NFSv4 Working Group
Internet-Draft
Intended status: draft
Expires: April 18, 2010

S. Faibish
EMC Corporation
D. Black
EMC Corporation
M. Eisler
NetApp

October 18, 2009

pNFS Access Permissions Check
draft-faibish-nfsv4-pnfs-access-permissions-check-00

Status of this Memo

Copyright Notice

Abstract

This document describe an extension to pNFS protocol addressing a gap related to the access permission checks to data servers used by the MDS in layouts sent to the clients. The draft addresses both the client access permission checks as well as the MDS access permissions to the data servers. The draft will address new errors related to access permission denial to devices included in valid pNFS layouts. The draft will also address the case when clients request direct NFS access to the MDS and the MDS has no permission to access some of the data servers included in valid layouts.

Table of Contents

1. Introduction

Figure 1 shows the overall architecture of a Parallel NFS (pNFS) system:

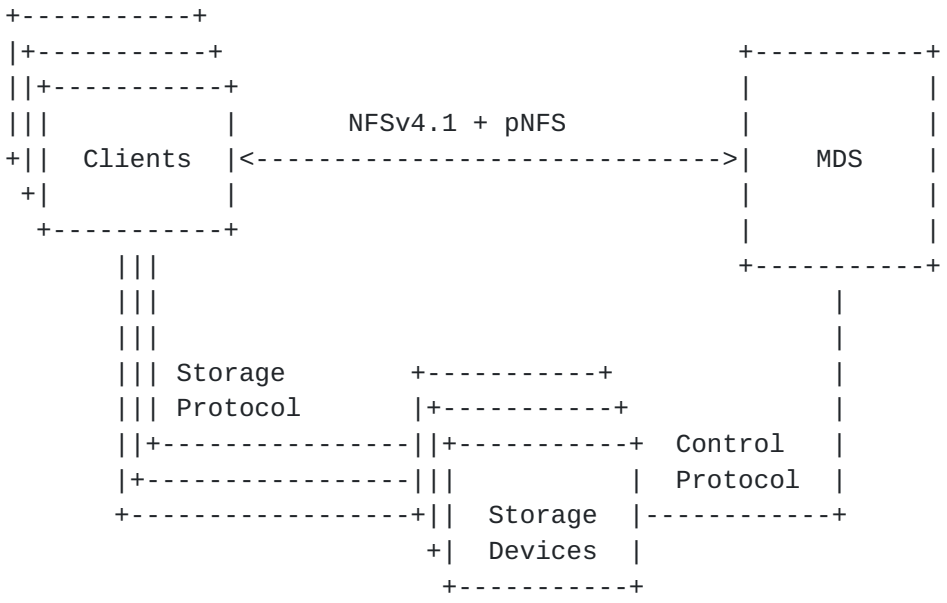

Figure 1 pNFS Architecture

There is a possible gap in the pNFS protocol regarding permissions of access to storage devices in the cases of a client that has no permission to access a storage device (SD) included in a valid layout sent by the MDS server. Some consider this gap an optimization or an implementation detail but the permission denials can defeat the performance scalability value of pNFS and to possible opportunities of unreported errors. From the pNFS protocol perspective there is no error mechanism to inform a system administrator that a client doesn't have the access permission to a storage device at mount time nor at I/O time. This is also the case with the MDS that doesn't have access permission to some storage devices and it is asked by a client to perform I/O to the device on behalf of the client. In this document storage devices mean also data servers and storage severs which could refer to file, block or object storage.

On one hand looking at the block layout if the MDS doesn't see the storage devices/LUNs it cannot mount the pNFS file system, of course it cannot allow a client to mount that FSID and an error is logged by the MDS server. If the pNFS block server can access all the storage devices/LUNs but the client doesn't have the access permission to some storage devices/LUNs at mount time the client will mount as

NFSV4.1 without pNFS support (fallback to NFS) without any

error/reason for the fall back. If the client doesn't have access permission to all the storage devices it will log an error at the client without any explanation of the reason of mounting NFSV4.1 without pNFS support.

This is true for file and object layout pNFS clients regardless if the MDS has permission to access the storage devices or not. On the other hand, for the file and object layouts there is no similar error mechanism to report the case when the client or the server cannot access a storage device and there is no CB for access permission check. The only fallback is a request for re-direct by the MDS server as storage device is inaccessible assuming that the MDS server has access to the storage device and it can serve the I/O to the client still without logging an error at least not at mount time. This assumption is weaker than in the case of the block layout that cannot allow to mount a FSID to which it has no access permission.

1.1. Example

A typical usecase is when a new storage device is added and all the pNFS clients (1000s of them) have no access permission to the new storage device. From this time on all the I/Os to the new storage device will be served by the MDS server creating a performance and scalability bottleneck that is difficult to detect.

A better approach to address this issue is to report the access failure before the client attempt to issue any I/Os to the MDS server rather than the MDS trying to diagnose the performance problem caused by client I/O using NFS path and not using the pNFS layout. In the current pNFS protocol a client cannot detect this situation at mount time in cases of complex mountpoint structures and we can perhaps only address the error for the root/top of the mount structure assuming we are only referring to pNFS capable clients. See section 1.2.1 for detailed example.

The intention of this draft is to introduce a new access permission check and error access permission denial report mechanism at both server and client to address the above issues.

One of the problems may be the fact that there is no mention in the pNFS spec to address the data protocol between MDS and storage devices, except for the block layout driver in which case the MDS cannot itself mount a pNFS file system due to access permission issue. In order for the MDS server to export a filesystem as NFSV4.1 filesystem for pNFS clients access it is mandatory for the MDS to have access permission to all the storage devices/LUNs for that filesystem as a pre-condition for the mount. In the case that there is any access permission issue the filesystem cannot be mounted by

the MDS and an error is sent to the MDS server log.

On the other hand for file and object pNFS layout MDS servers there is no requirement in the spec to check access permission to all the storage devices even when the NFSV4.1 filesystem is exported to the pNFS clients. In fact an MDS that accesses the storage devices is considered an unhealthy pNFS server except for the case when a pNFS client fall back to NFS and requests the MDS server to perform an I/O on his behalf. At that time the MDS must access the storage server in order to perform the I/O. It is then possible that the MDS I/O to the storage device fails due to access permission denial in which case the MDS will send a error to the client and the client I/O fails.

There is no error report mechanism in the pNFS protocol for this type of error. Even if we correct the access permission issue the introduction of a new error reporting mechanism at I/O time for both server and client can be problematic as it may be too chatty. We propose to introduce a new error case but leave the error reporting mechanism at I/O time OPTIONAL or an optimization to the latitude of the server and client implementation.

Although the change to the protocol is delicate logging some kind of warning at the client might be appropriate to be recommended as an implementation option on the client to reduce chattiness.

1.2. Issues with the current pNFS protocol

Scenario of Interest: Client expects to be able to use pNFS (e.g., use -pnfs switch to mount command, or similar), but one or more devices are inaccessible. This discussion does not apply to a client that doesn't care (e.g., uses pNFS to optimize if available, but is ok if all of its access is via the main NFS server).

Desired client behavior: Client gets entire device list for mount point from server and checks it as part of the mount operation (or at whatever point it first realizes that it expects to use pNFS).

Missing piece of protocol: Client has no obvious way to report an inaccessible device to the server.

1.2.1. Client access permission denial to SD

A client doesn't communicate to the MDS server that the client's access to a storage device is denied as a result of an access permission issue. When the pNFS server grants a layout to the client, it assumes the client can access the storage devices (files, luns, or objects). The server cannot check this because the server cannot issue I/Os via the client and because connectivity is not transitive - the client may have good network connectivity to the MDS, the MDS may have good storage connectivity to the storage devices, but something in the storage network prevents the client from talking to one or more of the storage devices. This could be a network mis-configuration or failure, and it's a possible scenario for all pNFS layout types.

The access permission problem cannot be reported at mount time for a number of reasons. Reporting the permission problem at mount time has some difficulties. First, the MDS pNFS server doesn't know that the client can even mount with pNFS support. Second, the MDS NFS server doesn't know that the client is mounting the NFS filesystem (there is no separate mount protocol in NFSv4). Third, the MDS server cannot know if the client mounts say, "/", and the file systems below "/" have pNFS capabilities, but refer to different storage devices. Or the client mounts say "/a/b/c/d", and d is in a pNFS capable volume. But the client is going actually do its I/O to "e/f/g/h/i/j/k", and k is either no pNFS capable, or it is, but uses a storage device that differs from d.

1.2.2. MDS access permission denial to SD

The current pNFS server protocol doesn't mandatory require to access the storage devices and there is only a control protocol (Fig. 1) between the MDS and the storage devices but there is no specific data access protocol between the MDS and the SDs. Although the MDS doesn't check permissions it is assumed that at the configuration is correct when the storage devices are initially configured and the pNFS filesystem is mounted on the MDS server. It is possible that the administrator checks the MDS access permission to all the SD during the configuration. The problem may not exist at the time of the initial mount of the pNFS filesystem but can surface when a new SD is added to the pool of SDs. If the MDS tries to do successful I/Os to the new added SD before including it in the layout to pNFS clients will avoid this set of problems. The pNFS specification does not address the data access protocol between the MDS and the storage devices.

1.2.3. Implied Requirement

Metadata server SHOULD NOT use devices in pNFS layouts that are not accessible to the MDS (or to clients if the MDS has any means of determining this).

2. Conventions used in this document

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 RFC-2119 [RFC2119].

3. Description of the proposed approaches to solution

A simple possible approach is to address the gap in the protocol by simply adding a new LAYOUTRETURN type and a new error case to LAYOUTRETURN. In the case that the pNFS client has a valid layout on a file but cannot perform I/O to a SD due to access permission denial, the client will fall back the I/O to the MDS NFS server. Before the client sends the I/O to the NFS server it will send a LAYOUTRETURN command for the purpose of avoiding unnecessary MDS CB_LAYOUTRECALL operations in the future. The client will send the LAYOUTRETURN operation for the layouts corresponding to the inaccessible SD and include a new error reporting that the reason of the fall back to the NFS server is access permission denial to the specific deviceid4. The client may return disjoint regions of the file by using multiple LAYOUTRETURN operations within a single COMPOUND operation. The client will include NFS4ERR_DEVICE_PERM_DENY in the new LAYOUTRETURN operation.

LAYOUTRETURN at FSID scope seems like the best simple choice available. Alternatively we can introduce a new LAYOUTRETURN type that is LAYOUT4_RET_REC_FSID_NO_ACCESS, i.e., return all layouts for this FSID and tell the server that the reason for the return is a connectivity issue. In order to differentiate the permission issue from a real connectivity issue the solution will require the client to do two LAYOUTRETURN operations to deal with servers that don't understand the new type. The two LAYOUTRETURN operations happen once per client using LAYOUT4_RET_REC_FSID_NO_ACCESS and only in an error case followed by a second operation for "FSID" in case the first one wasn't understood.

3.1. Simple implementation adding new LAYOUTRETURN error code

3.1.1. ARGUMENT

When the LAYOUTRETURN operation specifies a LAYOUTRETURN4_FILE_return type, then the layoutreturn_file4 data structure specifies the region of the file layout that is no longer needed by the client. We will modify the layoutreturn_file4 changing the opaque "lrf_body" field of the "layoutreturn_file4" data structure to include the deviceid with access permission error. Alternative, more complex, add the deviceid to the layoutreturn_type4.

```
struct layoutreturn_file4 {
        offset4         lrf_offset;
        length4         lrf_length;
        stateid4        lrf_stateid;
        deviceid4       lrf_deviceid;
        /* layouttype4 specific data */
        opaque          lrf_body<deviceid>;
};
```

3.1.2. RESULT

```
union LAYOUTRETURN4res switch (nfsstat4 lorr_status) {
case NFS4_OK:
        layoutreturn_stateid     lorr_stateid;
case NFS4ERR_DEVICE_PERM_DENY:
        layoutreturn_deviceid    lorr_deviceid;
default:
        void;
};
```

3.1.3. Description

This solution will add a new error case to LAYOUTRETURN. The implementation will use LAYOUTRETURN when FSID is sent to the client. When the client fails an I/O as a result of access permission denial it will send a LAYOUTRETURN operation to the MDS server with new error NFS4ERR_DEVICE_PERM_DENY specifying the deviceid4 with permission denial.

When the server receives this error it can OPTIONALLY log an error to the syslog and perform a access performance check to the SD expecting that the client will fall back the I/O to the MDS. If the permission check of the server fails the NFS4ERR_DEVICE_PERM_DENY will be sent to the syslog.

3.2. Implementation using a new layoutreturn_type4

In this section we will define the usecase addressed by this implementation.

3.2.1. ARGUMENT

```
/* Constants used for new LAYOUTRETURN and CB_LAYOUTRECALL */
const LAYOUT4_RET_REC_FILE      = 1;
const LAYOUT4_RET_REC_FSID      = 2;
const LAYOUT4_RET_REC_ALL       = 3;
const LAYOUT4_RET_REC_DEVICE    = 4;


enum layoutreturn_type4 {
      LAYOUTRETURN4_DEVICE = LAYOUT4_RET_REC_DEVICE_NO_ACCESS,
      LAYOUTRETURN4_FILE = LAYOUT4_RET_REC_FILE,
      LAYOUTRETURN4_FSID = LAYOUT4_RET_REC_FSID,
      LAYOUTRETURN4_ALL  = LAYOUT4_RET_REC_ALL
};


struct layoutreturn_device4 {
      offset4         lrf_offset;
      length4         lrf_length;
      stateid4        lrf_stateid;
     deviceid4    lrf_deviceid;
      /* layouttype4 specific data */
      opaque          lrf_body<>;
};


union layoutreturn4 switch(layoutreturn_type4 lr_returntype) {
      case LAYOUTRETURN4_DEVICE:
              layoutreturn_device4      lr_layout;
      default:
              void;
};
```

3.2.2. RESULT

```
union LAYOUTRETURN4res switch (nfsstat4 lorr_status) {
case NFS4_OK:
      layoutreturn_stateid     lorr_stateid;
case NFS4ERR_DEVICE_PERM_DENY:
      layoutreturn_deviceid    lorr_deviceid;
default:
      void;
};
```

3.2.3. New LAYOUTRETURN type description

We will use a new LAYOUTRETURN layoutreturn_type4, let's call it LAYOUT4_RET_REC_DEVICE_NO_ACCESS, in which case the client returns all layouts for this DEVICE and OPTIONAL for the FSID and tell the server that the reason for the return is a connectivity issue. The same stateid may be used or in order to report a new error client will force a new stateid. We will also add the operation to report a new error NFS4ERR_DEVICE_PERM_DENY.

To address the backward compatibility may require a client to do two layout return operations to deal with servers that don't understand the new layoutreturn_type4. If the server doesn't understand the new layoutreturn_type4, then the server will come back with an error code. The client needs to do a FSID return and remember that this server doesn't understand the new return type. This assumes that the client is sufficient disrupted by the connectivity problem to the point it decided to drop all layouts for the filesystem (FSID), which matches the failure case of client data server access permission deny. Alternatively when the server receives a new stateid it will check the error or issue an CB_LAYOUTRECALL to get the error.

4. Reporting the permission denial

4.1. Permission denied to client at mount time

The most suitable time for the client reporting the permission denial by a data server is at the mount time. This would be the preferred way to address the issue but it is not possible with the current protocol for several reasons: If the server initiates the request, MDS doesn't know if the client wants to use pNFS or NFS. If the client is the initiator of the error the is mounting the pNFS filesystem knowing that it will use pNFS for access the client doesn't specifically request pNFS.

The solution will be to use a special tag -pnfs or a switch to mount/syscall. To the latest issue the client cannot explicitly request pNFS as it needs first to discover that the server is supporting pNFS. In order to address this issue the client needs to send a request at mount time to the server as part of the initial handshake. There is no reportable error of the client to cope with this currently.

The client makes a file access and it finds that the NFS server is pNFS capable it will request a LAYOUTGET command and if the NFS server doesn't accept and returns an error the client will request access using plain NFS. The client will decide if this is an error or not. In the case that the LAYOUTGET command succeeded the client may still ask the MDS to deliver the I/O. So, inherently the client has to query the MDS access permissions to all the DS that are used in the layout send to the client before putting the device into a layout. The pNFS protocol doesn't require the MDS to check access permission to the devices that are included in the layout. It is assumed that the MDS has permission access to all the devices it includes in the layout without any checks.

If the MDS doesn't know if it has access or not it shouldn't put that device in the layout granted to clients to prevent cases when the client ask the I/O using plain NFS from the MDS. If the MDS doesn't have permission access to a data server it will send an error to the client and the I/O will fail. Based on the above behavior the best time to check is at the time when the initial configuration of the pNFS filesystem is done. Currently the pNFS spec states that a client can write through or read from the MDS, whether it has a layout or not or it does not support pNFS assuming that the MDS has permission access to all the data servers. We propose to make this implicit recommendation explicit.

4.2. Permission denied to the client at I/O time

In this case when the pNFS capable client receives a valid layout from the pNFS capable MDS server and due to access permission denial to some devices cannot write to the storage devices, it will fall back to the NFS server for the I/O. There is no error logged by the client nor sent back to the MDS server mentioning the reason for the fallback. As a result there is no way to fix the configuration problem until the client unmounts the pNFS filesystem. And potentially if there is no permission check at mount time even the remount will not detect the problem. Moreover as the MDS server never checks access permission to the storage devices the MDS will not be able to perform the I/O unless the MDS is also a storage device itself, in which case the I/O will fail without any error mentioning permission denial. One option is for the MDS to send a LAYOUTRETURN

with FSID_PERM_CHECK in the case when the a pNFS client request the MDS to write an I/O to one of the devices from a layout sent to the

client by the MDS the MDS will check the error and send a CB request for FSID_PERM_CHECK.

4.3. Permission denied to MDS server at I/O time

In case when the client holding a valid layout requests the NFS server to execute the I/O the MDS will have to access the data server/device that the client requested to write to and gets an access permission denial from the storage device, the MDS cannot perform the I/O and will return an error to the client. In this case the client I/O will fail indefinitely and there no error information about the reason of the failure related to permission denial to data servers. The client has no means to communicate to the server the permission denial as there is no check and error case. To address this case a new error code will be added to the LAYOUTRETURN call mentioning DEVICE_PERM_DENY and the MDS will send an error to the client NFS4ERR_PERM_DENY. An additional option is to send a CB to the client requesting permission access check and on failure the MDS will log an error NFS4ERR_DEVICE_UNACCESSIBLE to inform the admin to correct the problem. On receiving the permission check the client will send the DS a GETDEVICEINFO and report NFS4ERR_DEVICE_PERM_DENY to the MDS server.

5. Security Considerations

All control operations from the MDS to the storage devices, including any operations required for access permission checks in order to detect permission denials to the MDS and the pNFS client, should be authenticated in order to address cases when the access permission is denied to the client by the administrator. It is expected that the permission denial to a certain data server to a certain client will be known to the MDS by configuration. This will be implemented for all the pNFS layout types.

6. IANA Considerations

There are no IANA considerations in this document beyond pNFS IANA Considerations are covered in [NFSV4.1].

7. Conclusions

This draft specifies additions to the pNFS protocol addressing access permission checks of the client and MDS server to storage devices used in pNFS layouts for all layout types.

8. References

8.1. Normative References

[LEGAL] IETF Trust, "Legal Provisions Relating to IETF Documents",URL http://trustee.ietf.org/docs/IETF-Trust-License-Policy.pdf, November 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[NFSV4.1] Shepler, S., Eisler, M., and Noveck, D. ed., "NFSv4 Minor Version 1", RFC [[RFC Editor: please insert NFSv4 Minor Version 1 RFC number]], [[RFC Editor: please insert NFSv4 Minor Version 1 RFC month]] [[RFC Editor: please insert NFSv4 Minor Version 1 year]]. <http://www.ietf.org/rfc/rfc[[RFC Editor: please insert NFSv4 Minor Version 1 RFC number]].txt>.

[draft-ietf-nfsv4-pnfs-block-12] Black, D., Glasgow, J., Fridella, S., "pNFS Block/Volume Layout".

[XDR] Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

8.2. Informative References

[MPFS] EMC Corporation, "EMC Celerra Multi-Path File System", EMC Data Sheet, available at: http://www.emc.com/collateral/software/data-sheet/h2006-celerra-mpfs-mpfsi.pdf link checked 16 October 2009

9. Acknowledgments

This draft includes ideas from discussions with the authors of the different pNFS layouts Jason Glasgow and Benny Halevy as well as pNFS maintainer of Linux kernel including Bruce Fields.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Sorin Faibish (editor)
EMC Corporation
32 Coslin Drive
Southboro, MA 01772
US

Phone: +1 (508) 305-8545
Email: sfaibish@emc.com

David L. Black
EMC Corporation
176 South Street
Hopkinton, MA 01748
US

Phone: +1 (508) 293-7953
Email: black_david@emc.com

Michael Eisler
NetApp
5765 Chase Point Circle
Colorado Springs, CO 80919
US

Phone: +1 (719) 599 8759
Email: mike@eisler.com