Running OpenAFS Clients and Servers on Linux

Stephan Wiesand
European AFS Workshop
Rome, 2009-09-28

HELMHOLTZ
GEMEINSCHAFT

DESY

“Linux” ?

- There are dozens of GNU/Linux distributions
- openafs.org offers pre-built packages for four of them
  - Fedora & Red Hat Enterprise Linux (& derivatives)
  - OpenSUSE & (Novell) SUSE Linux Enterprise Server
- Some more distributions come with pre-built packages
  - At least: debian, Ubuntu, Mandriva
- OpenAFS built from source should work on any of them
- When going into details, this talk will focus on RHEL + rebuilds
  - Red Hat Enterprise, CentOS, Scientific Linux
- Will use the common term *EL* to refer to those
  - “EL5” means RHEL 5 or CentOS 5 or SL 5

Building and Installing OpenAFS from Source

- Method 1: `./configure --prefix=/some/path; make; make install`
  - will install everything under /some/path (default: /usr/local)
    - common: --prefix=/usr , but with some further customization
      - in particular: configuration files typically go into /etc/openafs/
- Method 2: `./configure --enable-transarc-paths; make; make dest`
  - will create *sysname*/dest in the source tree
    - with subdirectories bin, lib, include, sbin, root.client, root.server
      - to be moved to their final destination
      - configuration files go into /usr/vice/etc/
- both methods may require pointing configure to the kernel headers:
  - `./configure --with-linux-kernel-headers=/usr/src/linux-2.6.18-....`
- there are many more configure options
  - to specify paths or enable/disable features

Configuring and Starting the Client

- client components:
  - kernel module (/usr/vice/etc/modload/libafs-2.6.18-128.7.1.el5.mp.ko)
  - daemon (/usr/vice/etc/afsd)
- configuration files required by afsd:
  - CellServDB (available from grand.central.org)
  - ThisCell (echo “my.cell” > /usr/vice/etc/ThisCell)
  - cacheinfo (echo “/afs:/usr/vice/cache:100000” > /usr/vice/etc/cacheinfo)
    - filesystem must be ext2 or ext3 !
- starting the client:
  - create the AFS mountpoint (mkdir /afs)
  - load the module matching your kernel (insmod libafs-...ko)
  - start the daemon (afsd [options...])

Installing init script & sysconfig File

- copy them from the source tree:
  - cp src/afsd/afs.rc.linux /etc/init.d/afs
  - cp src/afsd/afs.conf.linux /etc/sysconfig/afs
- both files are out of date
  - autotuning should now be done by afsd
- require fixes & customization
  - default cache size too small, pathsx
- client can now be started with `service afs start`
  - init script will read /etc/sysconfig/afs
  - and create cacheinfo and choose afsd options automatically
    - setting up ThisCell and CellServDB is still required
- for automatic start at boot: `chkconfig afs reset`

Using Pre-Built Packages Instead

- example: Scientific Linux 5

  -
    ```
    yum install openafs-client \
     kernel-module-openafs-`uname -r`
    ```

  -
    ```
    echo “my.cell” > /usr/vice/etc/ThisCell
    ```

  -
    ```
    service afs start
    ```

Differences & Commonalities in Packages

- service names may differ
  - afs & afs-server vs. openafs-client & openafs-server
- paths may differ
  - /usr/vice/etc/ vs. /etc/openafs
  - /usr/vice/cache vs. /var/cache/openafs
- configure options may differ
  - ---enable-fast-restart …
- a sysconfig file should always be present
  - names may differ (afs vs. openafs)
  - content and functionality will differ
- naming convention for kernel module packages will differ
- policies may differ (w.r.t. update strategy, backward compatibility)

Example: Available Builds for the EL Family

- There are (at least) 3 different sets of packages available that could be used on RHEL, CentOS and Scientific Linux
  - Here, we'll compare those from openafs.org to those coming with SL
    - Both were quite similar a few years ago, but since have diverged significantly
    - Policies are quite different, divergence is mainly a consequence
      - SL: stability, continuity, backward compatibility
        - currently provides 1.2.13 with SL3, 1.4.7 with SL4 and SL5
        - client/server service name: afs/afs, afs/afs, afs/afs-server
        - /etc/sysconfig/afs is backward compatible in functionality since SL3
        - naming convention for kernel module package is unchanged since SL3
      - openafs.org: supports latest OpenAFS release
        - currently provides 1.4.11 for EL4 and EL5
        - service names were afs/afs, now are openafs-client, openafs-server
        - /etc/sysconfig/afs was stripped rigorously a while ago
        - naming of kernel module pks changed, following Red Hat convention
    - Both are ok (from my point of view), but target different sites

Modifying Pre-Built Packages

- it's not uncommon to spin off package builds for the own site
- get the source RPM and install it
  - `rpm -ivh openafs...src.rpm`
- modify the spec file
  - change configure options, modify default config, etc.
  - always modify the release number
- rebuild (consult output of `rpm -qip openafs...src.rpm` for info):
  - `rpmbuild -ba openafs.spec`
  - `rpmbuild -ba --define 'kernel 2.6.18-128.7.1.el5' openafs.spec`

Customization with RPM Triggers

- example: SL RPMs may overwrite CellServDB when updated
- to always override with your site's CellServDB, create spec, build the RPM, and install it on all clients:

```
Summary: Install our local CellServDB
Name: override_cellservdb
Version: 20090928
Release: 1
BuildArch: noarch
Source: CellServDB
%description
…
%install
mkdir -p %{buildroot}/usr/share/%{name}
install -m 644 %{SOURCE0} %{buildroot}/usr/share/%{name}
%files
/usr/share/%{name}
%triggerin -- openafs-client
install -m 644 /usr/share/%{name}/CellServDB /usr/vice/etc
```

Client Configuration & Tuning

- Best practice: have a separate filesystem for the cache
  - typical size: O(1 GB), must be ext2 or ext3
  - configure the cache size to be a fair bit smaller than the filesystem
    - 70% is safe, more possibly isn't
      - remember the journal (32 MB), keep an eye on inodes (df -i)
  - tuning should probably be left to afsd itself
    - do not specify -daemons, -dcache, -stat, -volumes
    - but increasing the chunksize may help if many large files are accessed
      - try values around 19 or 20 (.5, 1 MB)
        - check automatic setting with cmdebug localhost -cache
- Alternative: memory cache (local access, fast network & servers)
  - `afsd -memcache`, still requires specifying a directory in cacheinfo
  - may require tuning other parameters; example for 256 MB cache:
    - `-stat 8192 -disable-dynamic-vcaches -daemons 8 -chunksize 20 -volumes 64`

Integration with System Authentication: PAM & SSH

- When using Kerberos 5, on current EL5 systems there's very little left to do
  - comes with an AFS aware pam_krb5
- `authconfig --enablekrb5` ... (command or kickstart option)
- /etc/krb5.conf:

```
[appdefaults]
        pam = {
                external = sshd
                tokens = sshd login
                ...
              }
```

- /etc/ssh/sshd_config:

```
UsePam yes
GSSAPIAuthentication yes
GSSAPICleanupCredentials yes
```

Firewalling the Client

- no action is strictly required, the client will work with iptables active with the default configuration
  - client initiates the traffic, iptables “related packets” logic lets replies from servers pass the local firewall
    - but this is time limited, and callbacks may arrive after hours of inactivity
    - the ipt_recent module can be used to let these pass as well
    - /etc/sysconfig/iptables:

```
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A OUTPUT -p udp --dport 7000 -m recent --update --name "AFS_CB_OK" --rdest -j ACCEPT
-A OUTPUT -p udp --dport 7000 -m recent --set    --name "AFS_CB_OK" --rdest -j LOG ...
:RH-Firewall-1-INPUT - [0:0]
-A INPUT -j RH-Firewall-1-INPUT
-A FORWARD -j RH-Firewall-1-INPUT
# loopback interface
-A RH-Firewall-1-INPUT -i lo -j ACCEPT
...
-A RH-Firewall-1-INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# AFS callbacks from fileservers we visited recently:
-A RH-Firewall-1-INPUT -p udp --dport 7001 -m recent --rcheck --seconds 86400 \
--name "AFS_CB_OK" --rsource -j ACCEPT
…
```

Servers

- DB Servers
  - do not require much resources
    - perfectly ok to run in a VM
  - but systems should be reliable - even when clustered
- File Servers
  - should run on bare metal
  - fast hardware
  - RAM helps
- Can be co-located, but common practice is to separate them
- No client (nor kernel module) is required on servers
- Package to install is typically openafs-server
- Will not cover cell setup here

The Fileserver

- Stack:
  - Disk + Firmware
  - Enclosure/Backplane + Firmware
  - Controller + Firmware
  - OS
    - Driver
    - SCSI/VFS/MM layers
    - Filesystem
  - OpenAFS fileserver/volserver
- This whole stack has to work and perform well
  - choose hardware wisely
    - consider getting everything from the same vendor, certified for your OS
  - keep firmware & software up to date

Storage Hardware: DAS vs. SAN

- AFS comes with a number of SAN features built in
  - volumes - can be grown, shrunk, and moved around online
  - backup volumes = single level snapshots
  - replication - r/o today, r/w in the future
- There's nothing wrong with using SAN storage for vice partitions
  - maximum flexibility
    - add servers, share storage with other applications
  - best redundancy - incl. controllers, HBAs & external data paths
- There's nothing wrong with using Direct Attached Storage either
  - best price/capacity, price/performance & aggregate performance
  - simple
  - truly independent storage units - sharing nothing

Disk Technology

- SATA for AV applications
- SATA for desktops
  - not suitable for use in file servers
- Enterprise (or “Nearline”) SATA
  - 5400-7200 rpm, up to 2 TB
- Nearline SAS
  - better protocol, other parameters like Nearline SATA
- SAS (& FC)
  - 10000 - 15000 rpm, up to 600 GB
  - much faster, more reliable & durable
- If budget and space allows, choose SAS for AFS storage

RAID

- Always use a redundant RAID level
  - 10 - best write performance
  - 6 or 60 - best reliability, more net capacity
- Experiment with stripe size
  - larger is not necessarily better; stay <= 128 kB with SATA drives
  - optimum depends on usage patterns; in doubt try 64 kB
- Hardware or Linux Software RAID (md)
  - use battery backed write cache, or write-through
- Have disks scrubbed & parities checked weekly
  - Linux md: `echo “check” >> /sys/block/md?/md/sync_action`
    - and monitor /sys/block/md?/md/mismatch_count
  - mdadm update coming with EL5.4 will now do this automatically

Choosing the Filesystem for Vice Partitions

- candidates: ext3, xfs
  - ext4 soon, btrfs hopefully later, no hope for zfs on Linux
  - stay away from reiserfs
- performance of ext3 and xfs is very similar
  - exception: deletion of large files is very slow on ext3
- xfs probably better at avoiding fragmentation, defragmenter exists
  - AFS comes with an effective tool for defragmentation: vos move
- only xfs can handle > 16 TB
- both can be grown online
  - ext3 can also be grown - and shrunk - offline
- do not use xfs on 32-bit systems (w/ 4 kB stacks)
- in doubt, choose ext3

32-bit or 64-bit ?

- for small servers (< 4 GB RAM), this doesn't matter much
- larger amounts of RAM are handled more efficiently w/ 64-bit
- 64-bit AFS servers work well
  - any 64-bit specific bugs left?
- 64-bit is not going away, 32-bit may

- in doubt, run 64-bit Servers

RAID, Partitions, Filesystems & Alignment

- block devices > 2 TB require a GPT disk label (= partition table)
  - parted: `mklabel gpt`
  - Linux (EL <= 5) can not boot from such a device
- partitions should be (full?) stripe aligned
  - example: 8+2 disks RAID-6 w/ 64 kB stripe size
  - `mkpart vicepa 512kib 2tib`
  - `mkpart vicepb 2tib 4tib`
- filesystem should be told about the RAID geometry:
  - `mkfs.ext3 -O dir_index -E stride=16 -L /vicepa /dev/sdb1`
  - `mkfs.xfs -d sunit=128,swidth=1024  -L /vicepb /dev/sdb2`

Choosing the I/O Scheduler

- This is the most effective single optimization step - and trivial to do
- The kernel comes with 4 schedulers = I/O reordering strategies
  - CFQ - completely fair queuing (per process request queues, default)
  - deadline - cluster I/Os as much as possible, but avoid starvation
  - anticipatory - heuristics to anticipate (=wait for) next I/O in stream
  - noop - do nothing, leave dirty work to others (RAID controller)
- anticipatory and CFQ are good for interactive workloads
  - not for file servers
- With hardware RAID, try deadline and noop
- With software RAID, use deadline
- To change on the fly: `echo noop >> /sys/block/sdb/queue/scheduler`
  - add command to /etc/rc.local to make choice persistent

Readahead

- Can make a big difference
  - depending on workload
- To change on the fly: `blockdev --setra <value> /dev/sdb`
  - sets readahead for device to <value> * 512 Bytes
- Try values like 4096, 8192, 16384 for large file read workloads
  - multiples of the RAID stripe width may make most sense
- Too high values may actually hurt
  - especially with insufficient RAM

AFS Fileserver Parameters

- start with -L (for “Large”; specifying -p 128 should no longer be necessary)
- candidates for tuning include -udpsize, -rxpck, -l, -s, -vc,...
  - and -cb
    - to find out whether the fileserver had to recycle space for callbacks:

```
% xstat_fs_test -fsname zentaur2 -collID 3 -onceonly|grep G
             0 GotSomeSpaces
             0 GSS1
             0 GSS2
             0 GSS3
             0 GSS4
             0 GSS5
% xstat_fs_test -fsname zentaur1 -collID 3 -onceonly|grep G
        398932 GotSomeSpaces
        794522 GSS1
        397261 GSS2
          1671 GSS3
             0 GSS4
        397261 GSS5
```

    - all values should be 0 - if not, increase default value of 64000
    - on SL, xstat_fs_test is packaged in openafs-debug

Jumbograms

- slighthly confusing issue
  - for years, recommendation was to run without
  - apparently, they were always used - and could not be turned off
  - until this was recently fixed
  - -nojumbo is now the default
- found that -jumbo is faster in our environment (~ 5% with 1.4.11)

Summary

- Linux is an interesting platform to run OpenAFS on
  - lots of choices: distribution, openafs packages, filesystem, 32/64-bit,…
    - usually, more than a single “good” one
- Linux servers will provide reasonable performance after very basic tuning
  - choose the right disk technology
  - choose the right I/O scheduler
  - adjust readahead
  - try -jumbo
  - stripe align your partitions, inform mkfs about RAID geometry
- A file server set up like this should easily saturate 2 bonded GbE links with a dozen clients doing large file I/O, and scale up to at least 100 clients doing (nothing but) this.