ET Siesmic Data Backup and Archiving

Andrew Pickford

Initial Problem

- Oct 2023 - Storing/Sharing of seismic survey data for the ET was failing
  - Got a request at Nikhef to increase storage for the Einstein Telescope seismic data
    - there is an ongoing seismic survey in the Euregio Meuse-Rhine border region as a potential site for the ET
    - survey data was being shared using surfdrive
    - surfdrive is a file sharing service hosted at Surf
  - The 500 GB of raw data and analysis files were being shared
    - 500 GB is the maximum size of a surfdrive group folder
  - Surfdrive used as
    - easy file sharing
    - in the mistaken belief it was backed up
  - Some files lost on surfdrive due to local file deletions and subsequent syncing
    - not the case with these file losses but potentially costly to replace, seismic surveys run to the millions of euros

Nikhef

Quick Fix

- Copy files from surfdrive into local Nikhef dcache system
  - Get a backup made
    - avoid any more file losses
  - Use dcache file redundancy
    - keep two copies of each file
    - each copy on a separate machine
    - but still in the same data center
  - Used davfs2 to mount the surfdrive folder
    - rsynced the files to local disc and then into dcache
    - copied 430 GB
    - in around 30 hours

Nikhef

Ongoing backup

- Requirements
  - simple to use, quick to get started
  - total data size expected: a few 10s of TB
  - copy data to Nikhef + an archived backup to tape at Surf
  - copy files into dcache, changes allowed, no file deletions in backup
  - files will not be retained at source (insuffient storage space)
- Quick solution: some sort of rsync equivalent for webdav or xrootd
  - ideally use certificates for authorisation
  - looked at: rclone, rucio, dirac, direct webdav mount, filezilla, custom code, ...
  - no particularly good choice
    - certificate handling was tricky for some
    - but mostly lack of knowledge/learning curve
  - ended up writing a small python client/server to handle the uploading

Nikhef

Client I

- python3 script
- as simple as possible for the user
- requires an x509 certifcate for authentication
- uploads new and changed files from a fixed directory (+ subdirectories)
- config done by system admins
  - set the required directory to upload, server to upload to, location of user certificate
- user runs the script
  - script generates a proxy if the cached proxy has less than 12 hours remaining
  - only user interaction is to enter certificate password to generate a new proxy
  - frequent status updates (every few seconds)
  - let the user see if something is happening (if not make that obvious)

Nikhef

Client II

- upload procedure
  - generate proxy
  - checksum new/changed local files
  - use cached checksums for files with unchanged size and last changed time
  - get checksums of files in dcache from server
  - check local file list against server file list
  - upload local files not in server file list or with a different checksum
  - multiple concurrent uploads for speed

```
andrewp@valen:~/et-emr-data-transfer-client$ ./upload.py
* X509 Proxy: ./upload-proxy - not found
* Create new certificate proxy
* Enter Certificate Password:
* Data dir: ./uploads/data - found
* Meta dir: ./uploads/meta - found
* Local Manifest: ./uploads/meta/manifest-source.pkl - no changes
* Remote Manifest: updated
* Remote Directories: synced
* Manifest File Sync: done
* Meta File Sync: done
* Remote Manifest: updated
* Manifests - local manifest is a subset of remote manifest
* FILE COPY SUCCESSFUL
```

Server

- dcache handles the file uploads directly
- python3 based web server
  - handles collating file checksums for the client script
  - as small and simple as possible
  - two actions
    - generate a file with the checksums of the files already in dcache
    - give the status of any running checksum generation (files processed/files remaining)
  - client script then downloads the checksum file via dcache
    - actually two files: a plain text human readable file and a pickled file of the checksum python objects
- server just responds the client requests
- the client script controls the uploading

Nikhef

Long Term

- Current copying method into dcache only meant as a stop gap solution
  - would like to use something better developed and tested
  - not obvious what to use and what to trust for file integrity
  - and that’s easy for the end user to use
- Files transferred to dcache to be copied into the surf archive storage
  - to be done
  - ongoing periodic backup required
  - files copied into designated directories on surf archive machines are automatically copied onto tape
  - file size is important, small (less than 100MB) need to be packaged together, ideal file size is 1 GB to 200 GB
  - requires some more custom python code to be written

Nikhef

Reality

- As of Friday 12th April
  - no uploads done to dcache after the test upload in early February
- The data is still on surfdrive
  - with a second 500 GB surfdrive folder now in use
- What was asked for and thought needed was a backup
  - what's being used is file sharing
  - which is not backed up
- Learn the lessions
  - provided a service as medium term fix that is not being used
  - user behaviour is saying the file sharing is the immediate desire/requirement
  - backing up not a priority

Nikhef

Questions ?