar que los distintos componentes de OpenAFS están ejecutándose ya:

```
ps aux | (sleep 1; egrep \
"(bosserver\
|fileserver\
|volserver\
|vlserver\
|ptserver\
|salvage\
|buserver\
|upclient\
|upserver\
|afsd)")
```

```
root 1338  0.0  0.1   5140  3808 ? Ss   22:13 0:00 /usr/sbin/bosserver
root 1342  0.0  0.2   7260  6092 ? S    22:13 0:00 /usr/lib/openafs/ptserver
root 1344  0.0  0.3   9548  8052 ? S    22:13 0:00 /usr/lib/openafs/vlserver
root 1347  0.0  0.4 289892  9772 ? S<l  22:13 0:00 /usr/lib/openafs/dafileserver \
     -p 23 -busyat 600 -rxpck 400 -s 1200 -l 1200 -cb 65535 -b 240 -vc 1200
root 1348  0.0  0.0 118056  1984 ? Sl   22:13 0:00 /usr/lib/openafs/davolserver
root 1416  0.0  0.0   2056   376 ? Ss   22:14 0:00 /sbin/afsd -afsdb -fakestat
```

Además de localmente a través de los procesos que se ejecutan en dklab1, podemos preguntar a "bosserver" sobre el estado de los servicios que supervisa, de una forma remota así.. Como siempre utilizamos su cliente "bos". De forma parecida podemos inspeccionar los logs localmente en el sistema de ficheros o remotamente usando a "bos".

```
bos status localhost -long


ls /var/log/openafs/
bos getlog -server dklab1.casafx.dyndns.org. -file BosLog
```

Antes de continuar por fin con la creación de volúmenes y su montaje, haremos una última mención a los ficheros CellServDB. Antes de ejecutar el script afs-newcell hemos registrado manualmente la IP donde volserver y ptserver son accesibles a los clientes. El script ha utilizado esa información para crear el CellServDB de servidor para las

comunicaciones fileserver-db, podemos comprobarlo inspeccionando el fichero o utilizando a bos:

```
bos listhosts -server dklab1.casafx.dyndns.org. -localauth

cat /etc/openafs/server/CellServDB
```

```
>casafx.dyndns.org      # Celda para casafx.dyndns.org
10.168.1.1              # dklab1.casafx.dyndns.org.
```

...la salida debiera ser el FQDN de la máquina, dklab1.casafx.dyndns.org., cualquier otro nombre o abreviación (por ejemplo "dklab1" sin más) debiera ser eliminado:

```
bos removehost -server dklab1.casafx.dyndns.org. -host "dklab1" -localauth

# El subcomando removehost tiene a su contrario addhost; se usari'a asi':

# bos addhost   -server dklab1.casafx.dyndns.org. \

#               -host "dklab1.casafx.dyndns.org." -localauth
```

1.2.7. Creación del volumen raiz de nuestra celda AFS

Preliminarmente, nos autenticamos como administrador consiguiento TGT KERBEROS de root/admin y, a partir de éste, un token AFS de root.admin:

```
kinit root/admin
```

```
Password for root/admin@CASAFX.DYNDNS.ORG:

r00tprincipa!
```

```
klist
```

Y a continuación ejecutamos la utilidad aklog; ésta es proveída por paquete openafs-krb5, utiliza un ticket TGT KERBEROS para obtener los tokens que presentar a OpenAFS como testigo de una autentificación exitosa anterior, y obtener finalmente acceso al sistema AFS:

```
aklog -d # -d para mayor verbosidad (debug)
```

```
Authenticating to cell casafx.dyndns.org (server dklab1.casafx.dyndns.org).
Trying to authenticate to user's realm CASAFX.DYNDNS.ORG.
Getting tickets: afs/casafx.dyndns.org@CASAFX.DYNDNS.ORG
Using Kerberos V5 ticket natively
About to resolve name root.admin to id in cell casafx.dyndns.org.
Id 1
Set username to AFS ID 1
Setting tokens. AFS ID 1 /  @ CASAFX.DYNDNS.ORG
```

```
tokens  # consulta las credenciales AFS
klist   # consulta la cache' de credenciales KERBEROS
```

El script afs-rootvol nos asistirá en la creación del volumen raiz de nuestra celda:

```
afs-rootvol
```

```
-"Do you meet these conditions? (y/n)"
y
-"What AFS Server should volumes be placed on?"
dklab1.casafx.dyndns.org.
-"What partition? [a]"
a
```

Enumeremos paso a paso qué lanza el script afs-rootvol:

- A continuación se crea en /vicepa el volumen que representa el raiz de la celda, root.cell. Ésto se traduce en la creación de un fichero de nombre V0536870915.vol o cualquier otro número, con un formato interno particular.

```
# No ejecu'tese, estamos relatando las acciones efectuadas por afs-rootvol
vos create dklab1.casafx.dyndns.org. a root.cell -localauth
```

```
Volume 536870915 created on partition /vicepa of dklab1.casafx.dyndns.org.
```

- Se establecen los permisos por defecto para las entidades del primer nivel: cualquiera puede leer los ficheros y listar los directorios.

```
fs setacl /afs system:anyuser rl
```

- Se crea el punto de montaje RO (read only) para el recién creado volumen raiz de la celda:

```
fs setacl /afs system:anyuser rl
```

- Se establece de nuevo que las entidades de su primer nivel puedan leerse (ficheros) y listarse (directorios) por cualquiera.

```
fs setacl /afs/casafx.dyndns.org system:anyuser rl
```

- Se crea un segundo punto de montaje para el raiz y el raiz de la celda, esta vez son RW.

```
fs mkmount /afs/.casafx.dyndns.org root.cell -cell casafx.dyndns.org -rw
fs mkmount /afs/.root.afs root.afs -rw
```

- A continuación se crea un nuevo volumen para los usuarios de la celda, se monta RO y se configura los permisos de lectura de ficheros y listado directorios, a cualquiera.

```
vos create dklab1.casafx.dyndns.org. a user -localauth
```

```
Volume 536870918 created on partition /vicepa of dklab1.casafx.dyndns.org.
```

```
fs mkmount /afs/casafx.dyndns.org/user user
fs setacl /afs/casafx.dyndns.org/user system:anyuser rl
```

- A continuación se crea un nuevo volumen "services" para los servicios de la celda. Igualmente al anterior, a continuación se monta RO y se configuran los permisos de lectura de ficheros y listado de directorios a cualquiera.

```
vos create dklab1.casafx.dyndns.org. a service -localauth
```

```
Volume 536870921 created on partition /vicepa of dklab1.casafx.dyndns.org.
```

```
fs mkmount /afs/casafx.dyndns.org/service service
fs setacl /afs/casafx.dyndns.org/service system:anyuser rl
```

- Se crean enlaces simbólicos de abreviaciones del nombre de nuestra celda a sus puntos de montaje RW y RO:

```
ln -s casafx.dyndns.org /afs/casafx    # ro
ln -s .casafx.dyndns.org /afs/.casafx  # rw
```

- Declaramos réplicas, "site replication" en terminología afs, de carácter RO tanto del volumen raiz afs y raiz de nuestra celda; al mismo tiempo que son RO, su sincronización con el volumen RW del que es réplica se hace de una forma asíncrona y manual, a través de los comandos vos release; por tanto tras declarar las réplicas con vos addsite, son creadas como si de una primera actualización se tratase con vos release:

```
vos addsite dklab1.casafx.dyndns.org. a root.afs  -localauth
```

```
Added replication site dklab1.casafx.dyndns.org. /vicepa for volume root.afs
```

```
vos addsite dklab1.casafx.dyndns.org. a root.cell -localauth
```

```
Added replication site dklab1.casafx.dyndns.org. /vicepa for volume root.cell
```

```
vos release root.afs -localauth
```

```
Released volume root.afs successfully
```

```
vos release root.cell -localauth
```

```
Released volume root.cell successfully
```

- El resto de acciones del script ya no involucran crear más volúmenes sino simplemente montar los volúmenes raiz correspondientes a celdas remotas (universidades, centros de investigación...) listadas en /etc/openafs/CellServDB. Puede comprobarse con "fs listcells".

Finalmente, repasemos los volúmenes creados hasta este momento:

```
vos listvol dklab1 -localauth
```

- root.afs, se monta en /afs, constituye pues el "volumen raiz de AFS" y por tanto no se espera que se creen archivos y directorios comunes en él, sino más bien que sirva de "soporte" para colgar la celda propia y ajenas; también se creó una réplica RO root.afs.readonly.
- root.cell, se monta en /afs/casafx.dyndns.org, y por tanto es el "volumen raiz de su celda"; se espera crear en él ficheros y directorios pero sobre todo nuevos puntos de montaje para volúmenes relacionados con usuarios, servicios, software... de nuestra celda. También se creó una réplica RO root.cell.readonly.
- user, se monta en /afs/casafx.dyndns.org/user, es un volumen cualquiera de nuestra celda pensado como soporte para colgar los puntos de montaje de los usuarios ("home").
- service, se monta en /afs/casafx.dyndns.org/service, es un volumen cualquiera de nuestra celda pensado como soporte para colgar los puntos de montaje del espacio reservado a datos de servicios (por ejemplo, correo).

```
unlog; kdestroy
```

1.2.8. Invocación de clientes OpenAFS tras creación de celda y volumen raiz

Al instalar openafs-client configuramos (respondiendo a las preguntas de debconf) que no se arrancase automáticamente al componente "afsd" (la parte del cliente en espacio de usuario) porque no estaba creada la celda etc. Podemos revertir[10] esa situación ahora con "dpkg-reconfigure openafs-client" (aceptar los valores por omisión, los que declaramos la última vez, excepto el que nos ocupa). O, no interactivamente:

```
(echo "set openafs-client/run-client true" | debconf-communicate) && \
 sed -i -e 's/AFS_CLIENT=false/AFS_CLIENT=true/g' \
 /etc/openafs/afs.conf.client
```

[10] En nuestro testbed con Qemu, dados los problemas que se comentaron con la recarga del módulo openafs.ko, recomendamos evitar hacer ésto y, en su lugar, levantar el cliente manualmente con "invoke-rc.d openafs-client start" cuando tengamos la certeza de que todos los demás servicios (especialmente OpenLDAP y MIT Kerberos) están cargados y operativos.

1.2.9. Inspección de nuestra celda AFS

Una primera vista de árbol bajo /afs/casafx.dyndns.org:

```
tree /afs/casafx.dyndns.org
```

```
/afs/casafx.dyndns.org
|
+- service
‘- user
```

Veamos de qué volúmenes son puntos de montaje esos directorios:

```
fs lsmount /afs/.root.afs
fs lsmount /afs/casafx.dyndns.org
```

```
’/afs/casafx.dyndns.org’ is a mount point for volume
’#casafx.dyndns.org:root.cell’
```

```
fs lsmount /afs/casafx.dyndns.org/user
```

```
’/afs/casafx.dyndns.org/user’ is a mount point for volume ’#user’
```

```
fs lsmount /afs/casafx.dyndns.org/service
```

```
’/afs/casafx.dyndns.org/service’ is a mount point for volume ’#service’
```

```
fs lsmount /afs/casafx
```

```
fs lsmount /afs/.casafx.dyndns.org
```

```
’/afs/.casafx.dyndns.org’ is a mount point for volume
’%casafx.dyndns.org:root.cell’
```

... donde el símbolo ’%’sustituye a ’#’para indicar que es una copia R/W; en todos los casos anteriores por ser montado el volumen sin el flag -rw o por estar montando una réplica (por definición RO) del volumen, en aquellos puntos de montaje no es posible

hacer cambios.

```
fs listquota /afs/casafx.dyndns.org
```

```
fs listacl /afs/casafx.dyndns.org/user
```

```
Access list for /afs/casafx.dyndns.org/user is
Normal rights:
 system:administrators rlidwka
 system:anyuser rl -> e'sta ACL es la que se aplica en ausencia de Token AFS
```

1.2.10. Creación de identidades AFS para usuarios comunes, sus volúmenes y puntos de montaje

Vamos a crear los recursos AFS necesarios para que el usuario umea tenga su directorio "home" como un volumen montado en el espacio de nombres de AFS.

El usuario umea existe como un nodo hoja en la base de datos LDAP dc=casafx,dc=dyndns,dc=org con una serie de atributos que acompañan. Además para la entidad umea existen unos recursos relacionados con KERBEROS también en ldap (aunque, por decisión nuestra al crear el realm, no están en atributos de umea sino bajo un organizationalUnit aparte). En cualquier caso, LDAP era el gran cajón de metadatos, los cuales son utilizados por los distintos servicios pero, en el caso de OpenAFS, éste no tiene soporte para utilizar a OpenLDAP como almacén sino que implementa sus propias bases de datos y, por tanto, debemos hacer que lo que se declaró en una sea consistente con lo que se declara para la otra. Veamos entonces qué valores tienen los atributos de la entidad uid=umea,ou=users,dc=casafx,dc=dyndns,dc=org en LDAP:

Preliminarmente, si no lo tenemos ya, obtenemos ticket para identificarnos ante los servicios de OpenAFS:

```
kinit root/admin; aklog
```

```
Password for root/admin@CASAFX.DYNDNS.ORG:
r00tprincipa!
```

```
slapcat -s uid=umea,ou=users,ou=accounts,dc=casafx,dc=dyndns,dc=org
```

```
dn: uid=umea,ou=users,ou=accounts,dc=casafx,dc=dyndns,dc=org
uid: umea
uidNumber: 31000
gidNumber: 31000
homeDirectory: /afs/casafx.dyndns.org/user/u/um/umea
... otros, irrelevantes en este momento ...
loginShell: /bin/bash
...
```

A la vista de ésto, crearemos en las db del Protection Server de OpenAFS a una entidad llamada umea (para ser mapeable desde su principal "umea"), indexada como 31000.

```
pts createuser umea -id 31000
```

...podemos inspeccionar a varios niveles que efectivamente se ha creado:

```
# Preguntando al ptserver:
pts examine umea
# Abriendo directamente el fichero de base de datos:
pt_util -user -group -members  -prdb /var/lib/openafs/db/prdb.DB0 \
        -datafile /dev/stdout
```

El tercer recurso en AFS es, claro, el volumen que montaremos luego. Para umea crearemos un volumen en el volserver de dklab1, partición "a" (/vicepa), de quota 15MB[11]

```
vos create dklab1.casafx.dyndns.org. a user.umea -maxquota 15000
ls /vicepa
vos listvol dklab1.casafx.dyndns.org. -localauth
vos examine user.umea
```

Como indicaba el valor de homeDirectory en LDAP, montaremos el volumen recién creado en /afs/casafx.dyndns.org/user/u/um/umea: es convención en afs crear los home de los usuarios colgando de /afs/<cell>/user y siguiendo una política de dos niveles (anticipando la posibilidad de tener miles de usuarios etc). Por tanto tenemos que crear los subdirectorios u/um/umea bajo el directorio /afs/casfx.dyndns.org/user.

[11]Modificable con fs setquota, véase http://docs.openafs.org/AdminGuide/#HDRWQ234.html

Como vimos con "fs lsmount /afs/casafx.dyndns.org/user", en ese directorio está montado el volumen #user que creó afs-rootvol, y que, como también vimos con "fs listacl /afs/casafx.dyndns.org/user", sólo permitía escribir a usuarios de system:administrators. Aquí vuelve a entrar en juego la pregunta que nos hizo afs-newcell: la entidad root.admin que dimos allí está incluída[12] en system:administrators, y es con la que hemos conseguido las credenciales (usando kinit y aklog). Por tanto, ya estamos autorizados para crear los subdirectorios:

```
cd /afs/casafx.dyndns.org/user

mkdir -p u/um

fs mkmount u/um/umea user.umea -rw
```

- Algunas anotaciones sobre este paso:
  - Para crear directorios no hace falta una utilería especial de OpenAFS, usamos la utilidad del SO, mkdir en linux etc.
  - A mkdir se le ha pasado u/um, no u/um/umea, porque ./umea será punto de montaje de un volumen y éstos no pueden preexistir a la llamada de fs mkmount.
  - Respecto a "fs mkmount", le hemos pasado el flag -rw para que no sea considerado read-only y permita a identidades debidamente autorizadas la escritura.

Aún tenemos que autorizar a la identidad umea para ser propietario y tener todo tipo de pemisos; las demás identidades tendrán denegado el acceso[13]:

```
fs listacl u/um/umea

fs setacl  u/um/umea umea all

fs setacl  u/um/umea system:anyuser ""

fs setacl  u/um/umea system:authuser ""
```

Podemos inspeccionar el resultado: acl, punto de montaje, volumen asociado, localización de éste:

```
fs listacl u/um/umea

fs lsmount u/um/umea

fs lv u/um/umea

fs whereis u/um/umea
```

[12] http://docs.openafs.org/QuickStartUnix/#HDRWQ53a.html

[13] Más adelante se explicará lo básico sobre ACL y grupos de usuarios en AFS

La idea de crear un volumen para cada usuario, permite que si el usuario que utiliza la cuenta umea cambiase de, por ejemplo, ciudad, podríamos migrar a otro servidor fileserver de la celda pero más cercano a esa ciudad su home. Esa migración se realizaría, por cierto, sin afectar al uso del volumen tal que mientras dure la migración los datos aparezcan com disponibles en ambas (y todas) las localizaciones.

Prueba de uso de AFS con umea

Por último, hagamos una prueba de uso. Nos identificaremos en MIT Kerberos como umea, conseguiremos un token AFS para umea, nos desplazaremos al punto de montaje recién creado e intentaremos escribir nuestro primer fichero:

```
unlog; kdestroy # nos despojamos de credenciales preexistentes (root/admin).


kinit umea; aklog -d
```

```
tokens          # en este punto ya estamos autorizados, como umea, en AFS
cd u/um/umea    # nos desplazamos al punto de montaje del volumen de umea
echo RW > test  # prueba de escritura, creamos fichero test con contenido RW
cat test        # prueba de lectura sobre el fichero recie'n creado, devuelve:
                #  RW
fs listacl test # listamos las ACL AFS sobre el fichero/directorio-padre
ls -l test      # listamos las ACL del fichero como recurso POSIX

# Recordemos que el fichero anterior es, a bajo nivel, parte del contenido
# de un fichero, /vicepa/VXXXXXXXXX.vol que representa para el volserver
# un volumen AFS. Asi', si volcamos en bruto ese fichero, podremos
# detectar en ASCII que existe ahora una referencia al fichero "test":
vos dump -id user.umea -server dklab1 -partition /vicepa -localauth \
    | grep -a test

reset
unlog; kdestroy # Nos desprendemos de la identidad umea:
```

```
ls              # comprobamos que, sin su credencial, no podemos leer ya


cd ~
```

1.3. DKLAB2

1.3.1. Instalación del soporte en el kernel Linux

module-assistant

Podemos copiar los módulos que compilamos en dklab1 si:

- La arquitectura del procesador en dklab2 es igual a la de dklab1.
- O si la de dklab2 es compatible hacia atrás con la de dklab1.
- O si la de dklab1 es compatible hacia atrás con la de dklab2 y no se usaron flags especiales en el compilador que utilicen las incompatibilidades (variable de entorno CFLAGS con opciones -march etc ...).

```
scp dklab1.casafx.dyndns.org:/usr/src/openafs*modules*deb /usr/src/
dpkg -i /usr/src/openafs*modules*deb
```

En otro caso repetiríamos los pasos que dimos en dklab1:

```
apt-get install openafs-modules-source \
                module-assistant linux-headers-‘uname -r‘


m-a prepare
m-a a-i -t -l ‘uname -r‘ openafs
modprobe -v openafs
```

DKMS

Si queremos las ventajas de DKMS, deberíamos de repetir los pasos que hiciéramos en dklab1:

```
apt-get install build-essential dkms linux-headers-‘uname -r‘
apt-get install openafs-modules-dkms
```

1.3.2. Instalación de OpenAFS en dklab2; justificación

Puesto que en dklab1 ya tenemos un servidor fileserver, unos gestores de bases de datos vlserver/ptserver y un cliente AFS, nos preguntamos qué función tendría volver a instalar aquellos en dklab2. Ciertamente los desarrolladores de debian contemplan todas las posibilidades y hacen depender unos paquetes de otros (el único paquete que podemos instalar independiente a los demás es el del cliente) pero, sin embargo, ésto no impide que nos planteemos, decíamos, qué papel principal tiene cada uno; veamos:

- Preliminarmente, el paquete openafs-krb5 nos permite utilizar el esquema de auth KERBEROS v5. Para las labores de cliente es imprescindible, para el resto de acciones podríamos evitarlo y usar el flag -localauth de bos etc, pero ésto impone la restricción de tener que estar logeado como root en dklab2; la solución más flexible es instalar openafs-krb5 acompañando a cualquiera de los siguientes paquetes.
- Instalaríamos openafs-dbserver si queremos proporcionar redundancia de las bases de datos AFS que ya funcionan en dklab1. Recordemos que existían dos bases de datos (principalmente): la de localización de volúmenes -vlserver-, y las de estructuras de seguridad como usuarios, grupos o ACL de AFS -ptserver-. Openafs implementa el protocolo ubik para que todos los servidores de bases de datos de una celda (dklab1 y dklab2 en nuestro caso) permanezcan sincronizados entre sí, sin más labor que registrar la existencia de los unos en los otros por parte del administrador de sistemas[14].
    - Anotación respecto de la dependencia con DNS: puesto que en un sistema distribuído éstos son los servicios a los que los clientes consultan para conocer la localización y uso final de la información, son necesarios registros SRV/AFSDB para dklab2 en el resolvedor DNS (ya se hizo cuando configuramos named).
    - Anotación respecto de la dependencia con NTP: como ocurrió con OpenLDAP, el proceso de sincronización Multi-Master requiere que los relojes de los sistemas que intervienen estén sincronizados: efectivamente, ésto ya se hizo, cuando configuramos ntpd. AFS utiliza en nuestro despliegue una tercera base de datos fundamental: la de autentificación, los principal de KERBEROS; esta base de datos también dispone de redundancia puesto que configuramos kadmind con storage-backend LDAP, y tenemos una red de servidores slapd con sincronización Multi-Master.

[14] Ésto no exime de hacer backups periódicos con tar en frío, dada la importancia de estas bases de datos,
http://docs.openafs.org/AdminGuide/#HDRWQ101.html#HDRWQ108

    - Un servidor AFS sólo gestor de las bases de datos es el único caso que no requiere instalar el módulo openafs.ko en el kernel.

- Instalaríamos openafs-fileserver si, independientemente de planear tener o no un servidor de bases de datos vlserver/ptserver en dklab2, queremos tener volúmenes AFS en una localización distinta a dklab1: bien porque queremos replicar copias read-only para aumentar la disponibilidad, bien porque queremos distribuir los volúmenes según la cercanía media que experimentan los usuarios respecto de dklab2 y dklab1. Ya se comentó que si nuestra organización tiene dos sedes separadas geográficamente, A y B, el volumen para los usuarios en sede A lo crearemos en el servidor geográficamente más cercano a sede A, por ejemplo dklab1; y si en algún momento un usuario de sede A se marcha durante un tiempo a sede B, moveremos su volumen al nuevo servidor geográficamente más cercano, por ejemplo dklab2.

- Instalaríamos openafs-client si necesitamos acceder y administrar el sistema de archivos AFS también desde dklab2.

Supongamos por ejemplo que en este momento nos interesa proveer de redundancia a las bases de datos, nos interesa la distribución de nuestros volúmenes, y no tanto acceder como cliente. En dklab2, por tanto:

```
apt-get install --no-install-recommends openafs-fileserver \
                                        openafs-dbserver openafs-krb5
```

...sin embargo, por dependencias en Debian, se instala también el cliente, así que se nos pregunta:

```
-"AFS cell this workstation belongs to:"
casafx.dyndns.org
-"Size of AFS cache in kB:"
50000
-"Run Openafs client now and at boot?"
No
-"Look up AFS cells in DNS?"
Yes            (usara' SRV y AFSDB)
-"Encrypt authenticated traffic with AFS fileserver?"
Yes
-"Dynamically generate the contents of /afs?"
No
-"Use fakestat to avoid hangs when listing /afs?"
Yes
```

Debconf pregunta, respecto del fileserver, el nombre de la celda:

```
-"Cell this server serves files for (fileserver):"
casafx.dyndns.org
```

...ésto hace que se edite /etc/openafs/server/ThisCell como si hiciésemos manualmente:

```
bos setcellname -server dklab2.casafx.dyndns.org. \
                -name casafx.dyndns.org -localauth
```

En este momento, bosserver debería estar corriendo en dklab2:

```
ps aux | (sleep 1; egrep \
"(bosserver\
|fileserver\
|volserver\
|vlserver\
|ptserver\
|salvage\
|buserver\
|upclient\
|upserver\
|afsd)")
```

1.3.3. Administración de las IP de los servidores y de sus entradas en la VLDB

Bajo las mismas premisas que se expusieron en dklab1:

```
printf "192.168.1.20\n10.0.2.15\n" > /var/lib/openafs/local/NetRestrict
```

1.3.4. Configuración de la autenticación KERBEROS previamente a las tareas de creación de la partición

Debemos tener en cuenta que:

- El principal es de la forma afs/<cellname>@<realm>, luego no es específico de un host particular. Ésto hace que el principal creado en dklab1 es usable en dklab2.
- OpenAFS no usa directamente un keytab Kerberos, hay que transformarlo en su contrapartida AFS utilizando asetkey (local):

Por tanto copiamos el keytab creado en dklab1 y lo registramos con asetkey [15]:

[15] No resultará hacer directamente: scp dklab1:/etc/openafs/server/KeyFile /etc/openafs/server/

```
scp dklab1.casafx.dyndns.org:/etc/keytab.d/openafs.keytab /etc/keytab.d/
asetkey add 2 /etc/keytab.d/openafs.keytab \
              afs/casafx.dyndns.org@CASAFX.DYNDNS.ORG
```

```
kadmin.local -q "getprinc -terse afs/casafx.dyndns.org"
cat /etc/openafs/server/KeyFile
bos listkeys dklab2.casafx.dyndns.org. -localauth
```

1.3.5. Creación de la partición vicepa

Simularemos una partición a través de un fichero, pero en un despliegue real debiera reservarse una partición real.

```
dd if=/dev/zero of=/var/local/vicepa bs=1024 count=76800 #75MB
mknod /dev/loop255 b 7 255
losetup /dev/loop255 #fallara'
losetup /dev/loop255 /var/local/vicepa
losetup /dev/loop255
mkfs -t ext3 -m 1 -v /dev/loop255
mkdir /vicepa
mount -t ext3 /dev/loop255 /vicepa
ls /vicepa
```

Si "ls" devolvió algo, entonces el fichero está montado, fue todo bien y podemos hacer los cambios permanentes entre arranques a través de /lib/udev/devices/ y /etc/fstab:

```
umount /vicepa
losetup -d /dev/loop255

cd /lib/udev/devices/ && mknod loop255 b 7 255 && cd -
vim /etc/fstab  # Puede interesar an~adir noatime pues penaliza
                # mucho el desempen~o.
```

```
...
####fx:
/var/local/vicepa  /vicepa  ext3  defaults,loop=/dev/loop255,rw,auto  0  0
####endfx
```

```
mount /vicepa
ls /vicepa
```

1.3.6. Despliegue de componentes AFS en dklab2; registro de los componentes de bases de datos en el resto de nuestra celda AFS

No hay un script para ello; directamente invocamos a "bos create" etc.
En la instalación ya se editó /etc/openafs/server/ThisCell, equivalente a:

```
bos setcellname -server dklab2.casafx.dyndns.org. \
                -name casafx.dyndns.org -localauth
```

Podemos añadir ya también al usuario root/admin a /etc/openafs/server/UserList en dklab2 para que pueda usar comandos privilegiados de bos, vos, backup. Nótese que traducimos el principal root/admin a root.admin, porque el '.'se utiliza en nombres especiales administrativos[16]:

[16] http://docs.openafs.org/AdminGuide/#HDRWQ502.html

```
bos adduser dklab2.casafx.dyndns.org. root.admin -localauth
cat /etc/openafs/server/UserList
```

Como "OpenAFS Simple Fileserver"

Declaramos que dklab2 tenga un rol de "OpenAFS Simple Fileserver". Para ello le pedimos al monitorizador bosserver que registre y lance el triplete formado por un fileserver, un volserver y un salvager[17]:

```
bos create dklab2.casafx.dyndns.org. dafs \
 -type dafs -cmd '/usr/lib/openafs/dafileserver -p 23 -busyat 600 -rxpck 400
                                                -s 1200 -l 1200 -cb 65535 -b 240
                                                -vc 1200' \
            -cmd /usr/lib/openafs/davolserver \
            -cmd /usr/lib/openafs/salvageserver \
            -cmd /usr/lib/openafs/dasalvager -localauth
```

Como "OpenAFS Database Server"

- Anotación sobre DNS: se aconseja un mapeo biyectivo de nombres de host y direcciones IP en un servidor de db OpenASF. De esta forma, la salida de gethostbyname() devolverá un único resultado que usarán consistentemente todos los OpenAFS de la celda[18]:

  - "It is best to maintain a one-to-one mapping between hostnames and IP addresses on a multihomed database server machine (the conventional configuration for any AFS machine). The BOS Server uses the gethostbyname( ) routine to obtain the IP address associated with the host name argument. If there is more than one address, the BOS Server records in the CellServDB entry the one that appears first in the list of addresses returned by the routine. The routine possibly returns addresses in a different order on different machines, which can create inconsistency".

Nosotros cumplimos con esta premisa, dada la política de nombes y direcciones en nuestra zona DNS. No obstante, para estar totalmente seguros de qué devuelve la función

[17] http://docs.openafs.org/QuickStartUnix/#HDRWQ99.html#HDRWQ100
[18] http://manpages.debian.net/cgi-bin/man.cgi?query=GETHOSTBYNAME

gethostbyname(), el siguiente código en C utiliza esa función y muestra por pantalla su resultado:

```
w3m -dump http://www.logix.cz/michal/devel/various/gethostbyname.c.xp > ghbn.c
cc ghbn.c -o /usr/local/bin/gethostbyname.elf
/usr/local/bin/gethostbyname.elf dklab1.casafx.dyndns.org.
/usr/local/bin/gethostbyname.elf dklab2.casafx.dyndns.org.
```

```
name: dklab1.casafx.dyndns.org
address: 10.168.1.1
#
name: dklab2.casafx.dyndns.org
address: 10.168.1.2
```

El código en C es éste:

```c
/* gethostbyname.c - Example of using gethostbyname(3) Martin Vidner */
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>

struct hostent *he;
struct in_addr a;
int main (int argc, char **argv) {
    if (argc != 2) {
        fprintf(stderr, "usage: %s hostname\n", argv[0]);
        return 1;
    }
    he = gethostbyname (argv[1]);
    if (he) {
        printf("name: %s\n", he->h_name);
        while (*he->h_aliases) printf("alias: %s\n", *he->h_aliases++);
        while (*he->h_addr_list) {
            bcopy(*he->h_addr_list++, (char *) &a, sizeof(a));
            printf("address: %s\n", inet_ntoa(a));
        }
    }
    else herror(argv[0]);
    return 0;
}
```

Registro en la celda de los nuevos servidores DB ofrecidos en dklab2 Todos las máquinas de nuestra celda donde hemos desplegado OpenAFS, necesitan conocer todos los servidores de db que existen operando en la celda.

Primero registramos dklab2 en dklab1; éste último listará entonces el FQDN de dklab2 en su /etc/openafs/server/CellServDB. Podemos hacer remotamente tanto el registro

como la comprobación gracias a bos si nos autorizamos debidamente (en este momento, lo más sencillo es indicar -localauth para hacer que se utilice el /etc/openafs/server/KeyFile, que aunque no exactamente igual al presente en dklab1, es válido para crear tokens válidos allí o en toda la celda).

```
# Registramos en dklab1 la existencia de servidores db en dklab2
bos addhost -server dklab1.casafx.dyndns.org. \
            -host dklab2.casafx.dyndns.org. -localauth
```

Los servidores db registrados en dklab1 hasta el momento, deberían ser él mismo (registrado por los comandos que lanzaba el script afs-newcell que ejecutamos en su momento) y dklab2 (recién incluído ahora):

```
bos listhosts -server dklab1.casafx.dyndns.org. -localauth
```

```
 Cell name is casafx.dyndns.org
   Host 1 is dklab1.casafx.dyndns.org.
   Host 2 is dklab2.casafx.dyndns.org.
```

Después en el propio dklab2, registramos a sí mismo y a dklab1:

```
bos addhost -server dklab2.casafx.dyndns.org. \
            -host dklab2.casafx.dyndns.org. -localauth
bos addhost -server dklab2.casafx.dyndns.org. \
            -host dklab1.casafx.dyndns.org. -localauth
```

Así pues, si inspeccionamos el fichero /etc/openafs/server/CellServDB:

```
cat /etc/openafs/server/CellServDB
```

```
>casafx.dyndns.org      #Cell name
10.168.1.2    #dklab2.casafx.dyndns.org.
10.168.1.1    #dklab1.casafx.dyndns.org.
```

- Si erramos al registrar el FQDN de dklab2 (o tenemos algo adicional que no lo es u otro problema) podríamos eliminar esa entrada utilizando

  `bos removehost -server <host donde desregistrar> -host <FQDN>`

y vuélvase a intentar.

- Nos ha ocurrido que apareciesen las dos IP anteriores primero entre corchetes y luego sin ellos. IP's entre corchetes no se utilizan en la sincronización de ubik, luego la situación de la celda era inconsistente y daría problemas más tarde; para borrar las IP con corchetes no parecía funcionar el comando anterior "bos removehost", de forma que paramos el servidor

  ```
  invoke-rc.d openafs-fileserver stop
  ```

  , borramos las IP con corchetes editando el CellServDB

  ```
  vim /etc/openafs/server/CellServDB
  ```

  y volvimos a reiniciarlo

  ```
  invoke-rc.d openafs-fileserver start
  ```

  .

Registro en dklab2 de los procesos db que su bosserver debe lanzar Ahora que los servidores db desplegados en dklab2 son conocidos, podemos levantarlos[19]. Como sabemos OpenAFS disponde de su "basic overseer", al que pedimos que registre, lance y monitorice un proceso vlserver y ptserver:

[19]

- En el caso de que la IP del nuevo servidor sea la más baja, debiéramos incluso avisar - véase siguiente apartado - a los clientes antes de desplegar los nuevos componentes.

```
bos create -server dklab2.casafx.dyndns.org. \
        -instance ptserver -type simple -cmd /usr/lib/openafs/ptserver \
        -localauth


bos create -server dklab2.casafx.dyndns.org. \
        -instance vlserver -type simple -cmd /usr/lib/openafs/vlserver \
        -localauth


# Por seguir el ejemplo que hicimos en dklab1, podri'amos de nuevo cargar a
# buserver (db copias de respaldo a cinta) pero no es necesario en absoluto.
bos create -server dklab2.casafx.dyndns.org. \
        -instance buserver -type simple -cmd /usr/lib/openafs/buserver \
        -localauth
```

Forzamos a que cada grupo de componentes de cada tipo (vlserver en dklab1 y dklab2, ptserver en dklab1 y dklab2, etc) negocien cómo actuará el protocolo de sincronización de bases de datos OpenAFS, "ubik":

```
bos restart -server dklab1.casafx.dyndns.org. \
            -instance ptserver vlserver buserver -localauth
bos restart -server dklab2.casafx.dyndns.org. \
            -instance ptserver vlserver buserver -localauth
```

Esperamos un tiempo prudencial:

```
sleep 5m
```

Todos los componentes deberían estar cargados (en otro caso ejecútese "invoke-rc.d openafs-fileserver restart"):

```
bos status -server dklab2.casafx.dyndns.org. -long -noauth
bos status -server dklab1.casafx.dyndns.org. -long -noauth
```

Podemos inspeccionar cómo ubik está decidiendo quién es master en cada momento y

otros parámetros con:

```
                                                #port it queries
                                                #----
udebug dklab1.casafx.dyndns.org. ptserver -long #7002
udebug dklab1.casafx.dyndns.org. vlserver -long #7003
udebug dklab1.casafx.dyndns.org. buserver -long #7021

udebug dklab2.casafx.dyndns.org. ptserver -long #7002
udebug dklab2.casafx.dyndns.org. vlserver -long #7003
udebug dklab2.casafx.dyndns.org. buserver -long #7021
```

Otras comprobaciones como cliente AFS se podrán utilizar (requieren Token AFS previo):

```
fs checkservers -cell casafx.dyndns.org
fs getserverprefs; fs getserverprefs -vlserver
```

Los clientes deben ser avisados de los nuevos componentes de db OpenAFS

- Anotación: en el caso de que la IP del nuevo servidor sea la más baja, debiéramos incluso avisar a los clientes antes de desplegar los nuevos componentes[20], pero no era nuestro caso.

Existen varios escenarios:

- Si el cliente en espacio de usuario (afsd) se ejecutó con el flag -afsdb y teníamos entonces un DNS RR AFSDB/SRV para dklab2, no hágase nada. Es nuestro caso.
- En otro caso, se debería modificar adecuadamente el fichero CellServDB de cliente, situado en /etc/openafs/CellServDB (ya vimos antes de ejecutar el script afs-newcell en dklab1 que el formato era trivial, véase de todas formas "man CellServDB"). La modificación consistiría en añadir el FQDN de dklab2. Entonces reiniciaríamos la máquina, o bien también añadiríamos a dklab2 directamente en memoria (requeriría Token AFS previo):

[20] Consúltese buscando "lowest IP address" enhttp://docs.openafs.org/QuickStartUnix/#HDRWQ114.html

```
fs newcell -name casafx.dyndns.org \
           -servers dklab1.casafx.dyndns.org. dklab2.casafx.dyndns.org.
```

- Un tercer escenario consiste en desplegar ciertos servicios adicionales (véase upserver/upclient) que constituyen un sistema de sincronización de archivos de configuración de OpenAFS.

1.4. Más posibilidades del lado del cliente

Los siguientes ejemplos suponen que estamos en dklab1, pero podríamos realizarlos en dklab2 iniciando antes el cliente AFS. Nótese el "force-start" para superar la restricción de carga automática que le impusimos al servicio.

```
invoke-rc.d openafs-client force-start
```

1.4.1. Caché

Mientras del lado del servidor aumentamos la disponibilidad replicando volúmenes y añadiendo servidores, del lado del usuario lo hacemos configurando el comportamiento de su caché.

En primer lugar, hay que decidir si ésta residirá en disco (default) o en RAM; sin embargo OpenAFS desaconseja ésto último por no funcionar igual de bien que el caché en disco y no producir un notable aumento de rendimiento. En /etc/openafs/cacheinfo teníamos configurado que la caché residiría en /var/cache/openafs.

Respecto al tamaño, podemos inspeccionar el valor de caché actual con "fs getcacheparms" (tras autenticarnos):

```
kinit root/admin; aklog
```

```
Password for root/admin@CASAFX.DYNDNS.ORG:
r00tprincipa!
```

```
fs getcacheparms
```

...y cambiarlo al vuelo con:

```
fs setcachesize <size in 1K byte blocks (0 => reset)>
```

...y hacer los cambios permanentes editando:

```
vim /etc/openafs/cacheinfo
```

Respecto a otros parámetros, simplemente diremos que el cache manager en espacio de usuario "/sbin/afsd", soporta varios flags para afinar más el comportamiento de la caché, y pueden declararse permanentemente en /etc/openafs/afs.conf[21].

Una vez que hemos configurado al cache manager, el uso del sistema de archivos bajo /afs será como cualquier otro, si bien es cierto que podemos sacar más partido de éste si conocemos algunos detalles de su funcionamiento, por ejemplo:

- Nótese, que los cambios se hacen en la caché y son subidos no cuando hay una escritura (no "write-on-commit") sino cuando se llama a close ("write-on-close"; también a fsync()). Ésto conviene tenerlo en cuenta mientras editamos un archivo: los cambios irán estando disponibles en la celda siempre que se llame a close(); ésto ocurre siempre que se cierra la aplicación, y normalmente (pero no siempre es así) cuando seleccionamos "Guardar/Save". Otro detalle de las modificaciones en afs es que openafs no contempla el concepto de "control de versiones", ni por tanto un sistema de resolución para colisiones: si varias personas editan el mismo fichero, la última copia recibida es la que se conserva; ésto nos dice que en openafs es buena costumbre tener un flujo de trabajo por el cual al decidir editar un archivo, pensamos 1) si el archivo necesitará una acl que impidan a otros editarlo también, 2) aplicar las acl si conviene, 3) editar el archivo. Un razonamiento similar se sigue para la modificación de directorios, etc[22].

- Otra nota a tener en cuenta es que si la caché se llena con contenido editado por nosotros y aún no se ha subido a los servidores, el cache manager no podrá deshacerse de nada y dará un error de I/O si pedimos algo nuevo. La aplicación nos hará llegar este error, aunque no nos dirá la causa.

- Por último, aunque es posible hacer un uso básico de /afs sin conocer su sistema de permisos, evidentemente nos conviene poder usar esta herramienta para asegurar nuestros documentos ante lecturas y, como hemos adelantado, ante escrituras inoportunas. O para predecir qué y qué no podremos hacer en determinados directorios. A continuación se expone una pequeña introducción a las ACL en openafs:

1.4.2. ACL's. Grupos AFS

ACL en AFS

Tipos de permisos definidos en AFS Las listas de control de acceso permiten o no a los usuarios o grupos de usuarios definidos en la celda (pts db), efectuar acciones sobre

[21] http://docs.openafs.org/AdminGuide/index.html#HDRWQ402.html
[22] http://docs.openafs.org/UserGuide/index.html#HDRWQ8.html

ficheros y directorios. Las ACL se aplican a nivel de directorio y las heredan todos sus nodos (en concreto, creados a partir de ese momento), por tanto no se puede especificar una ACL por fichero (si bien hay una excepción: usar parcialmente las acl posix, como se verá). Comparativamente a POSIX, se pueden aplicar más de 3 ACL (hasta 20), y cada ACL se refiere a más de 3 acciones, resultando en un sistema más granular. Las (7+) acciones en AFS son:

- Aplicables al directorio:
  - l permite list sus ficheros y subdirectorios (y, ojo, en AFS para acceder a un subdirectorio, uno tiene que tener permiso "l" en todos los directorios padre).
  - i insert nuevos ficheros en el directorio.
  - d delete ficheros y subdirectorios.
  - a administer las acl. Los miembros de system:administrators implícitamente tienen este permiso.
- Aplicables a sus ficheros:
  - r read contenido y stat de los ficheros del directorio
  - w write en el contenido de los ficheros y los bit de modo posix.
  - k lock files
- (Permisos Auxiliares: ABCDEFGH, programables si se precisa mayor granularidad).

POSIX ACL's De una forma incompleta, en el caso de ficheros, AFS también usa los permisos POSIX a través de los bits "r,w del propietario"; en el caso de directorios todos los bits posix son inefectivos. La consecuencia es que de esta forma existen acl (en sentido conjunto POSIX+AFS) aplicadas sobre ficheros que no coinciden con las que se aplican sobre el directorio del que son hoja. Por ejemplo, si en <directorio>se permite la escritura, "chmod u-w <directorio>/<file>" la impide sólo para ese fichero, etc[23].

Negative Rights Se refiere a máscaras. El administrador puede configurar una acl, y modificarla momentáneamente a continuación sin que el sistema, y por tanto tampoco el administrador, se olvide de la primera. Para ello lo que el administrador hace es, en lugar de cambiar la acl, le impone una máscara de bits, llamada "negative rights". El que estemos configurando la acl "normal" o la máscara depende de pasar o no la opción -negative a "fs setacl".

[23] http://docs.openafs.org/UserGuide/index.html#HDRWQ59.html

Administración de ACL's con "fs setacl/listacl" La interfaz general de ambos en pseudo-BNF sería:

```
fs setacl  -dir dir1 dir2 ... -acl ACL1 ACL2 ... [-negative]
fs listacl -dir dir1 dir2 ...
```

...donde -acl puede llevar como argumento estos alias:

- all (rlidwka)
- none ()
- read (rl)
- write (!a)

Por ejemplo:

```
fs listacl misubdir                     # comprueba si tenemos permiso "a"
fs setacl  misubdir umea rlw            # "rlw" para usuario umea
fs setacl  misubdir umea l    -negative # "rw"  para umea gracias a la ma'scara
fs setacl  misubdir umea none -negative # "rlw" de nuevo al quitar la ma'scara
```

Creación de grupos AFS para ACL's

Los grupos de usuarios facilitan la aplicación de ACL's. En afs los grupos se nombran con el formato prefijo:sufijo, donde prefijo es el usuario propietario y sufijo el nombre de grupo. Ser propietario implica tener ciertos permisos[24] sobre la administración del grupo, y no implica pertenecer a él automáticamente. A un grupo pueden pertenecer otros grupos, con hasta 5 niveles de profundidad hasta las identidades de usuario. También, se pueden asociar incluso direcciones IP ("pts createuser" lo admite) proporcionando la posibilidad de usar ACL's para rangos de máquinas, de IP's.

Para listar los usuarios y grupos definidos actualmente en la celda:

```
pts listentries -users
pts listentries -groups
```

[24] http://docs.openafs.org/UserGuide/index.html#HDRWQ74.html

Tareas de administración de grupos AFS Utilizamos subcomandos de la utilidad pts. Lo ilustraremos con ejemplos:

```
pts creategroup root/admin:migrupo   #grupo "migrupo" propietario root/admin
pts chown root/admin:migrupo umea    #cambio de propietario root/admin por umea


pts adduser umea -group umea:migrupo # an~adimos el user umea al grupo migrupo
pts membership umea:migrupo          # que' usuarios/grupos pertenecen a "migrupo"
pts membership umea                  # a que' grupos pertenece umea
pts removeuser umea -group umea:migrupo #quitamos user umea del grupo migrupo
pts delete umea:migrupo              # eliminamos grupo migrupo
pts cleanacl ./directorio            # eliminamos acl's en "directorio"
                                     # relacionadas con grupos no
                                     # existentes ya.
```

Grupos predefinidos "System Groups" Existen tres grupos de usuarios predefinidos en OpenAFS[25]:

- system:anyuser, contiene a todo aquel usuario que pueda acceder al árbol AFS sin estar autenticado. Por ejemplo, usuarios de otras celdas pero que tienen montada la nuestra bajo /afs/. Ejemplos específicos de uso:
  - Usuarios de otra celda: permiso r para system:anyuser les permite acceder al recurso.
  - Procesos de entrega: como un mail delivery system, printing... pueden necesitar permiso r|/l|/i para system:anyuser en sus directorios. Podemos subir un escalón y usar k5start para que al servicio se le apliquen las acl de system:authuser o propias.
- system:authuser, contiene a todo aquel usuario que haya conseguido autenticarse en AFS. Ni este grupo ni el anterior aparecen en la lista de grupos a los que pertenece el usuario, así que no aparecerían en la salida de "pts membership", pero para evitar verbosidad redundante.
- system:administrators, grupo para proveer extensos permisos en la celda a unos pocos usuarios. El administrador no necesita generalmente explicitar este grupo en sus ACL, porque sus miembros tienen siempre potestad para ejecutar acciones

[25] http://docs.openafs.org/UserGuide/index.html#HDRWQ50.html

propias del permiso "a" (administer), esté o no su grupo especificado en la ACL del recurso.

Sistema de cuotas de creación de grupos Los usuarios tienen un sistema de quotas de creación de grupos, consúltese:
http://docs.openafs.org/AdminGuide/#HDRWQ558.html

Ejemplos de configuración de ACL's usando grupos Supongamos que tenemos un directorios "misubdir", entonces:

```
fs listacl misubdir                    # comprueba si tenemos permiso "a".
fs setacl  misubdir umea:migrupo rlw   # "rlw" para usuarios del
                                       # grupo umea:migrupo.
fs setacl  misubdir umea:migrupo  l     -negative # "rw"  para umea:migrupo
                                                  # gracias a la ma'scara.
fs setacl  misubdir umea:migrupo none -negative  # "rlw" de nuevo al quitar
                                                  # la ma'scara.
```

1.5. Más posibilidades del lado del servidor. ¿Por qué AFS?

Aunque la documentación oficial[26] es más que suficiente, haremos aquí un breve recorrido sobre algunas de las posibilidades más interesantes que nos brinda OpenAFS. Hasta ahora conocemos que:

- Usa a KERBEROS, proporcionándonos un esquema SSO (Single Sign On) de autenticación.
- Nos permite mezclar ficheros locales y remotos en un único espacio de nombres, /afs
- Implementa un sistema de ACL que permite delegar la administración de diversas partes del árbol de directorios.

Estas cualidades, unidas a las que se exponen a continuación, justifican la elección de OpenAFS como sistema de ficheros en un despliegue en el que recursos y servicios se reparten en diversas máquinas de una red. En sucesivos capítulos, además, se particularizará cómo interviene OpenAFS en el despliegue de servicios (correo).

[26] http://docs.openafs.org/AdminGuide/index.html

- Algunas anotaciones:
  - Puede verse un listado informal de las fortalezas y debilidades de AFS y otros sistemas de ficheros en
    - http://wiki.linux-nfs.org/wiki/index.php/Comparison_of_NFS_vs._others.
  - también es interesante la presentación de la "Open Source Data Center Conference 2010"
    - http://www.netways.de/uploads/media/Fabrizio_Manfred_Use_Distributed_Filesy
  - La hoja de ruta con las funcionalides planeadas en
    - http://www.openafs.org/roadmap.html.
  - Ante problemas desconocidos, consúltese la sección de comprobaciones de volúmenes y bases de datos que detallamos más adelante en unas subsecciones de ese nombre.

1.5.1. Traslado de un volumen a otro servidor de la celda

Como se adelantó al hablar del posible papel de un segundo fileserver en la red, queremos distribuir los volúmenes según la cercanía geográfica que experimentan los usuarios respecto de dklab2 y dklab1. Por ejemplo, si nuestra organización tiene dos sedes separadas geográficamente, A y B, el volumen para los usuarios en sede A lo crearemos en el servidor geográficamente más cercano a sede A, por ejemplo dklab1; y si en algún momento un usuario de sede A se marcha durante un tiempo a sede B, moveremos su volumen al nuevo servidor geográficamente más cercano, por ejemplo dklab2[27].

Supongamos que queremos mover user.umea, el volumen tras el punto de montaje:

```
fs lsmount  /afs/casafx.dyndns.org/user/u/um/umea
```

Para conocer cuántos Kilobytes ocupa el volumen que vamos a mover:

```
vos examine user.umea | egrep '(user.umea|server)'
```

Para saber cuántos kiloBytes por usar tenemos en el destino, dklab2:

```
scout -server dklab2.casafx.dyndns.org #Interactivo, Control-C para salir
```

```
 # Columna "Disk attn" (en kiloBytes)
 ...        a:69723
```

[27] Existen intentos para efectuar este balance automáticamente según ciertos criterios como el tamaño del volumen, el uso (vos listvol <host>-extended), véase:
ftp://ftp.andrew.cmu.edu/pub/AFS-Tools/balance-1.1b.tar.gz

Si todo es correcto, utilizamos "vos move":

- Anotación: durante el proceso, "vos listvol dklab1" recalcará que el volumen está "busy", lo cual no afecta a que los clientes puedan seguir leyendo y escribiendo del volumen sin notar penalización o diferencia alguna (en situaciones de escasez de espacio puede ser mejor bloquear, sin embargo, usando -live).

```
vos move -id user.umea -verbose \
         -fromserver dklab1.casafx.dyndns.org. -frompartition /vicepa \
         -toserver   dklab2.casafx.dyndns.org. -topartition /vicepa
```

```
Starting transaction on source volume 536870932 ... done
...
Volume 536870932 moved from dklab1.casafx.dyndns.org. /vicepa \
                 to dklab2.casafx.dyndns.org. /vicepa
```

```
vos listvldb user.umea # comprueba si esta' actualizada la vldb
fs whereis /afs/casafx.dyndns.org/user/u/um/umea
```

... ésto puede tardar un tiempo en algunos casos, los vlserver tienen que sincronizar sus DB.

1.5.2. Creación de réplicas de sitio (copias RO) distribuídas por la celda

Justificaremos la creación de una réplica con un ejemplo. Imaginemos que el usuario umea, que trabaja en la sede B, se dedica principalmente a publicar documentación oficial de su organización en su espacio AFS.

Cuando tenemos volúmenes muy usados y muy estables (ejem: documentación, código ejecutable), son candidatos ideales a ser replicados en otros fileservers aumentando su disponibilidad. Las réplicas son siempre de sólo lectura y el sistema las nombra como el original pero añadiendo el sufijo ".readonly". La documentación de OpenAFS llama a su tecnología de replicación "sites"[28], y de ahí el nombre de los subcomandos addsite, remsite, etc. Veamos:

Primero registramos en la "Volumen Location DB" la réplica: dónde residirá, de qué volumen.

[28] http://docs.openafs.org/AdminGuide/#HDRWQ192.html

```
vos addsite -server dklab1.casafx.dyndns.org. -partition a \
            -id user.umea \
            -verbose
```

```
Added replication site dklab1.casafx.dyndns.org. /vicepa for volume user.umea
```

En este momento, si descubrimos haber cometido algún error con el nombre etc, todavía podemos simplemente desregistrar el sitio de la VL DB con:

```
vos remsite -server dklab1.casafx.dyndns.org. -partition a -id user.umea
# Los siguientes dos seri'an necesarios solamente
# en determinadas ocasiones, consu'ltese su p'agina man
#vos syncserv dklab a
#vos syncvldb dklab a
```

Pero si todo fue bien con addsite, el segundo paso es hacer efectiva la creación de la réplica. Es muy interesante saber que si ejecutamos este comando en futuras ocasiones, el efecto será el de sincronización de las réplicas read-only con el original read-write.

```
vos release user.umea -verbose
```

```
Released volume user.umea successfully
```

Podemos lanzar una comprobación con:

```
vos listvol dklab1.casafx.dyndns.org.
```

```
...
user.umea.readonly
```

Si por falta de espacio u otra razón queremos eliminar una réplica, utilizaríamos:

- Anotación: las réplicas no se pueden mover (vos move), pero equivalentemente se pueden eliminar con este "vos remove" y recrear en otro servidor con "vos addsite" y "vos release" como se ha visto.

- Anotación sobre la implementación de réplicas como "clones": Tanto con los volúmenes réplica como con los backup que se verán más tarde, OpenAFS no hace una copia literal del

volumen RW, sino que crea una estructura llamada clon[29] que, por poseer sólo el "vnode index" y otras técnicas, es más económico en espacio de disco. De hecho, por ser clones y no "copias", puede ser beneficioso crear una réplica local, en el mismo servidor donde está el volumen RW, y aprovechar el comportamiento del caché manager para crear rutas RO en las lecturas.

En dklab1 se crearon algunos volúmenes imprescindibles para nuestra célula: root.afs y root.cell. Efectivamente, ambos dan el soporte para el punto de montaje raiz de nuestra celda y puede ser conveniente hacer una réplica RO en todos los fileserver:

```
vos addsite -server dklab2.casafx.dyndns.org. -partition a \
            -id root.afs -verbose
vos release root.afs -verbose
vos listvol dklab2.casafx.dyndns.org.
```

```
root.afs.readonly                536870913 RO        214 K On-line
...
```

```
vos addsite -server dklab2.casafx.dyndns.org. -partition a -id root.cell
vos release root.cell
vos listvol dklab2.casafx.dyndns.org.
```

```
root.afs.readonly                536870913 RO        214 K On-line
root.cell.readonly               536870916 RO          4 K On-line
...
```

Comportamiento del cache manager y réplicas

El cliente AFS, cuando resuelve, a través de una consulta a algún vlserver, el volumen y el servidor donde se encuentra el contenido montado en una parte del árbol, tiene noticia tanto del volumen (<volumename>) como de sus réplicas (<volumename>.readonly) y backups (<volumename>.backup). Entonces tiene lugar una toma de decisión sobre cuál de ellos utilizará para lectura. Estas reglas de "travesía de puntos de montaje"[30] nos interesa conocerlas para entender con mayor precisión en qué situaciones nos interesa hacer réplicas y cómo montarlas. Veamos:

[29] http://docs.openafs.org/AdminGuide/#HDRWQ190.html
[30] http://docs.openafs.org/AdminGuide/#HDRWQ208.html#HDRWQ209

- Si el punto de montaje alude a una réplica o backup, el cliente no tiene problema en usarlos en lugar de buscar por sí mismo la versión RW.
- Si el punto de montaje alude al volumen base:
  - pero ese punto de montaje reside en un volumen RO, busca si el volumen tiene una réplica y la usa, en otro caso usa la RW. Si en /afs se monta el volumen raiz de AFS root.afs en modo RO (así se hizo), entonces esta regla mantiene al caché manager en rutas RO mientras cada sucesivo volumen tenga una réplica. Y así se hizo de hecho: al montar subsecuentemente el root.cell.readonly, se montó en /afs/casafx.dyndns.org. Mientras no se rompa la regla, el cliente está en una continua ruta RO. Por tanto, no es necesario en este caso montar las réplicas para que sean usadas.
  - pero si ese punto de montaje reside en un volumen montado con flag -rw, entonces se acude al volumen base, haya o no réplicas, y si no está disponible fallará. Si en los puntos de montaje subsiguientes se sigue esta regla, el cliente está en una continua ruta RW. Los puntos de montaje de celdas son una excepción, siempre se eligirán réplicas si las hay.

Consúltese "man fs_mkmount".

1.5.3. Montaje de otras celdas

- La razón de considerar este apartado como de servidor y no de cliente es porque en la documentación oficial se encuadra como administración de volúmnes.

Para montar volúmenes usamos, en general, "fs mkmount -dir <dir>-vol <vol>". El caso de montaje de celdas debe ser parecido, puesto que una celda es al fin y al cabo un volumen raiz registrado en un volserver, accesible en un fileserver, y que montamos en /afs/<cell>. Efectivamente es así[31], pero además tendremos específicamente en cuenta:

- Que el nombre del volumen de la celda remota debiera ser, por convenio, root.cell.
- Que hay que indicar la celda a la que pertenece ese root.cell con -cell <name>.
- Que /afs/<cell>tiene montada una réplica RO del root.afs (al menos en nuestro despliegue[32]), mientras que el volumen en sí está montado RW en /afs/.<cell>; por tanto hemos de hacer el montaje de la celda ahí para luego hacer un release a su réplica en /afs.
- Que las ACL de /afs/.<cell>permitirán o no hacer algo con ella.

[31] http://docs.openafs.org/AdminGuide/#HDRWQ208.html#HDRWQ213
[32] http://docs.openafs.org/QuickStartUnix/#HDRWQ91.html

En nuestro caso, el script afs-rootvol montó todas las celdas listadas en /etc/openafs/CellServDB[33] que no incluía, por ejemplo, la celda ihep.su (Instituto Ruso de Física de Altas Energías). Vamos a ejemplificar nuestro montaje de nueva celda con esta ihep.su. Entonces podríamos:

Previamente, debemos indagar si la celda ihep.su puede ser descubierta a través de DNS y sus registros SRV/AFSDB (pues afsd los usaría sin duda, porque se levanta con el flag -afsdb):

```
dig _afs3-vlserver._udp.ihep.su SRV +short
dig ihep.su AFSDB +short
```

... en este caso no existen tales registros en DNS, y hemos de conseguir su localización (en la documentación del sitio web etc) y registrarlos localmente en /etc/openafs/CellServDB.

```
# Registro de sus db servers local (a falta de registros SRV/AFSDB)
printf '>ihep.su          #IHEP\n' >> /etc/openafs/CellServDB
printf '194.190.165.195   #afssrv00.ihep.su\n' >> /etc/openafs/CellServDB
```

Finalmente ya podemos montar e intentar usar:

```
# Montamos su root.cell, es decir con: -vol root.cell -cell ihep.su
cd /afs/.root.afs/
fs mkmount -dir /afs/.root.afs/ihep.su -vol root.cell \
           -cell ihep.su # afssrv00.ihep.su
vos release root.afs -localauth -verbose

# Forzamos al cache manager (cliente AFS) a actualizar su informacio'n de vols
fs checkvolumes

# Y listamos su contenido, por fin:
ls -d /afs/ihep.su
```

- Anotación: efectivamente, el punto de montaje de nuestra celda, /afs/casafx.dyndns.org, sigue una estrategia parecida.

[33] http://docs.openafs.org/QuickStartUnix/#HDRWQ66.html

- En principio en ese directorio está montado el volumen root.cell y no es ya por tanto un directorio de la réplica RO del root.afs montada en su padre, /afs.
- Pero, de nuevo, en /afs/casafx.dyndns.org no está montado rw nuestro root.cell, tampoco está montado RO exactamente, sino que vuelve a ser una réplica y root.cell está montado RW en /afs/.casafx.dyndns.org.
- A partir de ahí, sí montaron directamente y sin el flag -rw los volúmenes service y users en /afs/casafx.dyndns.org/{service,user}.
- Y en /afs/casafx.dyndns.org/user/u/um/umea se montó con flag -rw el volumen umea.
- La razón de este esquema es posibilitar los path RO, como se explicó al hablar de réplicas y el comportamiento del cache manager.

Anuncio de nuevas celdas a otros clientes

Una identidad con permisos de administrador usó éstos para crear el punto de montaje. Con ello, el resto de identidades en otras máquinas con el cliente OpenAFS instalado pueden encontrar el punto de montaje de la nueva celda al utilizar su permiso de listado sobre el directorio /afs. Sin embargo, por la ausencia de registros SRV/AFSDB en DNS para ihep.su, no sabrán qué hacer con ese punto de montaje al no conocer dónde está su volserver.

- En caliente, es necesario usar "fs newcell" (como POSIX root) pues permite modificar la lista de servidores de DB en la memoria del kernel.

```
ls /afs/ihep.su
```

```
ls: cannot open directory /afs/ihep.su: No such device
```

```
fs newcell -name ihep.su -servers afssrv00.ihep.su
ls /afs/ihep.su
#... ls funcionara' siempre que el grupo system-anyuser
#    tenga permisos de listado ahi', segu'n dicte el ptserver
#    de ihep.su.
```

```
afsdoc
atlas
data
...
```

- Para el siguiente reinicio, los registramos en /etc/openafs/CellServDB como sabemos:

```
printf '>ihep.su            #IHEP\n' >> /etc/openafs/CellServDB
printf '194.190.165.195     #afssrv00.ihep.su\n' >> /etc/openafs/CellServDB
```

1.5.4. Mecanismos para copia de respaldo: dumps, backups, copies

Si las replicaciones proveen tolerancia a fallos (entre otros), los mecanismos de copia de respaldo permiten recuperaciones ante desastres. OpenAFS dispone de varias técnicas y criterios de uso:

Dump (puntual y manualmente)

Una primera solución de respaldo ante posibles desastres son los volcados del volumen a un fichero.

- Es una técnica utilizable de forma puntual.
- El flag -clone hace que no se bloqueen las escrituras del volumen. Con volcados de réplicas no podemos usar "-clone".
- El flag -time permite hacer incremental el volcado: si el argumento es 0 el volcado es total, si es "mm/dd/yyyy hh:MM" vuelca las diferencias entre la fecha actual y aquella.

```
vos dump -id user.umea -clone -time 0 -file /var/local/afsvol-user.umea.dump \
         -server dklab2.casafx.dyndns.org. -partition /vicepa -verbose
```

La opción -server y -partition son sólo útiles cuando no queremos consultar la VL DB, por ejemplo para seleccionar entre una réplica particular y no, como ocurriría en otro caso, la primera conocida en la VL DB.

```
vos dump -id root.cell.readonly -time 0 \
         -file /var/local/afsvol-root.cell.dump \
         -server dklab2.casafx.dyndns.org. -partition /vicepa \
         -verbose -localauth


ls -lh /var/local/*dump
```

La contraparte a "vol dump" es "vol restore". La restauración, en caso de que exista aún el volumen original, contempla la posibilidad de hacerse incremental (-overwrite i), o no (-overwrite f[ull], -overwrite a[bort]).

```
vos restore -name user.umea -file /var/local/afsvol-user.umea.dump \
            -server dklab2.casafx.dyndns.org -partition /vicepa \
            -overwrite f -verbose
```

```
 ...
 Restored volume user.umea on dklab2.casafx.dyndns.org /vicepa
```

Existe un programa para examinar el formato de los volcados:

- restorevol (viene con openafs).
- dumpscan: http://dl.central.org/dl/software/dumpscan/dumpscan-1.2.tar.gz

Backup (regular y automáticamente)

Es una técnica utilizable regularmente, de hecho openafs provee un sistema propio para crearlos de forma cronológica. Los backups quedan en el mismo servidor que su volumen base.

- Facilitan a los usuarios la restauración de datos borrados y otras modificaciones. Una vez más, OpenAFS permite delegar partes de la administración liberando a los administradores para tareas más cruciales.
- Es conveniente conocer que los backups son parte del sistema de respaldo a *cinta* que incorpora AFS, el AFS Backup System. Uno de los componentes de ese sistema es buserver y, aunque en su momento lo registramos y levantamos, no se dispone de hardware para explorar su uso efectivo, así que no se insistirá más en él. Puede consultarse su documentación oficial en: http://docs.openafs.org/AdminGuide/#HDRWQ248.html. Además, pueden replegarse las instancias de buserver con "bos delete" (man bos_delete),

la contraparte de "bos create". Hecho este comentario, podemos continuar y crear backups a disco.

Creamos el primer backup, que luego actualizaremos cronológica y automáticamente:

```
vos backup user.umea
vos listvol dklab2.casafx.dyndns.org.
```

```
...
user.umea.backup
```

Esta tarea "respaldovolusers" del cron afs hará que se lleven a cabo backups de todos los volúmenes que empiecen por "user" todas las noches a las 3h. Recuérdese que un backup reside en el servidor de su volumen base, luego si tuviésemos volúmenes de usuarios en todos los fileserver, debiéramos crear esta tarea del sistema cron de OpenAFS en todos ellos.

Podemos simular y ver qué volumenes se afectarían con el prefijo "user":

```
vos backupsys -prefix user "03:00" -dryrun
```

Y entonces ya:

```
bos create -type cron -instance respaldovolusers \
           -server dklab2.casafx.dyndns.org. \
           -cmd "/usr/bin/vos backupsys -prefix user -localauth" "03:00"


bos status -server dklab2.casafx.dyndns.org. -long
```

```
...
Instance respaldovolusers, (type is cron) currently running normally.
  Auxiliary status is: run next at Mon Aug 22 03:00:00 2011.
  Command 1 is '/usr/bin/vos backupsys -prefix user -localauth'
  Command 2 is '03:00'
```

- Anotación: si se mueve con "vos move" el volumen base, el sistema automáticamente mueve el volumen y su backup.

Hacer disponibles los contenidos de un backup Para usar sin restaurar la copia de respaldo y, por ejemplo, recuperar archivos antiguos, la identidad umea podría simple-

mente montarla donde tiene permisos (en su home). Como siempre, da igual dónde resida el backup, y podemos montarlo desde dklab1 o dklab2:

```
# Montaje:
cd /afs/casafx.dyndns.org/user/u/um/umea
fs mkmount .backup user.umea.backup
# Uso:
ls .backup/
# Desmontaje:
fs rmmount -dir .backup
```

Si el administrador monta por defecto estos volúmenes, debiera avisar al usuario de su existencia, localización, ritmo de actualización, carácter RO y de que no consumen adicionalmente cuota de disco[34].

Copies/Renames

OpenAFS, en las versiones actuales, tiene la posibilidad de copiar volúmenes (cambiando el nombre) y renombrarlos a los originales como forma de backup. El flag -offline hace que los clientes no puedan usar la copia y les pase desapercibida.

```
vos copy -id user.umea -verbose \
         -fromserver dklab2.casafx.dyndns.org. -frompartition a \
         -toname user.umea.copy -toserver dklab1.casafx.dyndns.org. \
         -topartition a -offline
vos listvol dklab1.casafx.dyndns.org.
```

```
...
user.umea.copy
```

Para restaurar, configuraríamos online la copia y la renombraríamos:

```
vos online -server <host> -partition <part> -id <volname>.copy
vos rename <volname>.copy <volname>
```

A veces nos decidimos a hacer la restauración porque, por ejemplo, el servidor remoto

[34] http://docs.openafs.org/AdminGuide/#HDRWQ201.html#HDRWQ204

que albergaba el volumen base dejó de funcionar permanentemente. En ese caso es también conveniente borrar a mano las entradas que hacen referencia a ese servidor o a esa partición del servidor, y ordenar una sincronización de la VL DB; ejem:

```
vos delentry -server <server_averiado> -partition a

vos syncvldb \

    -server <server_donde_hemos_puesto_online_y_renombrado_la_copia>
```

Conversión de una de las réplicas en un volumen base RW

Las últimas versiones de openafs también permiten este comportamiento:

```
vos convertROtoRW <server> <partition> <volumename>.readonly

bos salvage <server> <partition> <volumename>.readonly
```

A veces esta solución se aplica cuando el servidor que alojaba el volumen base queda desconectado no se sabe cuánto tiempo, a la vez que se necesita trabajar sobre el volumen. Entonces si se dispone de una réplica local, se puede hacer la conversión anterior y, si la réplica era realmente reciente, borrar el anterior volumen cuando éste volviese a estar disponible:

```
vos remove -server <server_remoto> -partition <partition> -id <volumename>
```

1.5.5. Comprobaciones en volúmenes

Offline

Ante cortes del suministro eléctrico, el sistema OpenAFS puede rehusar a cargarse al detectar inconsistencias en los volúmenes y estructuras de las que se hace cargo; usualmente volúmenes RW pueden quedar parcialmente marcados como "on". La solución es dejar que /usr/lib/openafs/salvager haga las comprobaciones (a todas las particiones AFS si se omite -partition), debe ser invocado así:

```
/usr/lib/openafs/salvager -partition /vicepa -inodes -force -salvagedirs \

                          -showlog

# Y, si todo fue bien, ya podemos volver a intentar cargarlo:

invoke-rc.d openafs-fileserver start
```

Online

Tradicionalmente lanzaríamos "bos salvage" y éste pararía el fileserver, realizaría la comprobación y lo reiniciaría nuevamente. Con el nuevo soporte de "Demand Attach", no deberíamos iniciar comprobaciones de esta manera sino dejar que operen los automatismos implementados con "Demand Attach".

```
bos salvage -server dklab2.casafx.dyndns.org. -partition /vicepa \
            -showlog -localauth
# Nos advertira' que no ejecutara' la accio'n a no ser que estemos seguros
# de lo que hacemos y demos muestra de ello pasando el flag -forceDAFS
```

1.5.6. Comprobaciones en las bases de datos de OpenAFS

Offline

Las DB también pueden comprobarse offline; por ejemplo en dklab2:

```
# VL DB:
bos shutdown -instance vlserver -server dklab2.casafx.dyndns.org. -localauth
vldb_check -entries -uheader -servers -verbose \
           -database /var/lib/openafs/db/vldb.DB0
bos startup  -instance vlserver -server dklab2.casafx.dyndns.org. -localauth
# PT DB:
bos shutdown -instance ptserver -server dklab2.casafx.dyndns.org. -localauth
prdb_check -entries -uheader -verbose -database /var/lib/openafs/db/prdb.DB0
bos startup  -instance ptserver -server dklab2.casafx.dyndns.org. -localauth
```

Online

Para la base de datos (opcional) relacionada con los respaldos a cinta, OpenAFS dispone de:

```
backup dbverify -detail -localauth
```

1.5.7. Diferentes vistas del árbol AFS según la arquitectura de CPU del cliente

El caso de uso típico son los volúmenes que almacenan código ejecutable (programas, librerías, imágenes de instalación SO...). Para usar esta característica, tendremos en cuenta dos conceptos:

- Mecanismo de traducción "@sys". El fundamento de las vistas es hacer enlaces simbólicos a una ruta que incluya el nodo especial "@sys", pues este nodo apuntará a un directorio u otro según el tipo de cliente.
- "sysname": OpenAFS define una lista abierta de posibles tipos de cliente. Este <sysname>es especificado a su vez con un binomio <hardwareplatform>_<SO>. Para consultar qué sysname está usando nuestro cliente OpenAFS (módulo openafs.ko):

```
fs sysname
```

```
Current sysname is 'i386_linux26'
```

Existen valores preestablecidos como ése, si bien el cliente puede configurar cualquier string y, más aún, puede configurar una lista de ellos, de forma que se intente el mecanismo de traducción "@sys" por orden hasta el éxito.

```
fs sysname -newsys "i386_linux30" -newsys "i386_linux26" \
           -newsys "i386_linux24" -newsys "foo_bar"
```

Estamos en condiciones de poner un ejemplo del mecanismo de traducción. Supongamos que el administrador de sistemas crea en el árbol AFS los siguientes directorios así como el siguiente enlace simbólico con el "@sys":

```
mkdir -p <parent>/platform/i386_linux26/bin
mkdir -p <parent>/platform/sgi_65/bin
ln -s    <parent>/platform/@sys/bin  <parent>/bin
```

... es reseñable cómo "@sys" ha quedado en el mismo nivel donde están los directorios <sysname>:

```
cd <parent>
tree
```

```
platform/i386_linux26/bin

platform/sgi_65/bin

bin -> /platform/@sys/bin
```

Entonces, el cliente intenta acceder a la ruta <parent>/bin. El servidor traducirá ésto a, en virtud del enlace simbólico, <parent>/platform/@sys/bin. A continuación traducirá el @sys que, en virtud de la lista de 3 sysname's que configuramos en el cliente, lo convertirá por orden primero en <parent>/platform/i386_linux30/bin. Éste no existe, luego el servidor prueba con el siguiente: <parent>/platform/i386_linux26/bin, que sí está y es la ruta a la que conduce al cliente.

Aún tenemos algo más que decir sobre vistas, esta vez sobre el nombre de los volúmenes y los permisos:

- Nombres de los volúmenes para código ejecutable: conviene seguir una convención; pongamos como ejemplo: <sysname>.usr.bin. En este ejemplo el prefijo alude al <sysname>, y el sufijo al punto de montaje tras @sys, en este caso se espera montar en <parent>/software/platform/<sysname>/usr/bin.
- Permisos en los puntos de montajes[35]: "rl" no permitido a system:anyuser si los binarios no los deberían usar usuarios sin previa autenticación. Otra posibilidad es dar "l" sólo a system:anyuser y "rlk" a system:authuser (el permiso "k", lock, porque algunos programas se ejecutan desde ahí y necesitarían lock).

Ejemplo completo

Por tanto, gracias a los enlaces simbólicos con @sys, a los enlaces simbólicos a otros directorios etc podríamos construir cualquier árbol pertinente para almacenar binarios, y a continuación montar los volúmenes con el formato de nombre especificado. Veamos un ejemplo en que se construye un árbol para binarios en 2 niveles "./" y "./usr", con ramas bin, lib y sbin; para cada nivel se ofrecen vistas para linux, windows y mac sobre i386:

[35] http://docs.openafs.org/AdminGuide/#HDRWQ417.html

```
/afs/<cell>/software/platforms/i386_linux26/bin
                                          lib
                                          sbin
                                          THIS_PC
                                i386_linux24->i386_linux26
                                i386_nt40/bin  -> .
                                          lib  -> .
                                          sbin -> .
                                          THIS_PC
                                x86_darwin90/bin
                                          lib
                                          sbin
                                          THIS_PC
                        /bin    ->platforms/@sys/bin
                        /sbin   ->platforms/@sys/sbin
                        /lib    ->platforms/@sys/lib
                        /THIS_PC->platforms/@sys/THIS_PC
#
# Los ficheros THIS_PC son ficheros simples con la
# salida esperada de fs sysname.
```

En las hojas de este árbol, montaríamos los volúmenes <sysname>.bin, <sysname>.lib... por ejemplo para ./bin y máquinas linux 2.6 en i386:

```
vos create <serverfqdn> a i386_linux26.bin -maxquota 200000
mkdir -p /afs/<cell>/software/platforms/i386_linux26/bin
fs mkmount /afs/<cell>/software/platforms/i386_linux26/bin
fs setacl /afs/<cell>/software/platforms/i386_linux26/bin system:anyuser l
fs setacl /afs/<cell>/software/platforms/i386_linux26/bin system:authuser rlk
```

El segundo nivel, "./usr", podría tener un aspecto similar:

```
/afs/<cell>/software/usr/platforms/i386_linux26/bin
                                                lib
                                                sbin
                                      i386_linux24->i386_linux26
                                      i386_nt40/bin  -> ..
                                                lib  -> ..
                                                sbin -> ..
                                      x86_darwin90/bin
                                                lib
                                                sbin
                            bin ->platforms/@sys/bin
                            sbin->platforms/@sys/sbin
                            lib ->platforms/@sys/lib
```

En las hojas de "./usr", montaríamos los volúmenes <sysname>.usr.bin, <sysname>.usr.lib... de una forma similar a como lo hicimos en el nivel uno "./" de software.

Está fuera del alcance de esta introducción dar más detalle sobre las vistas. Cada entorno en producción precisará una política de directorios, sistemas soportados y software que ofrecer; aún así, baste decir que lo expuesto hasta ahora es suficiente para, usando "mkdir", "ln", "vol create", "fs mkmount" y "fs setacl", desplegar este esquema propuesto como ejemplo o cualquier otro que resulte más adecuado en el entorno final.

1.6. Aún más

Hasta aquí hemos poco más que rascado la superficie de OpenAFS: el número de posibilidades, opciones y detalles implementados de este producto industrial es ingente. A continuación se dan referencias sobre algunas otras posibilidades interesantes. La documentación de OpenAFS consiste en tres documentos: una guía de instalación rápida (QuickStartUnix), una guía completa de administración (AdminGuide) y una guía de usuario (UserGuide). Existe también un documento con la descripción aislada de comandos y ficheros (Reference) y que se corresponde con las páginas man instaladas. Todas ellas las hemos estado referenciando continuamente en nuestra exposición anterior, y es el camino que seguimos ahora:

http://docs.openafs.org

- Sobre autentificarse en varias celdas a la vez: Se consiguen distintos TGT KERBEROS usando la variable KRB5CCNAME para que residan en ficheros distintos, y

para que "aklog -cell <celda>" sepa cuál debe usar al pedir los tokens. Una vez recibidos los tokens, el cliente usará uno u otro según nos situemos en el subárbol del punto de montajela de cada celda (/afs/celda1.org/... o /afs/celda2.org/...), sin necesidad de más variables o intervención alguna del usuario. Para descartar los tokens de alguna de las celdas, se usará "unlog -cell <celda>". Puede consultarse de forma más detallada:

  - http://docs.openafs.org/UserGuide/index.html#HDRWQ20.html#Header_46

- Añadir/Quitar discos y volúmenes:

  - http://docs.openafs.org/AdminGuide/#HDRWQ130.html

- AFS/NFS translator: interfaz NFS al árbol AFS. (Adicionalmente, opinamos que no sería nada difícil, en equipos que tampoco tuviesen cliente NFS, exportar partes del árbol AFS usando scp/sshfs. Incluso el sshd podría tener su identidad en afs y así poder obtener un token e interactuar con el sistema de ACL).

  - http://docs.openafs.org/AdminGuide/#HDRWQ603.html
  - http://docs.openafs.org/AdminGuide/#HDRWQ595.html

- Monitorizar el sistema: scout, fstrace, afsmonitor

  - http://docs.openafs.org/AdminGuide/#HDRWQ323.html

- Más sobre respaldos:

  - http://docs.openafs.org/AdminGuide/#HDRWQ201.html
  - http://docs.openafs.org/AdminGuide/#HDRWQ248.html
  - http://docs.openafs.org/AdminGuide/#HDRWQ283.html

- Más sobre reparaciones (salvager):

  - http://docs.openafs.org/AdminGuide/#HDRWQ232.html

- Más sobre cuotas:

  - http://docs.openafs.org/AdminGuide/#HDRWQ234.html

- ...

LyX