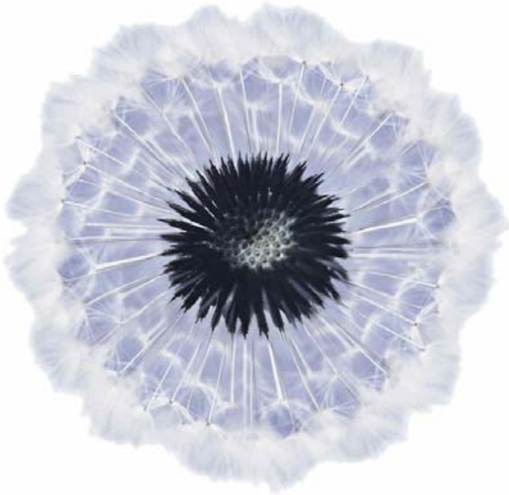
NetApp™

Go further, faster™

NFSv4 and Sub-File Caching



Mike Eisler

2009-03-25

Data Caching today in NFSv4

- Whole file, via delegations
- Exclusive (writer) or shared (reader)
- Delegations are recalled when a conflicting OPEN is received
  - OPEN for READ or WRITE recalls an exclusive delegation
  - OPEN for WRITE recalls a shared delegation
- Work best for read-only or single instance write workloads
  - Many (most?) single instance write workloads use locking
    - Exclusive delegations are good for caching byte range locks

Why Consider Sub-File Delegations Now?

- Our experience with pNFS shows that we know how to deal with sub-file organization
  - A blocks (SCSI) layout is sort of like a sub-file delegation
    - read layouts conflict with read/write layouts
- Critical applications need sub-file sharing
  - byte range locks aren't good enough
    - not recallable
    - advisory in many cases
      - consistency not ensured
- Flash memory is a game changer
  - The benefits of flash are best closest to the application
    - Unlike disks, aggregating flash storage devices does little to improve latency and throughput
  - NFS server vendors: embrace it or else

Requirements, Non-Requirements, & Maybes

Requirements

- Sub-file consistency
- Enable applications that are already splitting data into fixed size power of 2 blocks
  - database
  - hypervisor
- Compact Representation
- Optimize for big files
  - existing delegations serve small files fine
- Deal with striping
- Deal with de-duplication

Non-Requirements

- Arbitrary byte ranges
  - We don't have to fix all gaps with POSIX (at least not now)

Maybe

- True sub-file coherency
  - If there is high contention on a block, the cost direct I/O from/to NFS server is probably less than thrashing on cache token

It turns out …

… draft-eisler-nfsv4-pnfs-dedupe-00.txt deals addresses many of the requirements:

- Sub-file consistency
  - I-D has a ddl_change_attr array in layout
    - If absent, server is promising to send CB_LAYOUTRECALL on the affected block
    - Trivial to allow client to tell server return layouts with ddl_change_attr absent
- Enable applications that are already splitting data into fixed size power of 2 blocks
  - I-D uses bit maps to represent blocks
- Compact Representation
  - ditto
- Optimize for big files
  - I-D uses hierarchical bit maps to represent blocks
  - But I-D works well with small files too
- Deal with striping
  - On a per block (or per block range) basis, I-D's protocol can refer client to a layout type
- Deal with de-duplication
  - It has "dedupe" in the I-D name ☺

Maybe

- True sub-file coherency
  - If we want to go there, trivial to allow client to demand coherency
    - If we want to go there

Why not add sub-file delegation operations?

- Might turn NFSv4.2 into another death march like NFSv4.1
- We've added a lot of extensibility to NFSv4.1
  - Let's see if we can use it

Suggested Next Steps

- Post a Requirements I-D
- Begin a discussion
- Post draft-eisler-nfsv4-pnfs-dedupe-01.txt
  - add block level caching

Thanks

Q/A

Or I can reprise the draft-eisler-nfsv4-pnfs-dedupe-00.txt presentation from Minneapolis

De-Duplication Awareness: What I am asking of NFSv4 WG

- Primary request
  - Add de-duplication awareness to the NFSv4 charter
    - virtualization is the justification
- Secondary request
  - Start with draft-eisler-nfsv4-pnfs-dedupe-00.txt
    - Seems to fit with known de-duplication schemes

Why?

- Magnetic disk is cheap
- And yet customers are driving storage vendors toward eliminating redundancy
  - first it was whole files
  - now it is blocks within files
- NFS clients cache data from storage arrays in DRAM and flash
  - DRAM and flash are expensive
- Ergo, de-duplication in NFS clients matters
- The hypervisors are doing it already
  - So storage arrays should give hypervisors the de-duplication maps

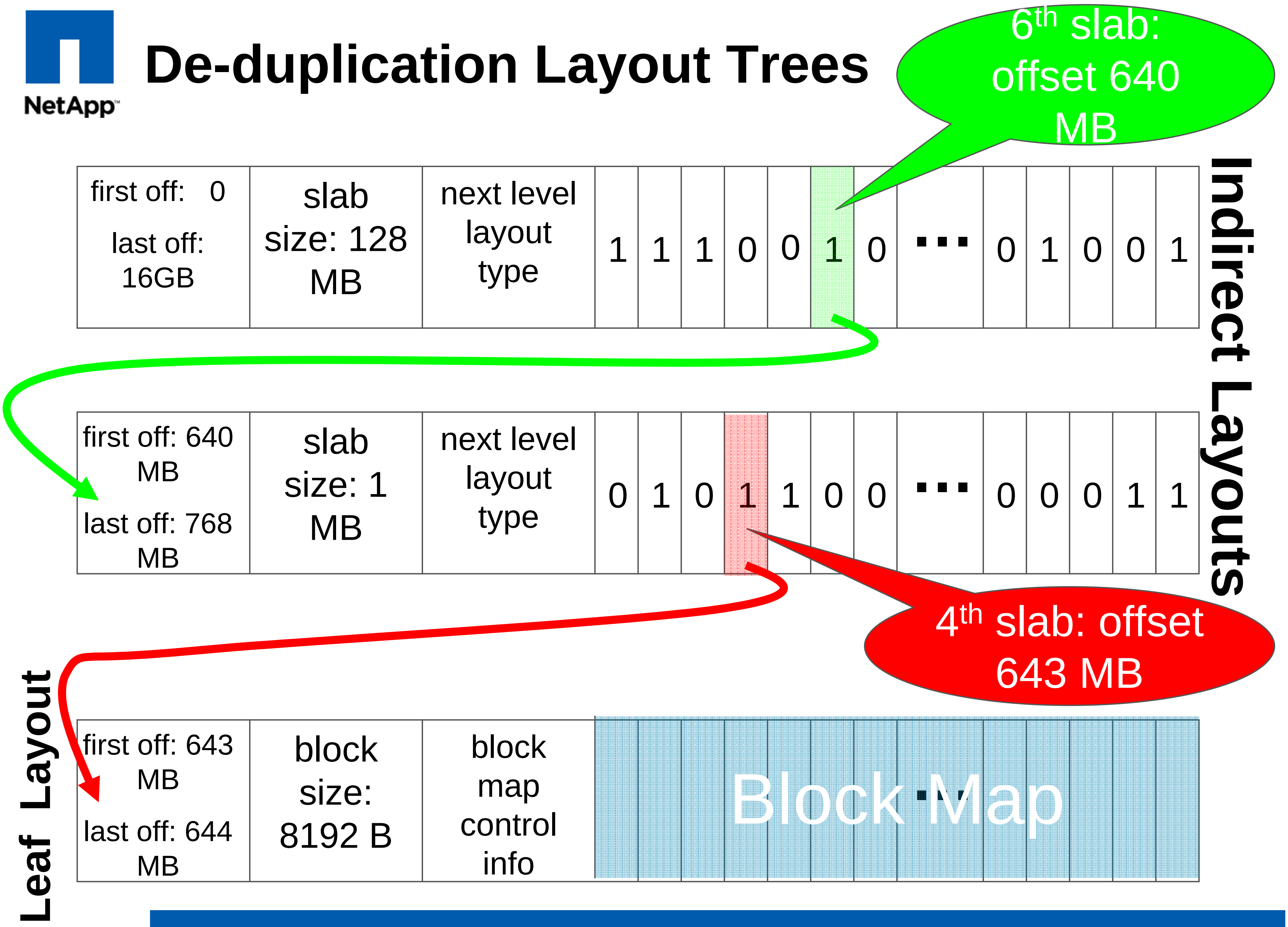
The proposal at a glance

- Does not require a new minor version of NFSv4
- Requires new layout types
- Use bit maps to indicate if a range of data in a file is a duplicate from another file
- Supports hierarchical (e.g., clones, snapshots), in-line, and background de-duplication
- Supports cross-storage-node de-duplication
  - Can integrate with existing files, objects, and blocks layouts
- Limited to regular files
  - De-duplication awareness of directories is reasonable,
    - but perhaps best captured in a separate document

Concepts

- Source file:
  - the file that contains the de-duplicated data.
- Target file:
  - the file the client has opened.
- Block:
  - the smallest unit of de-duplication that the server is willing to support.
- Slab:
  - a byte range that refers to lists of smaller slabs or blocks
- Regular file:
  - An object of file type NF4REG or NF4NAMEDATTR
- Indirect layouts contain slabs
  - Refer to indirect layouts or leaf layouts
- Leaf layouts contain blocks
  - Leaf layouts indicate the source files

De-duplication Layout Trees



first off: 0 last off: 16GB	slab size: 128 MB	next level layout type	1 1 1 0 0 1 0 ... 0 1 0 0 1
first off: 640 MB last off: 768 MB	slab size: 1 MB	next level layout type	0 1 0 1 1 0 0 ... 0 0 0 1 1
first off: 643 MB last off: 644 MB	block size: 8192 B	block map control info	Block Map

6th slab: offset 640 MB

4th slab: offset 643 MB

Leaf Layout
Hierarchical De-duplication (snapshot, clone)

first off: 643 MB last off: 644 MB	block size: 8192 B	block map control info	1	0	1	1	1	1	0	...	1	1	1	0	1

125th block: target offset 675250176

- ddll_fhlist[0] – source file
- ddl_change_attr[0]
  - If absent: server will recall leaf layout before changing active blocks of source file.
  - If present: client must compare ddl_change_attr[0] with change attribute of source file before using block from source.
- source offset: also 675250176
  - 643 * 1024 * 1024 + (125 - 1) * 8192

Let Client Dictate

Leaf Layout
Non-Hierarchical De-duplication (inline, background)

first off: 643 MB last off: 644 MB	block size: 8192 B	block map control info	1, 2, 67	1, 1, 100	▪▪▪	0, 0, 0	1, 1, 5001

2nd block: target offset 674242560

- ddll_fhlist[] – source files – { 0x12, **0x67**, 0x43 }
- source fh of 2nd block: 0x67
- source offset of 2nd block: 100 * 8192 = 819200
- target offset: 674242560 = 643*1024*1024 + (2-1)*8192

Leaf Layout
Cross-Node De-duplication

first off: 643 MB last off: 644 MB	block size: 8192 B	block map control info	1, 2, 2, 67	1, **1**, **2**, 100	…	0, 0, 0	1, 1, 0, 5001

2nd block: target offset 674242560

- ddll_devlist[] – device IDs – { 0x333, **0x111**, 0x222 }
- ddll_fhlist[] – source files – { 0x12, 0x67, **0x43** }
- source file's device: ID 0x111
    - can map to network address of another MDS
    - can map to any non-de-dupe layout type (files, blocks, objects, metadata, …)
- source fh of block 1: 0x43
- source offset of block 1: 100 * 8192 = 819200
- target offset: 674242560 = 643*1024*1024 + (2-1)*8192