NFSv4
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2011

M. Eisler
D. Kenchammana
J. Lentini
M. Shankararao
NetApp
R. Iyer

February 23, 2011

NFS space reservation operations
draft-iyer-nfsv4-space-reservation-ops-02.txt

Abstract

This document describes a set of NFS attributes and operations that are useful for applications like hypervisors to manage storage in a better manner.

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 August 27, 2011.

Copyright Notice

Table of Contents

1. Requirements notation 3
2. Introduction 3
3. Use Cases 4
3.1. Space Reservation 4
3.2. Space freed on deletes 4
4. Operations and attributes 5
4.1. Attribute X: space_reserve 5
4.2. Attribute Y: space_freed 6
4.3. Attribute Z: max_hole_punch 6
4.4. Operation A: HOLE_PUNCH - Zero and deallocate blocks backing the file in the specified range. 6
5. Security Considerations 8
6. IANA Considerations 8
7. Normative References 8
Authors' Addresses 8

1. Requirements notation

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].

2. Introduction

This document describes a set of operations that allow applications such as hypervisors to reserve space for a file, report the amount of actual disk space a file occupies and freeup the backing space of a file when it is not required.

In virtualized environments, virtual disk files are often stored on NFS mounted volumes. Since virtual disk files represent the hard disks of virtual machines, hypervisors often have to guarantee certain properties for the file.

One such example is space reservation. When a hypervisor creates a virtual disk file, it often tries to preallocate the space for the file so that there are no future allocation related errors during the operation of the virtual machine. Such errors prevent a virtual machine from continuing execution and result in downtime.

Another useful feature would be the ability to report the number of blocks that would be freed when a file is deleted. Currently, NFS reports two size attributes:

- size - The logical file size of the file.
- space_used - The size in bytes that the file occupies on disk

While these attributes are sufficient for space accounting in traditional filesystems, they prove to be inadequate in modern filesystems that support block sharing. Having a way to tell the number of blocks that would be freed if the file was deleted would be useful to applications that wish to migrate files when a volume is low on space.

Since virtual disks represent a hard drive in a virtual machine, a virtual disk can be viewed as a filesystem within a file. Since not all blocks within a filesystem are in use, there is an opportunity to reclaim blocks that are no longer in use. A call to deallocate blocks could result in better space efficiency. Lesser space MAY be consumed for backups after block deallocation.

We propose the following operations and attributes for the

aforementioned use cases:

space_reserve: This attribute specifies whether the blocks backing the file have been preallocated.

space_freed: This attribute specifies the space freed when a file is deleted, taking block sharing into consideration.

HOLE_PUNCH: This operation zeroes and/or deallocates the blocks backing a region of the file.

max_hole_punch: This attribute specifies the maximum sized hole that can be punched on the filesystem.

3. Use Cases

3.1. Space Reservation

Some applications require that once a file of a certain size is created, writes to that file never fail with an out of space condition. One such example is that of a hypervisor writing to a virtual disk. An out of space condition while writing to virtual disks would mean that the virtual machine would need to be frozen.

Currently, in order to achieve such a guarantee, applications zero the entire file. The initial zeroing allocates the backing blocks and all subsequent writes are overwrites of already allocated blocks. This approach is not only inefficient in terms of the amount of I/O done, it is also not guaranteed to work on filesystems that are log structured or deduplicated. An efficient way of guaranteeing space reservation would be beneficial to such applications.

If the space_reserved attribute is set on a file, it is guaranteed that writes that do not grow the file will not fail with NFSERR_NOSPC.

3.2. Space freed on deletes

Currently, files in NFS have two size attributes:

- size - The logical file size of the file.
- space_used - The size in bytes that the file occupies on disk.

While these attributes are sufficient for space accounting in traditional filesystems, they prove to be inadequate in modern filesystems that support block sharing. In such filesystems,

multiple inodes can point to a single block with a block reference count to guard against premature freeing.

If space_used of a file is interpreted to mean the size in bytes of all disk blocks pointed to by the inode of the file, then shared blocks get double counted, over-reporting the space utilization. This also has the adverse effect that the deletion of a file with shared blocks frees up less than space_used bytes.

On the other hand, if space_used is interpreted to mean the size in bytes of those disk blocks unique to the inode of the file, then shared blocks are not counted in any file, resulting in under-reporting of the space utilization.

For example, two files A and B have 10 blocks each. Let 6 of these blocks be shared between them. Thus, the combined space utilized by the two files is 14 * BLOCK_SIZE bytes. In the former case, the combined space utilization of the two files would be reported as 20 * BLOCK_SIZE. However, deleting either would only result in 4 * BLOCK_SIZE being freed. Conversely, the latter interpretation would report that the space utilization is only 8 * BLOCK_SIZE.

Adding another size attribute, space_freed, is helpful in solving this problem. space_freed is the number of blocks that are allocated to the given file that would be freed on its deletion. In the example, both A and B would report space_freed as 4 * BLOCK_SIZE and space_used as 10 * BLOCK_SIZE. If A is deleted, B will report space_freed as 10 * BLOCK_SIZE as the deletion of B would result in the deallocation of all 10 blocks.

The addition of this problem doesn't solve the problem of space being over-reported. However, over-reporting is better than under-reporting.

4. Operations and attributes

In the sections that follow, one operation and three attributes are defined that together provide the space management facilities outlined earlier in the document. The operation is intended to be OPTIONAL and the attributes RECOMMENDED as defined in section 17 of [RFC5661].

4.1. Attribute X: space_reserve

The space_reserve attribute is a read/write attribute of type boolean. It is a per file attribute. When the space_reserved attribute is set via SETATTR, the server must ensure that there is

disk space to accommodate every byte in the file before it can return success. If the server cannot guarantee this, it must return NFS4ERR_NOSPC.

If the client tries to grow a file which has the space_reserved attribute set, the server must guarantee that there is disk space to accommodate every byte in the file with the new size before it can return success. If the server cannot guarantee this, it must return NFS4ERR_NOSPC.

It is not required that the server allocate the space to the file before returning success. The allocation can be deferred, however, it must be guaranteed that it will not fail for lack of space.

The value of space_reserved can be obtained at any time through GETATTR.

In order to avoid ambiguity, the space_reserve bit cannot be set along with the size bit in SETATTR. Increasing the size of a file with space_reserve set will fail if space reservation cannot be guaranteed for the new size. If the file size is decreased, space reservation is only guaranteed for the new size and the extra blocks backing the file can be released.

4.2. Attribute Y: space_freed

space_freed gives the number of bytes freed if the file is deleted. This attribute is read only and is of type length4. It is a per file attribute.

4.3. Attribute Z: max_hole_punch

max_hole_punch specifies the maximum size of a hole that the HOLE_PUNCH operation can handle. This attribute is read only and of type length4. It is a per filesystem attribute. This attribute MUST be implemented if HOLE_PUNCH is implemented.

4.4. Operation A: HOLE_PUNCH - Zero and deallocate blocks backing the file in the specified range.

ARGUMENTS

```
struct HOLE_PUNCH4args {
    /* CURRENT_FH: file */
    offset4        hpa_offset;
    length4        hpa_count;
};
```

RESULTS

```
            struct HOLEPUNCH4res {
                nfsstat4        hpr_status;
        };
```

DESCRIPTION

Whenever a client wishes to deallocate the blocks backing a particular region in the file, it calls the HOLE_PUNCH operation with the current filehandle set to the filehandle of the file in question, start offset and length in bytes of the region set in hpa_offset and hpa_count respectively. All further reads to this region MUST return zeros until overwritten. The filehandle specified must be that of a regular file.

Situations may arise where hpa_offset and/or hpa_offset + hpa_count will not be aligned to a boundary that the server does allocations/ deallocations in. For most filesystems, this is the block size of the file system. In such a case, the server can deallocate as many bytes as it can in the region. The blocks that cannot be deallocated MUST be zeroed. Except for the block deallocation and maximum hole punching capability, a HOLE_PUNCH operation is to be treated similar to a write of zeroes.

The server is not required to complete deallocating the blocks specified in the operation before returning. It is acceptable to have the deallocation be deferred. In fact, HOLE_PUNCH is merely a hint; it is valid for a server to return success without ever doing anything towards deallocating the blocks backing the region specified. However, any future reads to the region MUST return zeroes.

HOLE_PUNCH will result in the space_used attribute being decreased by the number of bytes that were deallocated. The space_freed attribute may or may not decrease, depending on the support and whether the blocks backing the specified range were shared or not. The size attribute will remain unchanged.

The HOLE_PUNCH operation MUST NOT change the space reservation guarantee of the file. While the server can deallocate the blocks specified by hpa_offset and hpa_count, future writes to this region MUST NOT fail with NFSERR_NOSPC.

The HOLE_PUNCH operation may fail for the following reasons (this is a partial list):

NFS4ERR_NOTSUPP: The Hole punch operations is not supported by the NFS server receiving this request.

NFS4ERR_DIR: The current filehandle is of type NF4DIR.

NFS4ERR_SYMLINK: The current filehandle is of type NF4LNK.

NFS4ERR_WRONG_TYPE: The current filehandle does not designate an ordinary file.

5. Security Considerations

There are no security considerations.

6. IANA Considerations

This document has no actions for IANA.

7. Normative References

[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", RFC 5661, January 2010.

Authors' Addresses

Mike Eisler
NetApp
5765 Chase Point Circle
Colorado Springs, CO 80919
USA

Phone: +1 719 599 9026
Email: mike@eisler.com
URI: http://www.eisler.com

Deepak Kenchammana
NetApp
475 East Java Drive
Sunnyvale, CA 94089
USA

Phone: +1 408 822 4765
Email: kencham@netapp.com

James Lentini
NetApp
1601 Trapelo Rd, Suite 16
Waltham, MA 02451
USA

Phone: +1 781 768 5359
Email: jlentini@netapp.com

Manjunath Shankararao
NetApp
3rd Floor, Fair Winds Block, EGL Software Park
Bangalore, Karnataka 560085
INDIA

Phone: +91 80 4184 3397
Email: rudra@netapp.com

Rahul Iyer
655 S Fair Oaks Ave Apt #I-314
Sunnyvale, CA 94086
USA

Email: rahulair@yahoo.com