OpenAFS Directory Objects

Sine Nomine Associates

October 16, 2012

Introduction

- Description of directory objects in AFS
- Discussion of directory object defragmentation project

Motivation

- How many files can I have in a directory?
- My directory is full, now what?

```
$ touch hello
touch: cannot touch ‘hello’: File too large
```

Internet Draft

- AFS-3 Directory Object Type Definition
- Written by Tom Keiser
- draft-keiser-afs3-directory-object-00

AFS Directory Objects

- Storage of directory entry names in AFS
- Path to AFS File Id (FID) lookup
- Servers and clients share a common object layout
- Servers and clients share a common lookup and modification algorithms

Note

The current directory object format was introduced in 1988, to expand the size of directories.

Tradeoffs

Pro

- Avoid fileserver load for frequent lookups
- Processing moved to the client for faster lookups
- Better performance for many general workloads

Con

- Hard limits on number of entries per directory
- Difficult to extend (again) in a backward compatible way
- Worse performance for some specific workloads

Pages

- Each directory object consists of 1 to 1023 *pages*, 2048 bytes per page
- Each page consists of 64 *records*, 32 bytes per record
- The first record of each page contains a page header which indicates which records in the page are in use (bitmap)
- The 2nd to 12th records of the first page contains a directory header, which contains a hash by name table.

Records

Each directory entry requires at least one record, with a header followed by the entry name.

- flags - 0x01 the first byte of the entry
- reserved - no longer used
- next - hash chain for lookups by name
- vnode - the AFS FID vnode number for this entry
- uniquifier - the AFS FID uniquifier for this entry
- name - the first 20 bytes of the entry name

Extensions

- Extent records for names too long to fit in one record
- Extents must be contiguous and may not span pages
- Extents do not have headers – the name just spills over
- An extent is allocated if the name is more than 15 bytes (nul excluded)
- Each additional 32 bytes requires another extent record

Note

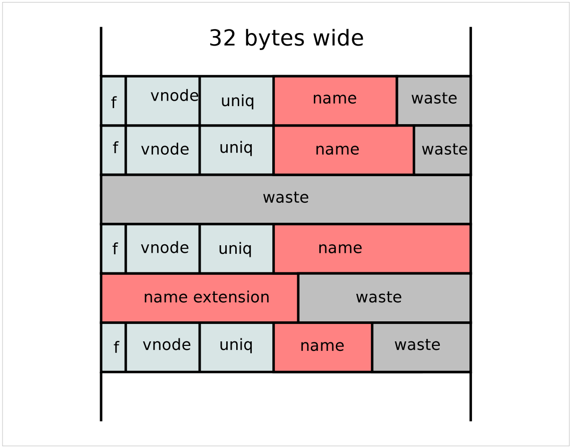
The maximum entry name size is 256 bytes, excluding the nul.

Records Needed

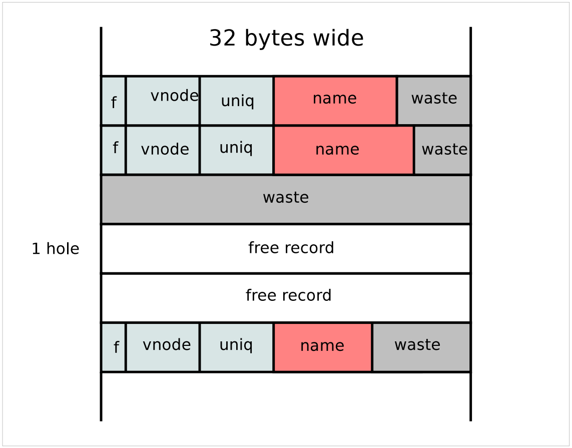
Records needed for directory entries by bytes (excluding nul).

```
records   entry size
-------   ----------
  1         1 ..  15 (not 19)
  2        16 ..  47
  3        48 ..  79
  4        80 .. 111
  5       112 .. 143
  6       144 .. 175
  7       176 .. 207
  8       208 .. 239
  9       240 .. 256 (not 272)
```

Records


32 bytes wide
f
vnode
uniq
name
waste
f
vnode
uniq
name
waste
waste
f
vnode
uniq
name
name extension
waste
f
vnode
uniq
name
waste

Records


32 bytes wide
f
vnode
uniq
name
waste
f
vnode
uniq
name
waste
waste
1 hole
free record
free record
f
vnode
uniq
name
waste

Holes and Fragmentation

- *Holes* are created in the directory object when entries are removed
- A hole may be one to 63 records long (54 on the first page)
- Since entries require contiguous records, fragmentation reduces the number of directory entries supported per directory

Maximum Entries per Directory

- The theoretical maximum number of directory entries per directory is 64436
- This max can only be reached only if all the entries are less than 16 bytes long
- In practice, the number of entries is about 25,000, depending on the sizes of the directory entries and the amount of record fragmentation

Note

Users will see a 'File too large error' if a hole cannot be found to create the directory entry.

Dir Defrag Project

- Code developed by Tom Keiser
- A small project to provide the means to defragment directory objects, allowing for more directory entries
- Determine directory usage statistics to show if a directory needs to be defragmented
- Defrag directories by packing largest to smallest entries, to a copy (not in place)
- Currently unit testing stats and defrag
- Defrag may be part of the salvage process

Usage Stats

Example full directory with no holes

```
$ ls -a | wc -l -c
12767 1657542

$ touch hello
touch: cannot touch ‘hello’: File too large

# dtest -y /vicepa/AFSIDat/7/7+++U/+/+/F++++ouP1
npages: 1023
nfree: 0
nholes: 0
hole_len_avg: 0.000000
```

Usage Stats

After 100 files removed

```
$ ls -a | wc -l -c
12667 1644769

# dtest -y /vicepa/AFSIDat/7/7+++U/+/+/F++++ouP1
npages: 1023
nfree: 499
nholes: 100
hole_len_avg: 4.990000
```

Defrag Example

Pathological case: every other record free

```
npages: 1023
nfree: 32217
nholes: 32217
hole_len_avg: 1.000000
```

```
$ touch 1234567890123456
touch: cannot touch ‘1234567890123456’: File too large
```

Defrag Example

After the defrag operation:

```
npages: 512
nfree: 24
nholes: 1
hole_len_avg: 24.000000
```

Discussion

- When should the defrag be run?
- Is there a way to capture 'File too large' errors?
- Could this be done automatically during a demand salvage?
- How can the directory usage stats be reported to admins?

Thank You

- Michael Meffie
- mmeffie@sinenomine.net