Nikhef

CVMFS for the rest of us

Dennis van Dok, Nikhef

Grid Deployment Board, CERN, 2017-05-10

Introduction

Nikhef's data processing facility in a nutshell:
Storage and compute cluster

- runs EGI middleware (DPM, CREAM)
- part of a shared infrastructure for science
  - serving dozens of VOs
  - operating on national, European and global level
- half of the NL-T1 (the other half at SURFSara)
- generic, multi-user
  - WLCG experiments are our major tenants
    - but they are not entitled to special treatment

The problem

Grid users have always struggled to get their software to the worker nodes:

- Software stacks can be large
  - and take a long time to download and compile
    - which is wasteful to do for each job
- using the `$VO_SW_AREA` on each site is cumbersome
  - needs to be specially requested
  - software location varies among sites
    - which requires special care not to use absolute paths when linking

CVMFS for the masses

CVMFS is great for large organisations. But for small research teams adoption poses serious challenges:

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

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 (at SURFSara), where users can log on (with ssh) and upload their software
- a Stratum-0 server (at Nikhef) which copies the user's files at regular intervals
  - We already had a Stratum-0 for other uses
- Stratum-1 at Nikhef and RAL
- mounted on all grid resources in the Netherlands

Technical details

The user interface offers users their own writable directory under `/cvmfs/softdrive.nl`

- which is not a cvmfs mount on this machine
  - but it has the same path so absolute paths will work

Users 'publish' their work by touching a flag file in their work directory.
The user interface runs an rsync server:

- polled every five minutes from the Stratum-0
- initial rsync of just the directory structure and the special flag files in each users' home directory
  - timestamps of the flag files are checked for freshness
    - for each directory with newly touched flag, a full rsync is done
    - after starting a cvmfs transaction
    - transaction is committed as rsyncs finish

User friendliness

A strong focus from the start on making the system as accessible and user-friendly as possible, providing:

- proper documentation
- access with ssh
- help
  - through existing structure at SURFSara
- some helpful tools

Not all users in our base are skilled software engineers.

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 explanation 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

(Some users have considerably more than 2 GB, but it's a useful safety net in case somebody accidentally tries to upload their 1 TB data set. Not software engineers, remember?)

Monitoring

Nikhef

We monitor the disk use on the Stratum-0/1 side, and increase disk space as repositories grow in size.

SURFSara

End-to-end monitoring by updating the repository every 5 minutes and measuring the time it takes for the change to reach the worker nodes. We've given ourselves some headroom, but users will expect to see their updates within a few hours.

Some quirks

I would like to report everything went smoothly and it was a huge success from the start. I would like to, but I can't.

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

- transaction lengths went from 5 minutes to half an hour and longer
  - seen on both on stratum-0 and stratum-1.

The solution for this problem was to switch to nightly garbage collection.

More quirks

Having a million files in a single catalog could be problematic

As the repository size began to grow, the system slowed down.

- CVMFS experts helpfully pointed out catalogs could be too large
- moved to having a nested catalog per user
  - may need to have further nested catalogs for power users

Without a nested catalog, the five-minute monitoring updates would create an index for the whole of the repository. It's kind of obvious how this slows down transactions once you understand the principles.

Feedback from the community

We welcome feedback from users as well as from CVMFS experts. Wat we do is hardly unique.
Our users never voiced any complaints.
SURFSara takes care of most of the user support.

- arranging logins on the user interface machine
- consulting with users
  - directing them to the correct part of the infrastructure
    - Grid
    - Hadoop
    - high performance cloud
    - supercomputer

Some numbers

There are about thirty active users of the system at this moment. They often represent user communities with multiple users.
There is currently about 120 GB of data in the repository, in $1.8 \cdot 10^6$ files.
The repositories are not just used on the Grid, but also on the high-performance cloud of SURFSara.
Rumour has it softdrive has spread across the borders, and is even used in the UK.

Thanks for your attention

Acknowledgements

- Coen Schrijvers (SURFSara), user documentation and monitoring.
- Catalin Condurache (RAL), fail-over Stratum-1.
- CVMFS experts on the mailing list (Jakob Blomer, Dave Dykstra, et al.)

Further reading

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

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

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

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?