CVMFS for the rest of us

Dennis van Dok

2016-06-06

CVMFS for the masses

CVMFS is great for large organisations. But for small research teams the use of CVMFS is a real challenge:

- set up and maintain a repository
- take care of a Stratum-0 server
- negotiate the replication at Stratum-1 sites
- negotiate with sites to include the repository in their CVMFS configuration

Now image dozens of small e-science groups knocking on your door to get their repositories mounted.

Our solution

Nikhef and SURFSara have jointly set up softdrive.nl to offer a single repository for all e-science users in the Netherlands.
The system consists of

- a user interface system, where users can log on (with ssh) and upload their software
- a Stratum-0 server which copies the user's files at regular intervals
- Stratum-1 at Nikhef and RAL (thanks Catalin)
- mounted on all grid resources in the Netherlands

User documentation

login message

Welcome to the SoftDrive, the National CVMFS self-service interface. On this machine you will be able to maintain your own software tree and publish it to the National CVMFS repository. In short this is achieved as follows:

1. prepare your software somewhere in your home directory
2. when satisfied, install your software under /cvmfs/softdrive.nl/$USER
3. then trigger publication by executing command publish-my-softdrive
4. wait patiently for your new software to become available in CVMFS

More user documentation

There's an extensive README with full disclosure about the guts of the system, including an explaination about what not to expect:

- data will be public
- it's read-only
- only meant for software, no science data
- default quota of 2GB/user

What's missing is a the introduction of nested catalogs.

Monitoring

The Stratum-0/1 servers are monitored for disk use, but currently have ample space to accomodate many users.
There is an end-to-end monitoring setup where an update to the repository is triggered every 5 minutes, and the time it takes to reach the worker nodes is reported. We've given ourselves some headroom, but users will expect to see their updates within a few hours.

Some quirks

Naturally not everything went smoothly right from the start.

bizarre repo growth when the end-to-end nagios check was implemented

Apparently a new revision every 5 minutes could be a bit much for the system. The administrative overhead of all these started to add up very quickly.
Thankfully there is garbage collection.

Publishing became very slow with auto garbage collection

When garbage collection was turned on, the transaction lengths went from 5 minutes to half an hour, and even more. This effect was seen on both on stratum-0 and stratum-1.
The solution for this problem may be to switch to nightly garbage collection.

Softdrive is still in beta

We have some early adopters as well as software consultants who are testing the system; because of the quirks in the previous slide we are still in beta (hopefully not for too much longer).
Some user's directories are so large they warrant their own nested catalog. Or should they all just be nested catalogs?

Some numbers

Here is a list of directories with the most files. All in all more than 800k files are in the repository (even though it is still under 50GB),

1273	mmoisse/project_mine	12002	miterson/R-3.2.5
1360	maartenk/project_mine	16828	emilioe/hathi-client
2234	lyklev/octave-3.8.2	17375	anatolid/hathi-client
2345	anatolid/octave-4.0.0	17375	mathijsk/hathi-client
2951	niekb/schultz	27229	jvanvugt/RepeatHunter
3962	emilioe/my_lofarsoft_build	27840	wjvriend/lofar_stack
5409	anatolid/gcc-4.8.5	48392	abalzer/gcc-hess
5409	apmechev/gcc-4.8.5	53792	jlinthor/env
5409	lyklev/gcc-4.8.5	75997	dgarza/anaconda-2-2.4.0
5409	s5p/gcc-4.8.5	77721	apmechev/anaconda-2-2.4.0
5410	wjvriend/local	78665	s5p/anaconda-2-2.4.0
6940	abalzer/hap_ICRC2015	83939	emilioe/anaconda-2-2.4.0
7978	jeroens/hathi-client	121287	anatolid/anaconda-2-2.4.0
9037	jeroens/miniconda2	173986	maartenk/gridtools

some questions for the experts

- How does garbage collection on the Stratum-0 relate to the Stratum-1? Does the Stratum-1 retain older revisions even if the Stratum-0 doesn't?
- How do you find the number of revisions in a repo?
- How should we handle the creation of nested catalogs? Automated or manual?
- What would be a better monitoring option?

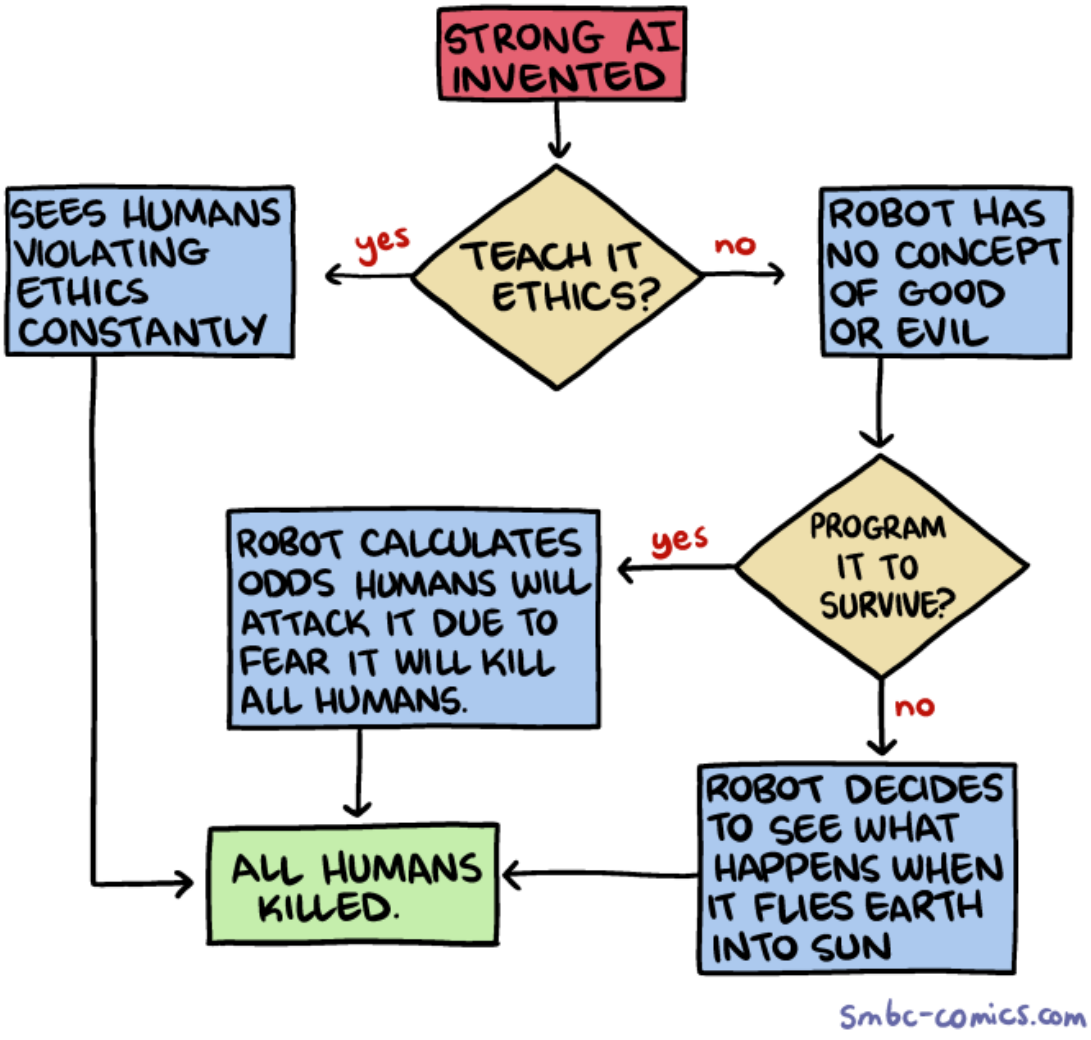
Thanks for your attention

Acknowledgements

- Coen Schrijvers (SURFSara), user documentation and monitoring.
- Catalin Condurache (RAL), fail-over Stratum-1.
- Ronald Starink for the initial setup of the CVMFS system at Nikhef.

- `http://doc.grid.surfsara.nl/en/latest/Pages/Advanced/grid_software.html`

Obcomic (picture unrelated)



Once you realize there is no hope, you can relax and just enjoy the progress in machine learning.

The source

#!/bin/sh

Synchronise the Stratum 0 from an upstream rsync source, with multiple sync points.
The remote source directory is traversed, searching for files named cvmfs.modified. If such a file exists, its timestamp is compared to the timestamp of the same file on the target directory (i.e. on the stratum-0 side). If the remote file is newer, or if the local file is absent, the directory is added to the list to synchronise. This method gives the maintainer the option to synchronise directories selectively, which is useful if a repository is shared among various users (or user groups), each with their own directory maintained independently of the others.

bail out on errors

```
die() {
    err=${2:-1}
    if [ -n "$logfile" ]; then
        log "$1"
    fi
    echo -e "$1" >&2
    exit $2
}
```

log formatting

prefix output with a timestamp

```
log() {
    if [ -n "$logfile" ]; then
        d=`LC_TIME=C date +'%b %d %H:%M:%S'`
        echo $d "$@" >> "$logfile"
    else
        echo "$@"
    fi
}
```

initial parameters

These parameters must be set on the command line

repo=
rsyncsrc=

If rsyncsrc requires a password this variable holds the name of the password file

rsyncopts=

 # Exclude certain trees based on an excludes−from file
excludeopts=

 # These are defaults for cvmfs, but can be
 # overridden for debugging purposes
rsyncdest=/cvmfs *# the destination once a transaction started*
rdonlylocation= *# the compare−dest before transactions start*
triggerfile=cvmfs.modified *# the file to check whether an update is needed*

parsing command line options

```
while getopts :r:s:d:S:R:i:l:t: OPT; do
  case $OPT in
    r|+r)
      repo="$OPTARG"
      ;;
    s|+s)
      rsyncsrc="$OPTARG"
      ;;
    d|+d)
      rsyncdest="$OPTARG"
      ;;
    S|+S)
      rsyncopts="--password-file␣$OPTARG"
      ;;
    R|+R)
      rdonlylocation="$OPTARG"
      ;;
```

command line options, contd.

```
        i|+i)
          inclexclfile="$OPTARG"
          excludeopts="--exclude-from=$OPTARG"
          ;;
        l|+l)
          logfile="$OPTARG"
          ;;
        t|+t)
          triggerfile="$OPTARG"
          ;;
        *)
            echo "usage: `basename $0` -r repository -s source \
  [ -d rsync destination ] [ -S secretsfile ] \
  [ -R readonly location ] [ -i inclexclfile ] \
  [ -l logfile] [ -t triggerfile ]"
          exit 2
        esac
      done
      shift `expr $OPTIND - 1`
      OPTIND=1
```

handling option sanity

```
if [ -z "$repo" ]; then
    die "missing argument -r repository"
elif [ -z "$rsyncsrc" ]; then
    die "missing argument -s rsyncsource"
fi
```

Set the compare-dest location if it was not set with an option.
Can't do this until $repo is set.

```
if [ -z $rdonlylocation ]; then
    rdonlylocation=/var/spool/cvmfs/$repo/rdonly
fi
```

Make sure there is only one update process at a time

Since the entire process may take a long time, we need a lockfile to signal that the process is already at work.

```
LOCKFILE=/tmp/cvmfs-$repo-update.lock
exec 9> $LOCKFILE;
flock -xn 9 || die "could not lock $LOCKFILE"
```

make sure everything gets cleaned up at the end

```
trap "rm -rf $LOCKFILE $tmp" EXIT;
```

working directory and more sanity checks

create a temporary directory to get the rsync 'cvmfs.modified' files.

```
tmp=‘mktemp -d --tmpdir cvmfs.$repo.XXXXXXXXXX‘ || \
die "cannot create temporary directory. Aborting."
```

Sanity check: see if we can rsync at all. This doesn't actually transfer anything.

```
rsync -q --no-recursive $rsyncopts $rsyncsrc /tmp
test $? -eq 0 || die "Can't rsync from $rsyncsrc, aborting."
```

Phase 1: find the cvmfs.modified files and build a list

This includes first level directories and cvmfs.modified files, but excludes all the rest. We use the read-only location (which is identical to what is currently published as the compare destination), so only newer cvmfs.modified are copied.

```
errstr=`rsync -q -rt $rsyncopts $excludeopts \
                     --filter 'include /*/' \
                     --filter "include $triggerfile" \
                     --filter 'exclude *' \
                     --compare-dest="$rdonlylocation/" \
        $rsyncsrc/ $tmp/ 2>&1`
result=$?
case $result in
    0) ;;
    23) ;; # file missing, this is ok
    *) # other error
      log "$errstr"
      die "rsync failed. Aborting." $result
      ;;
esac
```

Phase 2: sync the directories that need updating

Now `$tmp` contains a copy of the source tree structure, including a `cvmfs.modified` in every tree that needs updating.
The following `find | sed` replaces all of that with just the names of the path elements that we need to sync.
There is an edge case where the `cvmfs.modified` file lives in the root of the tree, in which case the output should be just '`.`'.

Funky regular expressions

Bonus points for understanding the regular expressions.
The original comment in the source code said *It's ugly and my brain hurts.*

```
publish=1 # keep track of failures
transactionstarted=0 # keep track of the transaction
find $tmp -name "$triggerfile" -type f | \
  sed -e "s,.*cvmfs\.$repo\.[^/]*,/.," \
      -e 's,.*/\([^/][^/]*\)/'"$triggerfile"'$,\1,' | while read p
do
```

Start a transaction, do the rsync...

...and if anything goes wrong, abort.

```
  if [ $transactionstarted -eq 0 ]; then
      log "Update requested of cvmfs repository $repo; \
starting transaction"
      cvmfs_server transaction $repo || \
      die "failed cvmfs transaction on $repo"
      transactionstarted=1
  fi
  log "Update requested of directory $p in repository $repo"

  rsync $rsyncopts -r -q -l -t --delete $rsyncsrc/$p/ "$rsyncdest/$repo/$p/"
  if [ $? -ne 0 ]; then
      log "Rsync from $rsyncsrc/$p failed, aborting transaction."
      publish=0
      break
  fi
done
```

finishing up

We resign the whitelist as part of this process, as the usual cronjob to do that can't get a foot in the door.

```
if [ $publish -ne 1 ] ; then
    cvmfs_server abort -f $repo || \
    die "failed to abort the transaction, exiting"
else
    # The rsync succeeded
    cvmfs_server publish $repo || \
    die "failed to publish the new CVMFS repo $repo, exiting"
    log "cvmfs publishing done for $repo"
    # resign the whitelist now
    log "signing whitelist for $repo"
    cvmfs_server resign $repo
fi
```

Did you spot the error?