NFSv4 Working Group
Internet-Draft
Intended status: draft
Expires: September 3 2010

S. Faibish
EMC Corporation
D. Black
EMC Corporation
M. Eisler
NetApp
J. Glasgow
Google

March 5, 2010

pNFS Access Permissions Check
draft-faibish-nfsv4-pnfs-access-permissions-check-02

Status of this Memo

Copyright Notice

Abstract

This document extends the pNFS protocol to communicate the results of permission checks for access to the data servers referenced by layouts, including checks performed by both clients and the MDS. The extension provides means for clients to communicate client-detected access denial errors to the MDS, including the case in which a client requests direct NFS access via the MDS that the MDS cannot perform.

Table of Contents

1. Introduction

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

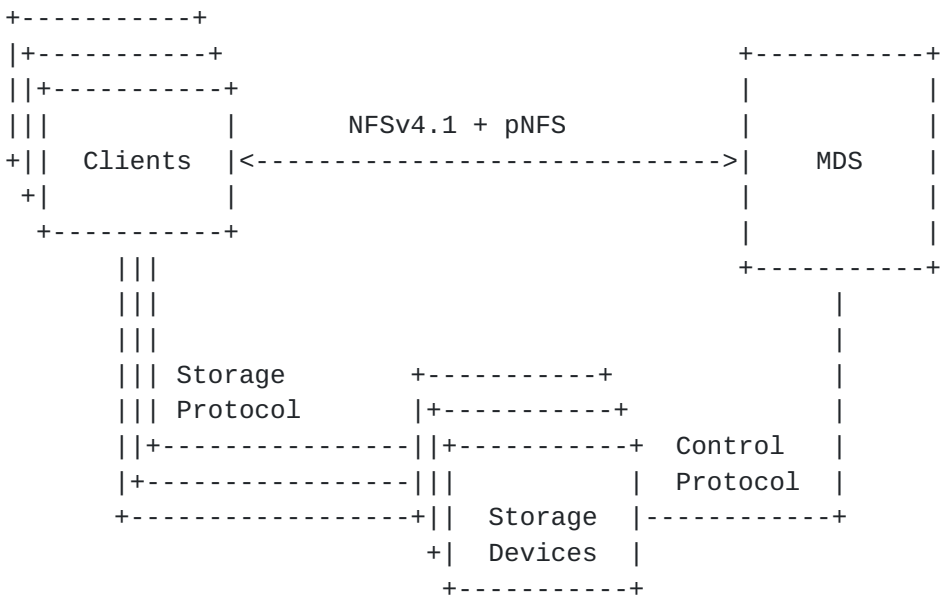

Figure 1 pNFS Architecture

Inconsistent access permissions expose a gap in the pNFS protocol. The pNFS protocol assumes that a client can access every storage device (SD) included in a valid layout sent by the MDS server, and provides no means to communicate client access failure to the MDS. It has been argued that this is an implementation detail, but access failures permission denials can impair the performance scalability value of pNFS and allow errors to go unreported. There is no pNFS error mechanism to inform a system administrator that a client lacks permission to access a storage device at mount time or when I/O is performed. There is a related problem when an MDS doesn't have access permission to some storage devices and hence cannot perform I/O on behalf of a client. In this document storage devices are a generic term for data servers and/or storage servers used by the file, block and object pNFS layouts.

In the case of the block layout [RFC5663] if the MDS has no access to a storage device (LUN) implementations are generally unable to export the NFS mount point for any filesystem using that storage device. In this situation, clients will be unable to mount that file system and an error will presumably be logged by the MDS server. If the MDS can access all the storage devices involved, but the client doesn't have sufficient access to some storage devices/LUNs, at mount time the client may choose to mount the file system using NFSV4.1 without pNFS support (fallback to NFS). This failure to mount as a pNFS file system cannot currently be communicated to the server because there are no protocol messages defined which convey this failure.

The above discussion also applies to the file and object layout pNFS clients regardless of whether the MDS has permissions to access the storage devices, with one important difference. In contrast to the block layout, MDSs for the file and object layouts are often unable to access the storage devices that store data for the exported filesystem. This make it significantly easier for a file or object layout MDS to provide layouts that contain inaccessible devices, in contrast to the block layout where an MDS should not allow a client to mount a FSID to which the MDS has no access permission.

There is no error reporting 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 as an implementation option on the client to reduce chattiness.

1.1. Example

A motivating use case is addition of a new storage device to which all the pNFS clients (1000s of them) lack access permission. Layouts cannot be granted that use this new device, requiring that all I/Os to that new storage device be served by the MDS server creating a performance and scalability bottleneck that may be difficult to detect based on I/O behavior.

A better approach to this issue is to report the access failure before the client attempts to issue any I/Os that can only be

serviced by the MDS server. This makes the problem explicit, rather than the forcing the MDS, or a system administrator to diagnose the performance problem caused by client I/O using NFS instead of the pNFS layout. There are limits to this approach because complex mount structures may prevent a client from detecting this situation at mount time, but at a minimum, access problems involving the root of the mount structure can be detected. See section 1.2.1 for a detailed example.

This document adds error reporting mechanisms to address both this situation and situations in which the client cannot detect the access problem until it attempts to perform I/O to the inaccessible storage device.

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 storage devices are inaccessible. This discussion does not apply to a client that doesn't care whether pNFS is used (e.g., uses pNFS to optimize if available, but for which it is acceptable that access is performed via the main NFS server).

Desired client behavior: Client gets the entire storage device list for a 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 protocol functionality: Client has no obvious way to report an inaccessible storage 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 prevents the client from talking to one or more of the storage devices. This could be a network mis-configuration or failure, and is a possible scenario for all pNFS layout types.

This access permission problem cannot be reported by the MDS server when the client mounts the filesystem for several reasons. First, the

MDS pNFS server doesn't know whether the client supports or intends to use pNFS. Second, the MDS NFS server doesn't know that the client is mounting the NFS filesystem (there is no explicit mount operation in NFSv4). Third, it is unreasonable to expect the MDS to know and check the entire mount structure below the mount point used by the client. For example, if the client mounts "/", the file systems below "/" may have pNFS capabilities, but refer to different storage devices. Or the client may mount say "/a/b/c/d", where "d" uses a pNFS capable storage device, but the client subsequently does I/O to "e/f/g/h/i/j/k", where "k" is either not pNFS capable or uses a storage device different from the storage device used by "d".

1.2.2. MDS access permission denial to SD

The current pNFS server protocol doesn't require MDS data access to the storage devices. Although the MDS is not required to check permissions, it is assumed that the devices are correctly configured when the pNFS filesystem is initialized on the MDS server and exported. Even if the administrator checks the MDS access permission to all storage devices during initial configuration, the problem may surface at a later point in time when a new storage device is added or other changes are made. For the specific case of adding a new storage device, an MDS check of I/Os to the newly added device before using it in layouts avoids this set of problems, but this does not cover loss of MDS access to existing storage devices.

1.2.3. New MDS Requirement

The metadata server (MDS) SHOULD NOT use storage devices in pNFS layouts that are not accessible to the MDS. To the extent that an MDS can determine whether storage devices are accessible to clients, if a client cannot access a storage device, an MDS SHOULD NOT include that storage device in a pNFS layouts sent to that client.

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

There are several possible solutions. The first is to implement a new operation, LAYOUTRETURN4x that returns layouts to the MDS along with error information. Clients that receive an NFS4ERR_NOTSUPP error SHOULD mark the server as not supporting this operation and use LAYOUTRETURN instead.

Another possible approach is to make use of the opaque field available in LAYOUTRETURN. One could define part of this field for all layout types. In the case that the pNFS client has a valid layout on a file but cannot perform I/O to a SD due to lack of access permission, 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 sends a LAYOUTRETURN command for the purpose of avoiding unnecessary MDS CB_LAYOUTRECALL operations in the future. The client sends the LAYOUTRETURN operation for every layouts that uses to the inaccessible SD and includes an error reporting that the reason for the fall back to the NFS server is an 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.

A third approach is to introduce a new LAYOUTRETURN type at FSID scope such as 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 the existing FSID scope to interoperate with an MDS that doesn't understand the new scope.

3.1. Defining the opaque fields of LAYOUTRETURN

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. For each layout type we define the opaque lrf_body so that it can communicate an error code to the server as well as the deviceid4 which encountered the error. This has already been defined for the object layout type [RFC5664]

For the file layout we define the opaque body as follows:

```
struct nfsv4_1_file_layoutreturn4 {
        deviceid4       lrf_deviceid;
        nfsstat4        lrf_status;
};
```

An MDS server should check the length of the lrf_body. If the length is zero, then the client has not communicated additional information with the layout return. This will generally be the case when a file is closed, or in response to a CB_LAYOUTRECALL operation.

For the block layout type, we similarly define the block specific structure as:

```
struct pnfs_block_layoutreturn4 {
        deviceid4      lrf_deviceid;
        nfsstat4       lrf_status;
};
```

The alternative, which is more complex is to make the status (error) and deviceid4 common to all LAYOUTRETURN operations, but do so by adding a new operation or a new return type(LAYOUT4_RET_REC_FILE_ERROR)

```
struct layoutreturn_file_error4 {
        offset4         lrf_offset;
        length4         lrf_length;
        stateid4        lrf_stateid;
        deviceid4       lrf_deviceid;
        nfsstat4        lrf_status;
        /* layouttype4 specific data */
        opaque          lrf_body<>;
};
```

3.1.2. RESULT

The LAYOUTRETURN4res remains unchanged.

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 and the deviceid4 on which access permission was denied.

When the server receives this error it MAY log an error to the syslog and perform an access permission 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 SHOULD be logged.

3.2. Implementation using a new layoutreturn_type4

In this section we will define the use case 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_FSID_NO_ACCESS    = 4;

enum layoutreturn_type4 {
      LAYOUTRETURN4_DEVICE = LAYOUT4_RET_REC_FSID_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;
      nfsstat4        lrf_status;
      /* 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;
     default:
           void;
     };
```

3.2.3. New LAYOUTRETURN type description

We will use a new LAYOUTRETURN layoutreturn_type4, let's call it LAYOUT4_RET_REC_FSID_NO_ACCESS, in which case the client returns all layouts for this FSID and informs the server that the reason for the return is an inability to access the device. 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 mechanism to report a new error NFS4ERR_DEVICE_PERM_DENY.

Backwards 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 respond with an error code. The client SHOULD do an ordinary FSID return and remember that the new return type is not to be used with this server. This assumes that the client is sufficiently disrupted by the problem to decide to drop all layouts for the filesystem (FSID). Alternatively, for servers that understand the new layoutreturn when the server receives a new stateid it will check if there is an NFS4ERR_DEVICE_PERM_DENY error or issue an CB_LAYOUTRECALL to get the error code from the client.

3.3. Operation: CB_LAYOUTACCESSCHECKRECALL - Ask client to check permissions

3.3.1. ARGUMENT

```
/*

 * NFSv4.1 callback arguments and results

/*

   struct CB_LAYOUTACCESSCHECK4args {
           nfs_fh4                 claca_fh;
           offset4                 claca_offsets[];
   };
```

3.3.2. RESULT

```
struct layoutaccesscheck_device4 {
        deviceid4 lac_device_id;
        nfsstat4  lac_status;
};


struct CB_LAYOUTACCESSCHECK4res {
        layoutaccesscheck_device4 clacr_status[];
};
```

3.3.3. DESCRIPTION

In this case the client checks that it has permission access to all the deviceid that are included in all the layouts in his possession and report to the MDS deviceid with permission access denial. Using this operation the MDS will find out what are the SDs that have permission access issues for more than one client that have valid layouts to that device and didn't yet found that there is a permission access issue. In this case the MDS can prevent the client from falling back to NFS by recalling the layout and removing the faulty device from the layout thus preventing a storm of I/Os to the MDS. The MDS will only send a CB_LAYOUTACCESSCHECK command to clients that already have a valid layout for the faulty device. As an implementation recommendation the MDS will remove that device from the valid devices list and will log an error mentioning that there is a problem with that device. All the layouts delivered to new client requests will exclude the device with the problem. Some servers may chose to perform the I/O via the MDS with the risk of a retry and I/O error of the MDS. In this latest case the MDS will unilaterally remove that device from the list and will recall all the layouts from all the clients that have layouts to that device and send new layouts excluding the faulty device.

4. Reporting storage device inaccessibility

4.1. Access denied to client at mount time

The most suitable time for the client reporting access 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 client is mounting the pNFS filesystem

knowing that it will use pNFS for access but the client doesn't specifically request pNFS.

The solution requires a special tag -pnfs or a switch to the mount command and syscall at the client. The client cannot explicitly request pNFS as it needs first to discover that the server is supporting pNFS by sending a pNFS LAYOUTGET request to the server at mount time. If this request fails, the resulting error cannot be reported by the client.

The client will send an OPEN request to access a file and to find if the NFS server is pNFS capable it will send 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 for access permissions to all the SDs that are used in the layout sent to the client before accessing the deviceid included in the 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 to a deviceid it shouldn't put that device in the layout granted to clients in order to prevent cases when the client sends the I/O using plain NFS from the MDS. If the MDS doesn't have permission access to a SD 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 SDs. 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 MDS server and cannot write to the storage devices, the client falls back to the NFS server to perform the I/O. There is no error currently logged by the client or sent back to the MDS server in this situation. The client will use the new error case added in section 3.1 or will use a LAYOUTRETURN including NFS4ERR_DEVICE_PERM_DENY error code as defined in section 3.2. If the client didn't access the SD that has the permission denial yet and it is not aware of such an issue the client couldn't send an error to the MDS. But if the MDS got a permission error for a deviceid from

another client it can send a CB_LAYOUTRECALL with FSID_PERM_CHECK to the client in the case when a pNFS client requests the MDS to write an I/O to one of the devices from a layout sent to the client by the MDS before. The client will send a LAYOUTRETURN and the MDS will check that the error is NFS4ERR_DEVICE_PERM_DENY and to confirm that this is a permission access issue not a connectivity or other error.

4.2.1. pNFS client detects permission access denial

The current protocol recommends that the client fallback to the NFSv4.1 server to do the I/O on the behalf of the client and in same compound command it includes a LAYOUTRETURN command for the layout part on the Storage Device with permission issues. The recommendation can be interpreted as LAYOUTRETURN of all the layouts for that file. According to the section 3.2 in this case the client will issue a LAYOUTRETURN mandatory for the layout offset in the range that resides on the deviceid with permission access denial together with the fallback of I/O to NFS. An error code NFS4ERR_DEVICE_PERM_DENY will be included in the LAYOUTRETUN command. In general fallback to NFS is restricted to the cases of server or client failure recovery. In this case the fallback will be related to the permission access issue as an additional case of fallback to NFS.

4.2.2. Layout command that require permissions check by the client

Assume there is a list of devices used by a given file. The client attempts a write operation and fails with a permission error. The client will retry (fallback) the I/O via the metadata server.

For block layout type, the client SHOULD return the layout before attempting to retry the I/O via the MDS. Object and file clients, need not return the layout before attempting to retry the I/O via the MDS.

If the client returns the layout, the client SHOULD indicate which device caused an error (or the range of the file in which the error occurred).

4.2.2.1. Case 1 - MDS successfully performs I/O to the device

MDS proactively sends an CB_LAYOUT_ACCESS_CHECK to all clients that have a layout referencing the storage device which recently returned a permission access error. The CB _LAYOUT_ACCESS_CHECK will contain a file handle, and a list of offsets. For file layout, the client can compute the data servers to which it must send an NFS ACCESS command. The client SHOULD issue the NFS ACCESS command on behalf of any one of the users that have the file currently open on each client. The

client should then accumulate the results of all the access checks (there may be more than one device checked). The client returns a vector of device handles and statuses.

The STATUS code is either NFS4_OK or the error code returned by the data store. The implementation details of how the server aggregates the client responses to CB_LAYOUT_ACCESS_CHECK is left as an exercise for the reader. In many instances if the server detects that a majority, or a large number of clients cannot access some devices, the server will issue CB_LAYOUT_RECALL to all the clients, if possible it will restripe (or re-layout) the file to exclude the failing device.

4.2.2.2. Case 2 - MDS fails to perform the I/O to the device

This is the same as case 1, except that the server can restripe the file, only if the failed device does not yet contain data for the file. Implementations may decide to remove the failing device from the list of devices used for new files.

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 and the MDS gets an access permission denial from the storage device, the MDS cannot perform the I/O and returns an error to the client. In this case all client I/O to that device will fail and the reason for these failures needs to be communicated to the MDS. To address this case the client will use the new layoutreturn_type4 operation defined in section 3.2 and the new NFS4ERR_DEVICE_PERM_DENY error code to inform the MDS of possible permission access issues. An additional option is to use the new CB_LAYOUTACCESSCHECKRECALL from section 3.3sent to the client requesting permission access check. On failure the MDS will log an error NFS4ERR_DEVICE_UNACCESSIBLE to inform the admin to correct the problem. On receiving the CB the client will send the SD a GETDEVICEINFO and report NFS4ERR_DEVICE_PERM_DENY to the MDS server using the new layout command from section 3.2.

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 is applicable to all pNFS layout types.

6. IANA Considerations

There are no IANA considerations in this document beyond pNFS IANA Considerations are covered in [RFC5661].

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.

[RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", http://tools.ietf.org/html/rfc5661, January 2010.

[RFC5663] Black, D., Glasgow, J., Fridella, S., "Parallel NFS (pNFS) Block/Volume Layout", http://tools.ietf.org/html/rfc5663, January 2010.

[RFC5664] Halevy, B., Welch, B., Zelenka, J., "Object-Based Parallel NFS (pNFS) Operations", http://tools.ietf.org/html/rfc5664, January 2010

[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

Jason Glasgow
Google
5 Cambridge Center, Floors 3-6
Cambridge, MA 02142
US

Phone: +1 (617) 575 1300
Email: jglasgow@google.com