NFSv4	T. Myklebust
Internet-Draft	NetApp
Intended status: Standards Track	March 14, 2011
Expires: September 15, 2011	

Enabling server side caching of file creation in NFSv4

Abstract

This document describes an extension to the NFSv4 protocol to allow clients to create and write files with greater efficiency.
The proposed extension allows the server to defer creating the file on stable storage when replying to an OPEN call. The aim is to improve server efficiency and scalability by reducing the number of required disk accesses when writing a file from scratch.

Keywords

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.
Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet- Drafts is at http://datatracker.ietf.org/drafts/current/.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."
This Internet-Draft will expire on September 15, 2011.

Copyright Notice

Table of Contents

*1. Introduction

*2. Definition of the 'stable_state' per-file attribute

*2.1. Use of the 'stable_state' attribute for unstable OPEN requests

*2.1.1. Client use of the 'stable_state' attribute

*2.1.2. Server response upon receiving an unstable OPEN request

*2.1.3. Delegation return and unstable OPEN

*2.1.4. Client unstable OPEN recovery in case of a server reboot

*2.1.5. Directory cache consistency and unstable files

*2.2. Use of the 'stable_state' attribute in unstable SETATTR requests

*2.2.1. Client unstable SETATTR recovery in case of a server reboot

*2.2.2. Delegation return and unstable SETATTR

*3. References

*Author's Address

1. Introduction

One of the remaining sources of performance and scalability issues in the NFSv4.1 protocol [RFC5661], for workloads that require the creation of large numbers of files, is that file creation is still required to be synchronous. This limitation means that the minimum number of disk accesses in a workload that involves creating a file, writing to it and then closing it is 2: one at OPEN time, and one at COMMIT. The following proposal allows the client to indicate to the server, by means of a new attribute, that it is prepared to take on the burden of re-creating the file from scratch if the server should reboot before the file has been fully written. The same attribute also allows the client to check on the state of the file on the server, and thus perhaps to optimise away unnecessary COMMIT requests.

2. Definition of the 'stable_state' per-file attribute

```
const NFS4_UNSTABLE_METADATA = 0x00000001;
const NFS4_UNSTABLE_DATA     = 0x00000002;
const NFS4_UNSTABLE_PNFS     = 0x00000004;
```

Name	Id	Data Type	Acc
stable_state	XX	uint32_t	R W

The attribute 'stable_state' is an optional per-file attribute that can be used by the client to determine whether or not the server believes that all metadata and data has been committed to persistent storage. It is expected that clients may wish to poll it as part of a post-op attribute request or an attribute refresh.

*If the server returns a zero value, then the client may assume that all metadata and data changes that were made since the server last rebooted have been committed to persistent storage.

*If the server sets the bits NFS4_UNSTABLE_METADATA and/or NFS4_UNSTABLE_DATA, then this means that there may be respectively metadata, or data that has not been synced to disk. The client should be prepared to send a COMMIT request in order to ensure persistence of metadata and data.

*If the server sets the bit NFS4_UNSTABLE_PNFS, then this indicates that there are outstanding layouts for write, and thus the state of the file may not be fully known to the server.

A naive server may choose to implement 'stable_state' in terms of a simple flag: it sets NFS4_UNSTABLE_DATA when it receives an unstable WRITE request, sets NFS4_UNSTABLE_METADATA when it receives an unstable OPEN or SETATTR requests and clears both flags when it receives a COMMIT. While such an implementation may not be as useful for avoiding unnecessary COMMIT operations, it is sufficient to support unstable OPEN and SETATTR.

2.1. Use of the 'stable_state' attribute for unstable OPEN requests

We propose a new mode of file creation named "unstable file creation". By choosing this mode of creation, the client is notifying the server that it may defer syncing to disk the new file's directory entry as well as the new file metadata. In case of a server reboot, the client is then responsible for replaying the file creation if the reboot occurred before the file metadata was committed to disk.

2.1.1. Client use of the 'stable_state' attribute

In order to indicate that the client wishes to have the server use unstable file creation, it must set the NFS4_UNSTABLE_METADATA bit in

the optional attribute 'stable_state'. Upon return of the OPEN call, the client then checks that 'stable_state' was indeed set by inspecting the 'attrset' bitmap in the usual way. It can assume that if the 'stable_state' was not set, then the file has been created in persistent storage.
The client MUST NOT set the 'stable_state' to any value other than NFS4_UNSTABLE_METADATA. The server SHOULD return NFS4ERR_INVAL if it receives an invalid value.
Once the client is done making changes to the file, it may use a COMMIT to force the server to flush all data and metadata changes to persistent storage.

2.1.2. Server response upon receiving an unstable OPEN request

Upon receiving an OPEN request that includes a 'stable_state' attribute, the server MAY choose to ignore it, and simply apply the usual NFSv4 rule that all metadata must be committed to persistent storage. If so, it simply omits the 'stable_state' bit from the returned attribute bitmap.
The server MUST NOT set the 'stable_state' flag if the file already exists.
If the server does choose to honour the 'stable_state' attribute, then it MUST also return a write delegation to the client. This write delegation is needed in order to allow the client to detect the recovery edge condition in which a second client attempts to rename the file or delete it just prior to a server reboot.
Once the file has been created in the server cache memory, the server is then free to process the remaining elements of the COMPOUND without syncing the new file metadata to disk.

2.1.3. Delegation return and unstable OPEN

If the client returns the write delegation, then it MUST ensure that the file metadata is in a stable state. It does so by sending a COMMIT operation, unless polling has already established that the 'stable_state' attribute no longer sets the NFS4_UNSTABLE_METADATA bit.

2.1.4. Client unstable OPEN recovery in case of a server reboot

```
enum open_claim_type4 {
        /*
         * Not a reclaim.
         */
        CLAIM_NULL              = 0,

        CLAIM_PREVIOUS          = 1,
        CLAIM_DELEGATE_CUR      = 2,
        CLAIM_DELEGATE_PREV     = 3,

        /*
         * Not a reclaim.
         *
         * Like CLAIM_NULL, but object identified
         * by the current filehandle.
         */
        CLAIM_FH                = 4, /* new to v4.1 */

        /*
         * Like CLAIM_DELEGATE_CUR, but object identified
         * by current filehandle.
         */
        CLAIM_DELEG_CUR_FH      = 5, /* new to v4.1 */

        /*
         * Like CLAIM_DELEGATE_PREV, but object identified
         * by current filehandle.
         */
        CLAIM_DELEG_PREV_FH     = 6, /* new to v4.1 */

        /*
         * Like CLAIM_PREVIOUS, but object identified
         * by directory filehandle + filename.
         */
        CLAIM_PREVIOUS_UNSTABLE = 7
};

union open_claim4 switch (open_claim_type4 claim) {
 /*
  * No special rights to file.
  * Ordinary OPEN of the specified file.
  */
 case CLAIM_NULL:
        /* CURRENT_FH: directory */
        component4      file;

 /*
  * Right to the file established by an
  * open previous to server reboot.  File
```

```
 * identified by filehandle obtained at
 * that time rather than by name.
 */
case CLAIM_PREVIOUS:
       /* CURRENT_FH: file being reclaimed */
       open_delegation_type4   delegate_type;

/*
 * Right to file based on a delegation
 * granted by the server.  File is
 * specified by name.
 */
case CLAIM_DELEGATE_CUR:
       /* CURRENT_FH: directory */
       open_claim_delegate_cur4        delegate_cur_info;

/*
 * Right to file based on a delegation
 * granted to a previous boot instance
 * of the client.  File is specified by name.
 */
case CLAIM_DELEGATE_PREV:
       /* CURRENT_FH: directory */
       component4      file_delegate_prev;

/*
 * Like CLAIM_NULL.  No special rights
 * to file.  Ordinary OPEN of the
 * specified file by current filehandle.
 */
case CLAIM_FH: /* new to v4.1 */
       /* CURRENT_FH: regular file to open */
       void;

/*
 * Like CLAIM_DELEGATE_PREV.  Right to file based on a
 * delegation granted to a previous boot
 * instance of the client.  File is identified by
 * by filehandle.
 */
case CLAIM_DELEG_PREV_FH: /* new to v4.1 */
       /* CURRENT_FH: file being opened */
       void;

/*
 * Like CLAIM_DELEGATE_CUR.  Right to file based on
 * a delegation granted by the server.
 * File is identified by filehandle.
 */
```

```
 case CLAIM_DELEG_CUR_FH: /* new to v4.1 */
         /* CURRENT_FH: file being opened */
         stateid4       oc_delegate_stateid;

 /*
  * Right to the file established by an
  * unstable open previous to server reboot.
  * File is specified by name.
  */
 case CLAIM_PREVIOUS_UNSTABLE: /* new to v4.2 */
        /* CURRENT_FH: directory */
        component4     file_previous_unstable;
};
```

A server that supports unstable file creation SHOULD reject all CREATE and ordinary file creation attempts during the grace period using the error NFS4ERR_GRACE in order to allow clients to recover any unstable files that may have been lost.
In order to recover the file, the client MUST replay the original OPEN that was used to create the file, using an open claim type of CLAIM_PREVIOUS_UNSTABLE.

- *If the server is no longer in the recovery grace period, then it MUST return NFS4ERR_NO_GRACE.

- *If the server discovers that the file already exists, it treats the OPEN as if it were a CLAIM_PREVIOUS request for a write delegation.

- *If the file does not exist, then the server creates the file in the usual fashion and returns a valid write delegation.

Recovery from a NFS4ERR_NO_GRACE error will, as usual, depend on the client's operating environment. However, client implementors should note the existence of the following race condition:

1. Client 1 creates an unstable file, and receives a filehandle which it caches.

2. The server reboots before it has committed the file to disk. The filehandle information, in particular, is lost.

3. A network partition prevents client 1 from recovering the file.

4. After the grace period expires, client 2 creates a file, and the server assigns it the same filehandle as was previously assigned to the unstable file created by client 1.

5. After the network partition is done, client 1 attempts to recover the unstable file, but receives NFS4ERR_NO_GRACE. At this point, client 1 may attempt to access the file via the filehandle, but this now points to the new file created by client 2.

If the server supports the 'time_create', 'time_metadata' or 'time_modify' recommended attributes, then client 1 may use that information to disambiguate the situation above. Should that information not be available, however, the client may have to fall back to attempting an exclusive create for the same file, and returning an error to the application should this fail.

2.1.5. Directory cache consistency and unstable files

While the client that created the file can easily recover in case of a server reboot, it is not necessarily so easy for other clients to do so. While the write delegation does indeed ensure that those clients do not hold the file open (neither do they hold any cached data), it does not guarantee that they are not caching LOOKUP or READDIR data.
In order to avoid issues with directory cache consistency across server reboots, it is therefore RECOMMENDED that servers ensure that initial file metadata be committed to persistent storage prior to replying to a another client's LOOKUP of the new file, or READDIR of the directory in which the new file was created. This will also prevent those clients from seeing filehandles and fileids that might change upon server reboot.

2.2. Use of the 'stable_state' attribute in unstable SETATTR requests

If it holds a write delegation, the client may also use the 'stable_state' attribute in a SETATTR request to indicate to the server that it is ready to replay this SETATTR in the case of a server reboot. The procedure is the same as for OPEN. In order to indicate to the server that it wants the SETATTR request to be unstable, the client sets the 'stable_state' attribute to the value NFS4_UNSTABLE_METADATA. Again, the server MAY ignore the 'stable_state' attribute, in which case it MUST immediately commit the attributes to stable storage, and MUST clear the 'stable_state' bit in the returned attribute bitmap.
If the client does not hold a valid write delegation, then the server MUST also ignore the 'stable_state' attribute.

2.2.1. Client unstable SETATTR recovery in case of a server reboot

If the server reboots before the client has had a chance to issue a COMMIT, then after recovering the write delegation, the client SHOULD check the server attributes against its own cached values. If there is a mismatch, then it is responsible for correcting this by replaying the relevant SETATTR calls.

2.2.2. Delegation return and unstable SETATTR

If the client returns the write delegation, then it MUST ensure that the file metadata is in a stable state. It does so by sending a COMMIT operation, unless polling has already established that the 'stable_state' attribute no longer sets the NFS4_UNSTABLE_METADATA bit.

3. References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels ", RFC 2119, .
[RFC5661]	Shepler, S., Eisler, M. and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol ", RFC 5661, .

Author's Address

Trond Myklebust Myklebust NetApp 3215 Bellflower Ct Ann Arbor, MI 48103 USA Phone: +1-734-662-6608 EMail: Trond.Myklebust@netapp.com