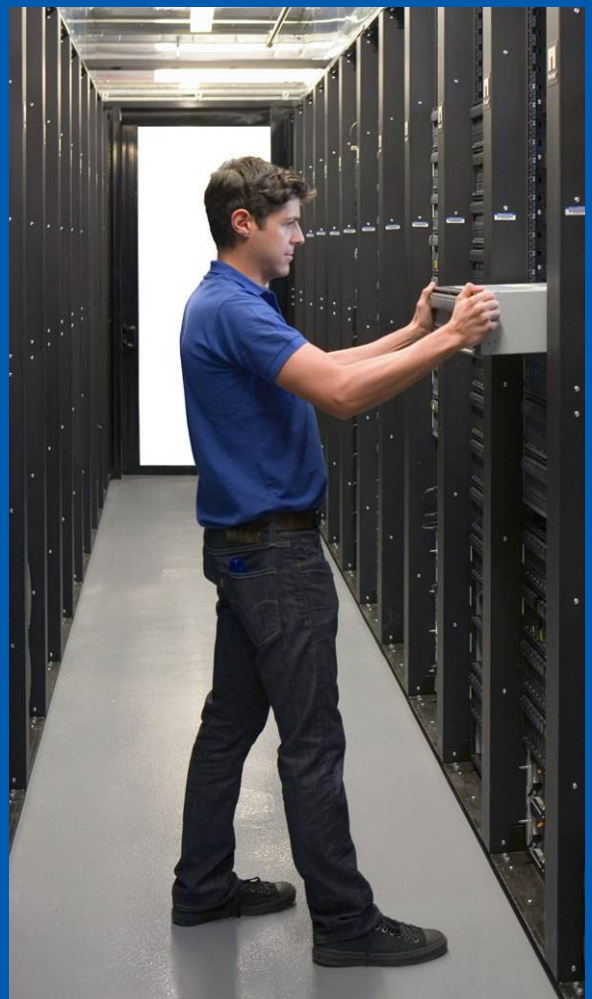
NetApp®

Go further, faster®

File creation speedups for NFSv4.2

Trond Myklebust
<Trond.Myklebust@netapp.com>



Outline

- Unstable file creation
  - Motivation
  - Definitions
  - Implementation details
  - Other issues
- Change attribute issues
  - Problem description
  - Conclusion

Unstable file creation

Motivation

- Speed up task of creating and writing a file.
  - Currently requires at least 2 synchronous disk accesses (one at file creation time, one at WRITE/COMMIT time).
- Speed up attribute changes
  - Why do truncate, permission changes, etc need to be synchronous operations if the client holds a write delegation?
  - NFSv4 is stateful. No longer bound by the stateless constraints that implied all RPC calls must be synchronous.

Definition

- Unstable files are defined as follows:
  - The server is allowed to cache all metadata changes until the client sends a COMMIT.
    - Can cache file creation, directory changes, setattr changes etc.
  - The client is required to recover all cached metadata changes in the case of a server reboot.
    - May be required to create the file entirely from scratch.
    - May be required to replay all SETATTR requests made since the last COMMIT.

Implementation

- Add a new attribute: stable_state
  - Read/write attribute that reflects whether or not the file metadata is synced to disk.
  - Client writes to this attribute at OPEN and SETATTR to signal an unstable request.
    - Server MAY ignore the stable_state
    - Server MUST ignore the stable_state if the client doesn't have a write delegation
      - Required in order to resolve file loss due to an unlink() versus loss due to server reboot.
  - Client may later poll this attribute (as part of post-op GETATTR etc) in order to find out if COMMIT is required.

Other issues

- There may be consequences for other clients after a server reboot
  - Cached filehandles may change if the file needs to be recreated from scratch
    - Server might want to mitigate effects by ensuring it syncs file metadata to disk before replying to READDIR or GETFH from other clients.
    - Note however that write delegation means these clients are not actively caching the file.

Further information

- See the internet draft “draft-myklebust-nfsv42-unstable-file-creation”

Change attribute issues

Problem statement

- When sending multiple GETATTR calls in parallel (e.g. as part of a WRITE compound) the client needs to know which is the most recent value for change attribute
  - Problem is that change attribute is opaque to the client.
    - There is no requirement for updates to be monotonically increasing or to have any other feature that the client can use.
  - Client could use time_metadata, but that is only a recommended attribute.
    - Also is subject to resolution issues

Problem statement (part 2)

- A second problem is that most common NFSv4 server out there (Linux) doesn't have a real change attribute
  - Uses ctime, but with 1sec resolution in most cases.
    - Known spec violation, but there is no alternative without changing the underlying filesystem formats.
- Finally, want to allow the client to do clever things if the server is implementing a true file version counter.
  - Allows for improved cache consistency checking in the absence of a delegation.

Conclusion

- We need a mechanism to impart more information to clients about the change attribute implementation
- Propose the addition of a new per-filesystem attribute “change_attr_type”
  - Bitfield that describes the change attribute, with 5 bits currently defined:
    - change_attr is monotonically increasing
    - change_attr is a version counter
    - change_attr is a version counter except pNFS case
    - change_attr is time_metadata
    - change_attr is undefined (i.e. structureless)

Further information

- See the internet draft “draft-myklebust-nfsv42-unstable-file-creation”

Questions?