dm_notify

Informing users about recalls

Peter Edwards | Sr Systems Administrator

December 2013

A twenty year old problem

- When a file is implicitly recalled to disk (ie: just by accidentally accessing an offline file rather than explicitly recalling it using the *dmget* command), the accessing program just hangs without any message until the file is online again, leaving the user wondering if there is a network or host problem.

- Too much essential information is lost by the time the request reaches the DMF daemon via the kernel:
  - The pathname of the file
  - The identity of the accessor
  - The location of the accessor

- This affects SGI DMF, Oracle SAM/QFS and maybe others.

A partial solution

A system daemon written in perl runs on the DMF server and NFS clients, providing messages to users using:

- ssh
- vnc
- torque interactive batch ("*qsub –I*")

CXFS is untested but is expected to work.

But there is no support for SMB/CIFS clients.

Basic method

dmdlog ➔ fhandle ➔ path ➔ fuser ➔ps ➔ pty

1. fhandles are extracted dynamically from the DMF daemon log
2. They are converted to pathnames via an *sqlite* database which is populated from the output of a nightly *dmscanfs*. http://hpsc.csiro.au/users/dmfug/Meeting_Dec2012/Presentations/CSIRO_ASC_DMF_tools.pdf
3. *fuser* finds which processes have the file open
4. *ps* determines the PTY if the process is interactive
5. The message is written directly to that PTY

Optimisation on server

dmdlog ➔ fhandle ➔ path ➔ **/proc** ➔ pty

fuser and *ps* turned out to be too slow on our DMF server when it got busy, so now /proc is being picked through the hard way.

This is twice as fast.

Extension to NFS clients

dmdlog → fhandle → path → **TCP-socket** → fuser → ps → pty

This allows clients to do their own *fuser* and write, based on a pathname passed over the socket by the server.

(We allow NFS mounts only on systems we control and therefore systems that we can run daemons on.)

On the NFS clients *fuser* and *ps* are still used - they don't slow down so much there, and more to the point, *automount* on the client interferes with the /proc idea.

Bells and whistles

It is easy to extract the name of the VG/MG/MSP being used for the recall, and to determine its type (DCM, MAID or tape). This is used to tailor the user message, based on the expected delays.

A simple shell script maintains a file in the users' home directories which is used to give them control over which recalls they wish to be notified about and to query the current setting.

- `dm_notify` Show current setting
- `dm_notify -a` Warn about all recalls
- `dm_notify -m` Warn about medium and slow recalls
- `dm_notify -s` Warn about slow recalls only
- `dm_notify -n` Opt out – never warn about recalls

Sample message

The warnings are classified according to the speed of the back-end media being used and the users' *dm_notify* settings.

```
cherax$  dm_notify
You are currently being warned about slow accidental file recalls only

cherax$  dm_notify –a
You will now be warned about all accidental file recalls

cherax$  ./a.out < q
** File /datastore/somewhere/q
** is being recalled by DMF – it may take several minutes or even longer.
** The oldest currently queued recall request has been waiting for 6m 43s
** Please consider using "dmget" when recalling multiple files.
```

Caveats

- If you try to access an offline file which someone else is already in the process of recalling, you won't receive a warning.
- These warnings take time to prepare; if the system is busy they might not be issued.
- Warnings cannot be given for files which were first migrated after the last database rebuild.
- Users of sudo might not receive warnings.

dmdlog.pm

We now have a general purpose perl module to do all the dmdlog tailing as well as manage the socket link.

The calling script can be quite short. It just says what data it wants from dmdlog (by name) and provides a function to receive it, and then just sits back while the module does all the work, using internal tables which specify which log messages contain the requested items and where.

For daemon requests in particular, it may have to examine multiple messages in order to collect the requested information, which is all provided as a group in a single call to a handler function.

dmdlog.pm example

Something like:

```
    dmdlog::register(\&handle_implicit, 'request', 'krclrea', 'fhdl', 'MSP');
    dmdlog::register(\&handle_implicit, 'request', 'krclvoi', 'fhdl', 'MSP');
    dmdlog::register(\&handle_new_log,  'new_log', '',        'new_file');
    dmdlog::run();      # never returns
...
    sub handle_implicit{
    my ($type, $subtype) = @_[0..1];
    my %values = @_;
...
    sub handle_new_log{
    my ($type, $subtype) = @_[0..1];
    my %values = @_;
...
```

Conclusion

If there is no obvious solution, just keep looking!

Thank you

CSIRO IM&T Scientific Computing

Peter Edwards
Sr Systems Administrator

e peter.edwards@csiro.au
w www.hpsc.csiro.au/