NetApp™

Go further, faster™

pNFS support for ONTAP Unstriped file systems (WIP)

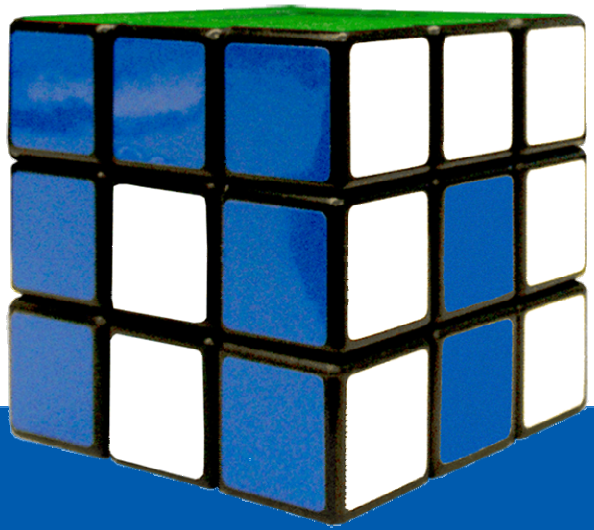
Pranoop Erasani
pranoop@netapp.com
Connectathon – Feb 22, 2010

Agenda

- Clustered ONTAP Architecture
- Striped WAFL
- pNFS and Striped WAFL
- Unstriped WAFL
- pNFS and Unstriped WAFL
- Scale out WAFL entities
- pNFS and Scaleout WAFL
- Questions

NetApp™

Clustered ONTAP Architecture

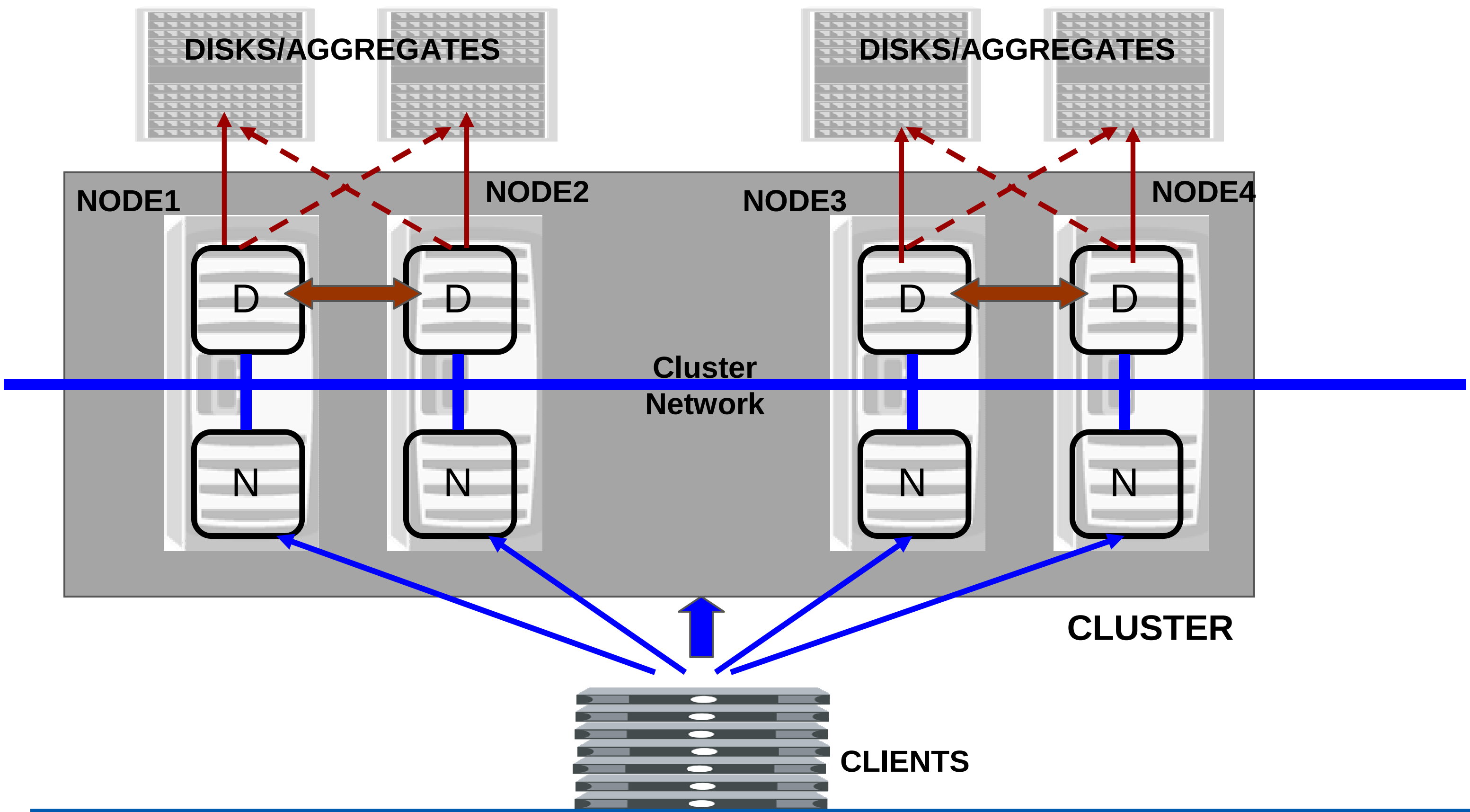


Clustered ONTAP

- NetApp's Next-generation ONTAP
- Basically, clustered system of HA (High Availability) pairs
- Building blocks
  - N-blade
  - D-blade
  - VLDB (Volume Location Database)
  - VifMgr (Virtual Interface Manager)
  - SpinNP
  - Others (Management)….
- Primarily built for Global namespace
  - **Junctions** (a new filesystem object) stitch the namespace
  - **Vserver (virtual server)** has it's own namespace stitched from volumes in the cluster
  - Each **vserver** has a root volume and rest of them are brought into namespace via junctions

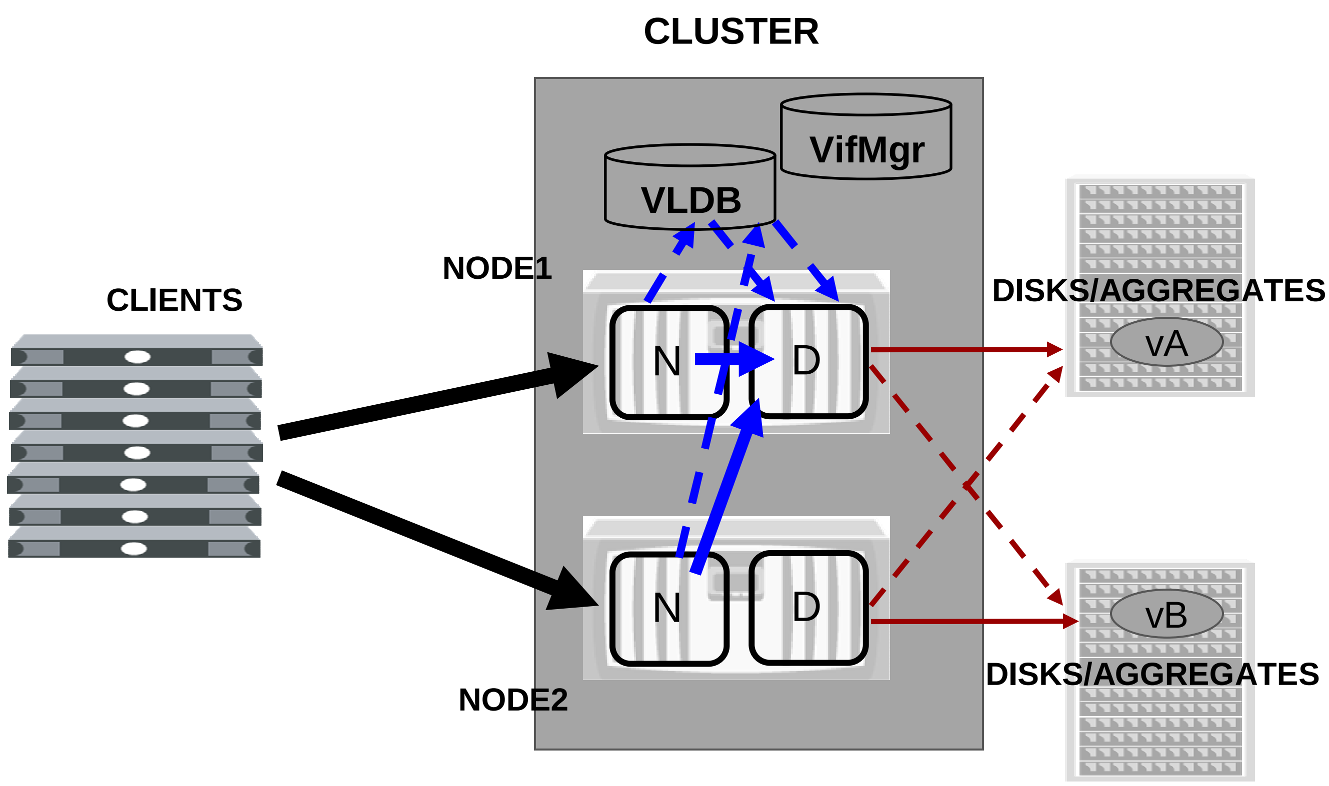
Clustered ONTAP outline

- Cluster of HA pairs - HA pairs help in storage failover
- N-blade: Client-facing, owns networking, protocol stack
- D-blade: Owns disks, aggregates (disk groups) and thus volumes
- High Speed interconnect for cluster traffic

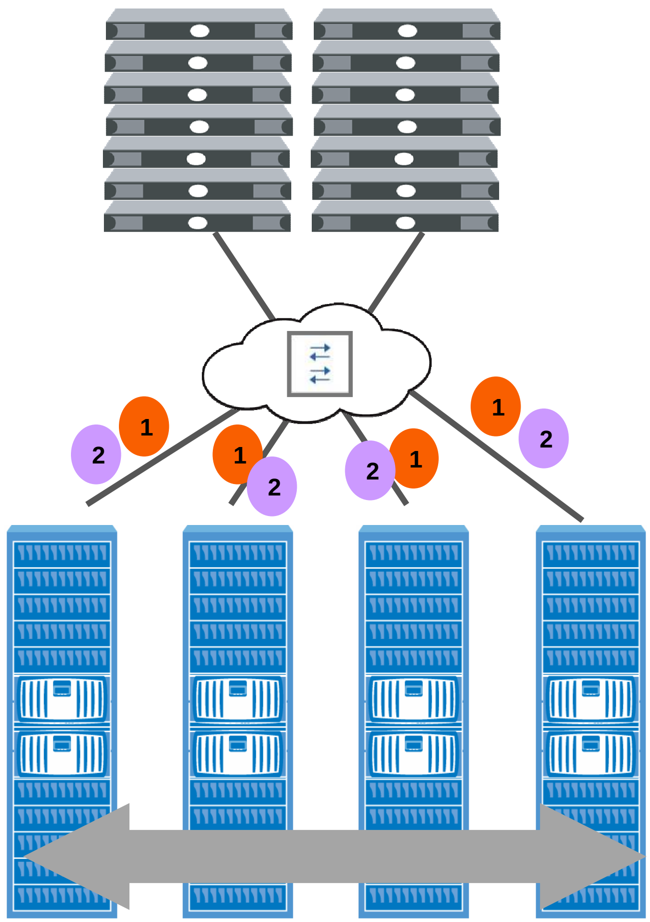


Scale-out cluster

- **N-blade**: Client-facing kernel module, owns networking, protocol stack
- **D-blade**: Storage-facing kernel module, owns disks, aggregates (disk groups) and volumes
- **VifMgr**: Virtual Interface Manager, manages networking related information
- **VLDB**: Volume Location Database, manages file system (WAFL) and name space information
- **SpinNP**: Protocol for communication between Blades



Clustered ONTAP and pNFS

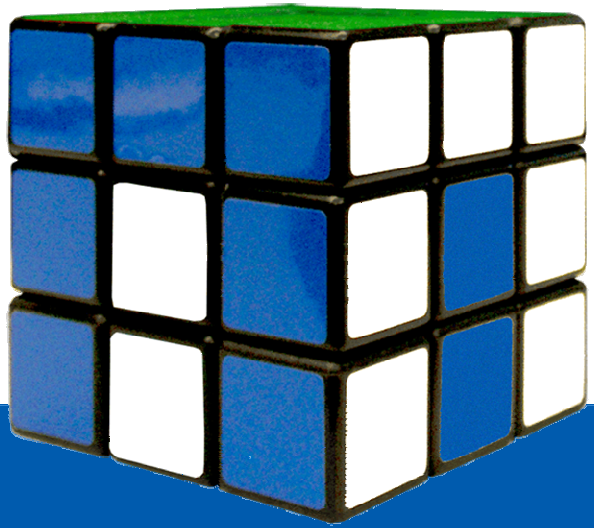


1 Metadata server operations

2 Data server operations

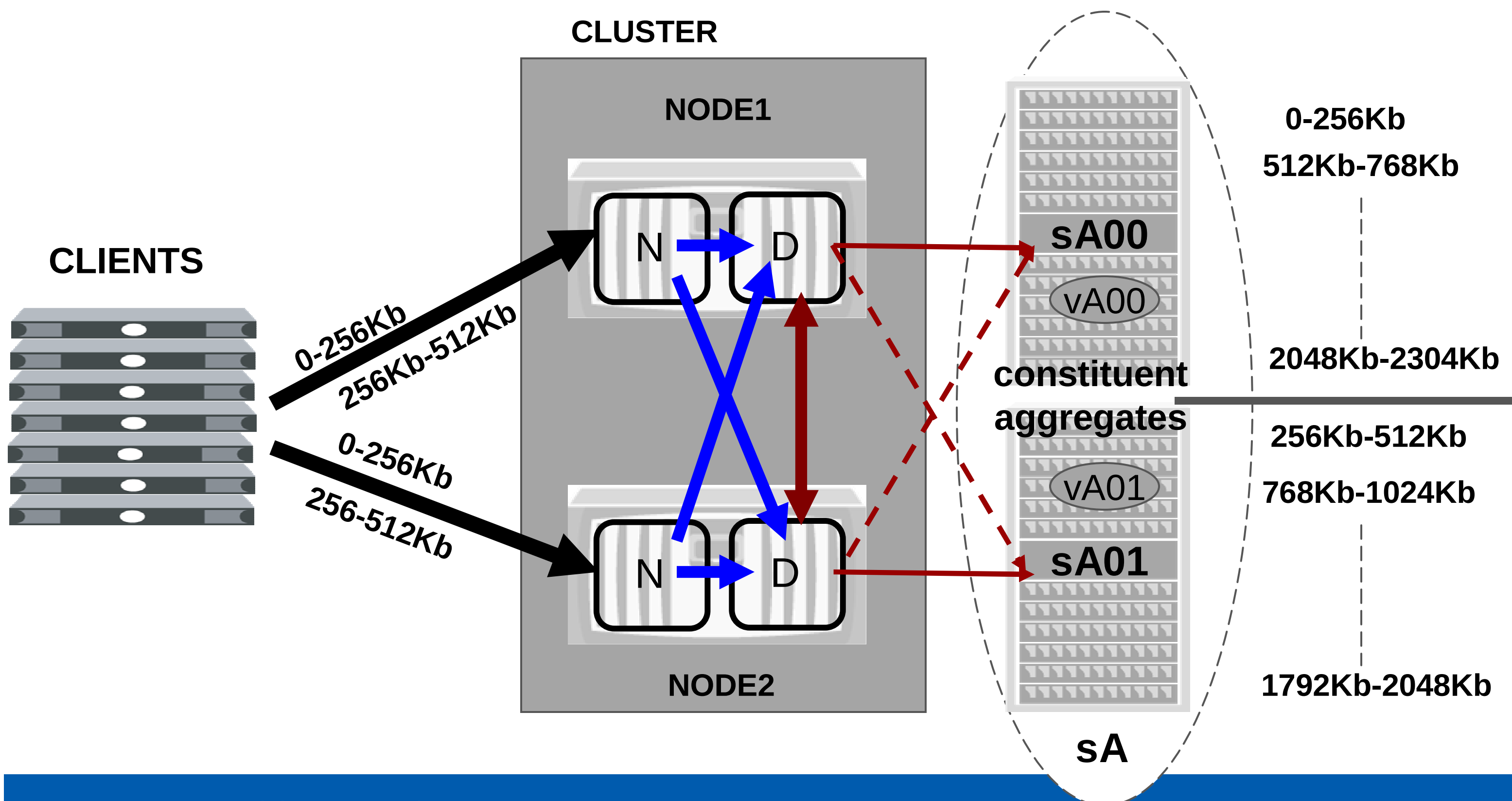
- Leverage ONTAP cluster backend (global name space)
- pNFS access to Striped WAFL volumes
- pNFS access to Unstriped WAFL volumes
- pNFS access to Scale Out WAFL entities
- Avoid single-blade data bottleneck
- Solve the remote N-D I/O latency problem
- Integrate with ONTAP manageability constructs (e.g volume move, failover etc.)

Striped WAFL



Striped Volumes

- A striped volume (vA) has constituent volumes (vA00, vA01)
- Striped aggregates (sA) – aggregates that hold striped volumes (sA00, sA01)
- Thus, Each D-blade could own a constituent volume (multiple too)
- Data gets distributed/striped (e.g., 2 stripes, 256Kb) across constituent volumes
- The striped volume looks like one manageable entity
- N-blade routes request based on striping configuration present in VLDB



Striped Volumes (Continued)

- Supports Data, Metadata striping
  - Data gets striped onto constituent volumes
  - Metadata owner varies with each file
  - Each constituent volume owns metadata of some files
- Supports Directory striping
  - Directory contents striped across constituent volumes
- Pattern repeats every 4096 stripes (proprietary algorithm)
- Example configuration (two constituent volumes) is a special case
  - As you may have seen in the previous slide
  - i.e.., we don't put two adjacent stripes on the same constituent volume
  - Appears as if its round robin

Example Striping Table (3-striped volumes)

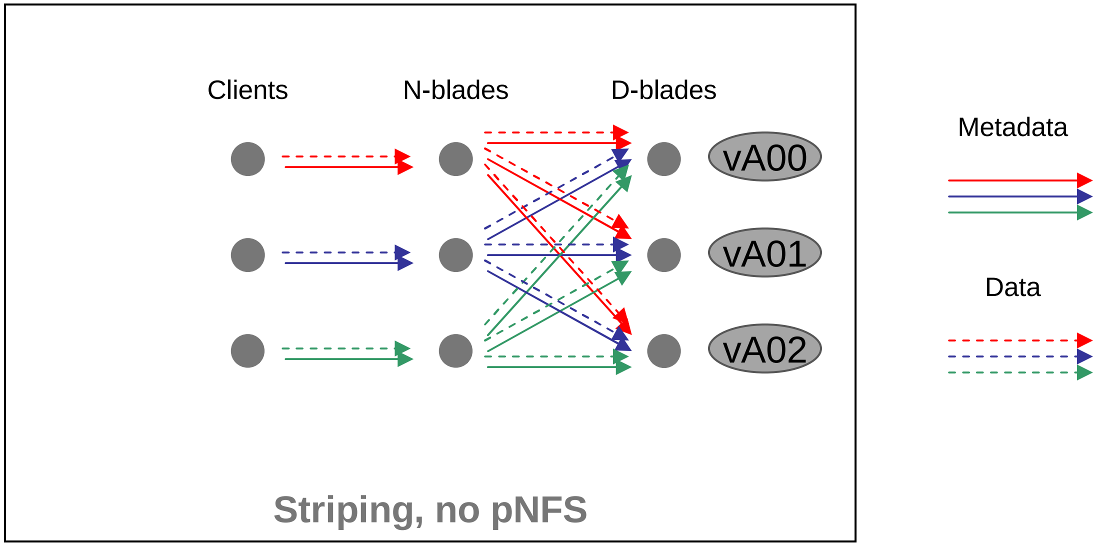
- Reminder: 0, 1, 2 are logical ID’s of constituent volumes
  - 0, 1, 2 do not necessarily appear in an order
  - Pattern repeats after 4096 elements (think large files)

<striping-table>
<data>02 01 00 02 00 01 00 01 02 01 00 02 00 02 00 01 </data>
<data>02 01 00 02 00 01 02 01 02 01 00 02 00 01 00 01 </data>
<data>02 01 00 02 00 02 00 01 02 01 00 02 00 01 02 01 </data>
<data>02 01 00 02 00 01 00 01 02 01 00 02 00 02 00 01 </data>
<data>02 01 00 02 00 01 02 01 02 01 00 02 00 01 00 01 </data>
<data>02 01 00 02 00 02 00 01 02 01 00 02 00 01 02 01 </data>
<data>02 01 00 02 00 01 00 01 02 01 00 02 00 02 00 01 </data>
<data>02 01 00 02 00 01 02 01 02 01 00 02 00 01 00 01 </data>
....
....
<data>02 01 00 02 00 01 00 01 02 01 00 02 00 02 00 01 </data>
<data>02 01 00 02 00 01 02 01 02 01 00 02 00 01 00 01 </data>
<data>02 01 00 02 00 02 00 01 02 01 00 02 00 01 02 01 </data>
<data>02 01 00 02 00 01 00 01 02 01 00 02 00 02 00 01 </data>
<data>02 01 00 02 00 01 02 01 02 01 00 02 00 01 00 01 </data>
<data>02 01 00 02 00 02 00 01 02 01 00 02 00 01 02 01 </data>
<data>02 01 00 02 00 01 00 01 02 01 00 02 00 02 00 01 </data>
<data>02 01 00 02 00 01 02 01 02 01 00 02 00 01 00 01 </data>
<data>02 01 00 02 00 02 00 01 02 01 00 02 00 01 02 01 </data>
<data>02 01 00 02 00 01 00 01 02 01 00 02 00 02 00 01 </data>
</striping-table>

Striped Volumes Terminology

- Constituent Count
  - Number of constituent volumes (2 to 254)
  - Striped volumes logically ID’d from 0 to N - 1
- Stripe Width (pNFS stripe unit size)
  - Maximum number of data bytes written and read to and from a specific constituent-volume (128K to 1G) before next one is picked
- Striping Table (pNFS stripe indices array)
  - Table that establishes striping pattern of striped volume files
  - Size fixed at 4096 entries – irrespective of number of constituent volumes
  - Basically, Pattern repeats after 4096 stripes
  - Elements in the table will be logical ID’s of constituent volumes
  - Proprietary algorithm – Not to be disclosed

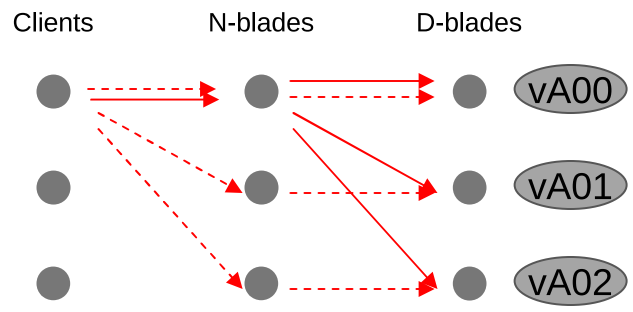
Striped WAFL (no pNFS)



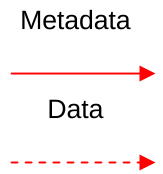
pNFS for Striped Volumes

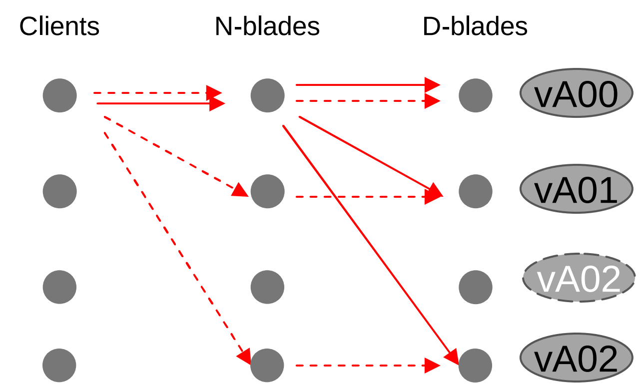
- Round Robin DNS - pick a network IP from a set of Lif’s
- With pNFS, MDS could be any of the Lif’s on any node
- So, there is a problem to solve i.e., find direct network path to each of the D-blade hosting Striped WAFL constituent volume
- Basically, ONTAP finds an IP that is local to each of the constituent volume
- Volume/network movement works fine as well, thanks to pNFS layout recalls
- Support sparse layouts - Routing within the cluster backend is based on logical file offset

Striped WAFL (pNFS)



Striping, pNFS



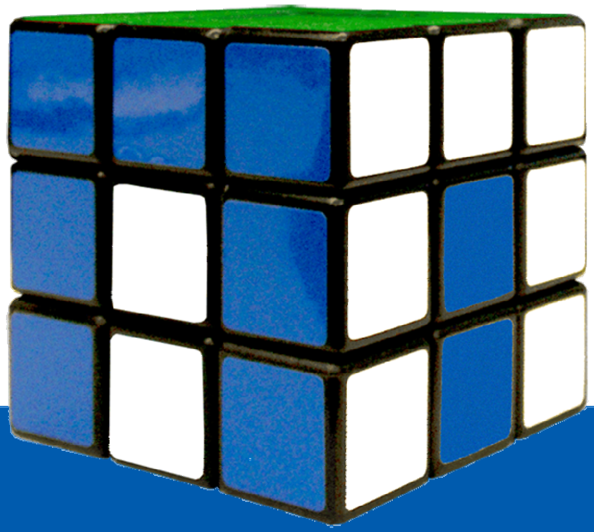


Striping, pNFS (after constituent movement)

Protocol mechanics

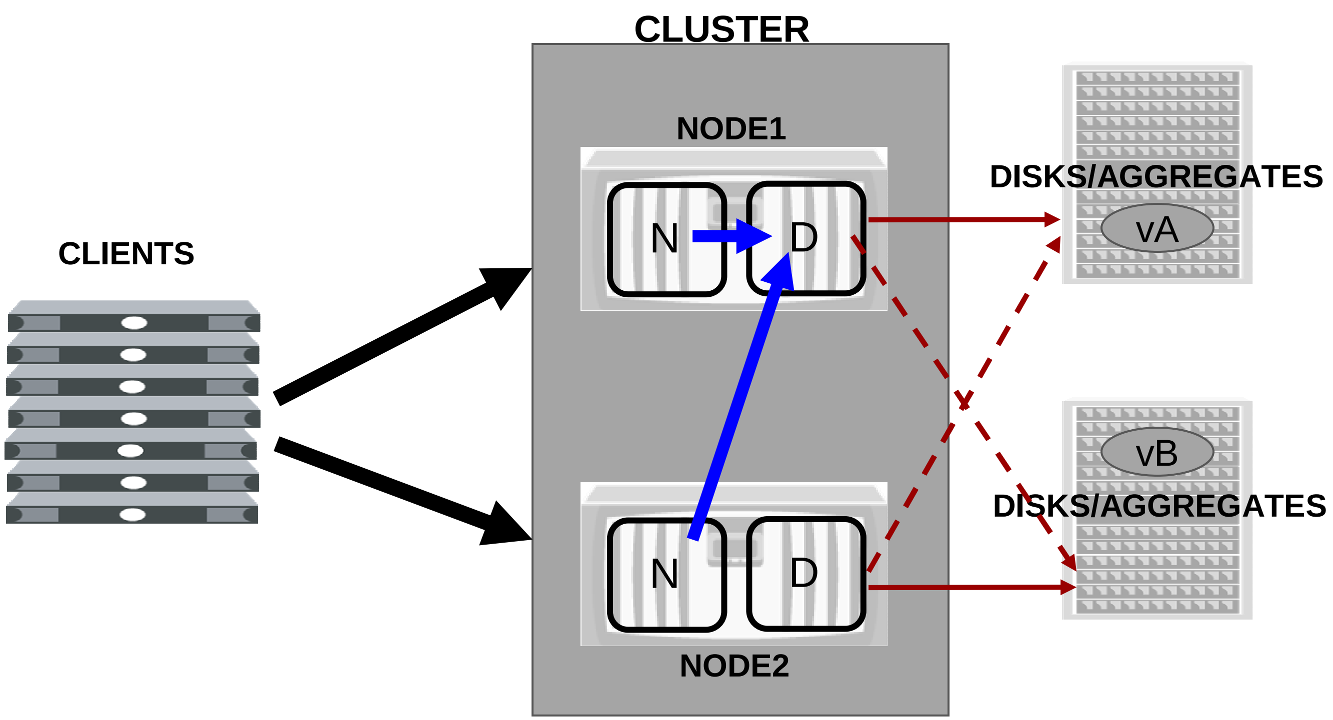
- No. of pNFS device addresses = constituent count
  - Of course, does not consider multipathing
  - When trunking is implemented, more addresses will be sent for every data server
- pNFS stripe indices count = file system striping table size
  - i.e. always 4096 stripe indices
  - Yes, our GETDEVICEINFO response is large
- pNFS first stripe index
  - Varies for each file
  - An index into the striping table of striped volume
  - Thus, different files start on different constituent volumes

Unstriped WAFL (Flexible Volumes)

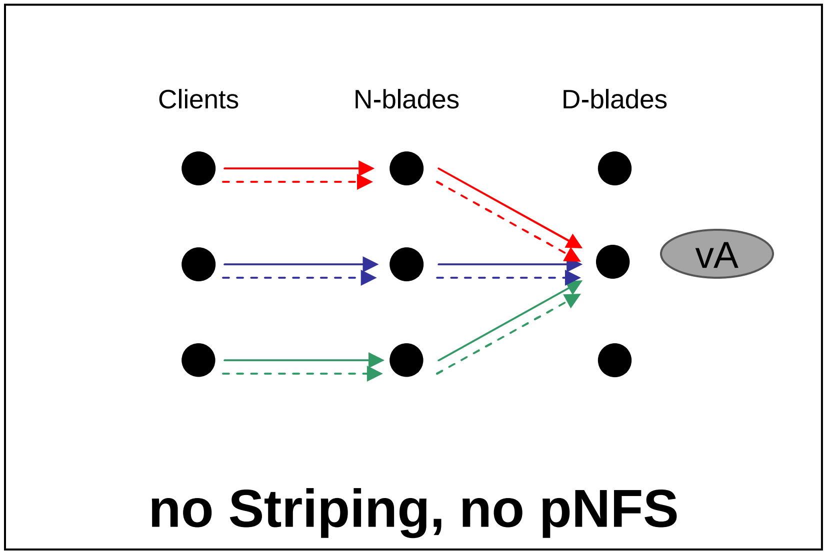


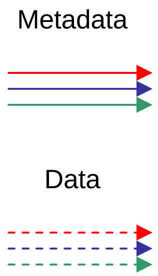
Flexible Volumes

- A D-blade owns aggregates (i.e.., RAID protected disk groups)
- Flexible volume sits within an aggregate
- Thus, at any moment, a flexible volume (vA or vB) sits on a D-blade
- Owning D-blade serves **SpinNP** protocol requests from all N-blades
- All protocol requests (NFSv[2,3,4,4.1], NLM, CIFS) converted to **SpinNP** requests



Unstriped volumes (No pNFS)

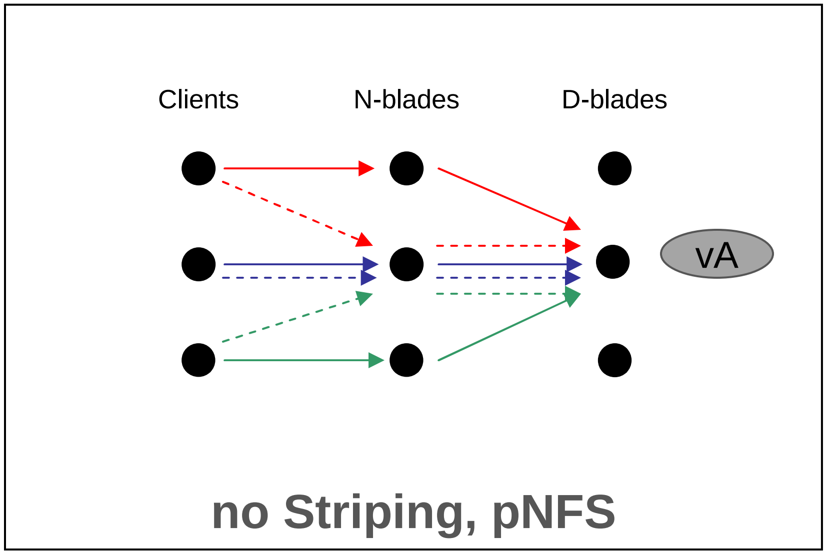


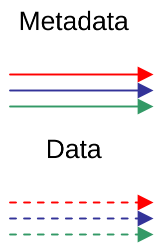


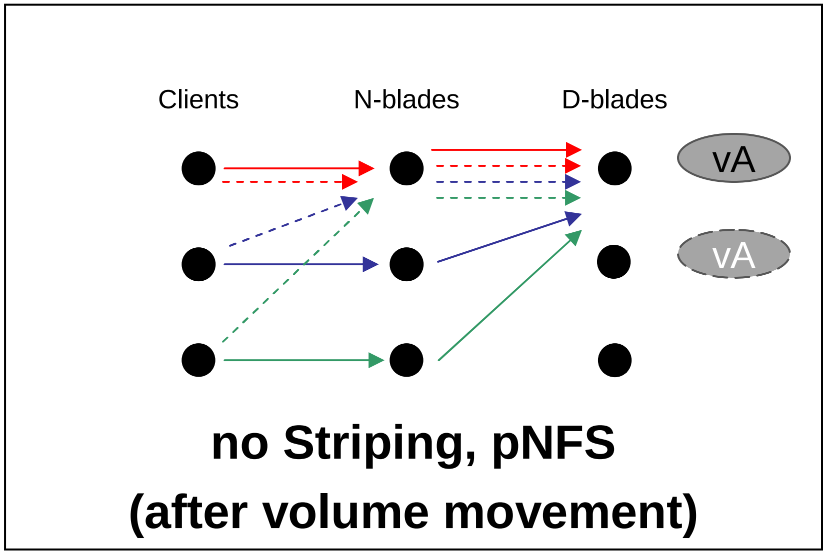
pNFS for Flexible Volumes

- Round Robin DNS - pick a network IP from a set of Lif's
- With pNFS, MDS could be any of the Lif's on any node
- So, there is a problem to solve i.e., find direct network path to the D-blade hosting Unstriped WAFL volume (data only for now)
- Basically, ONTAP finds a Lif that is hosted on the same D-blade as the flexible volume
- Enables metadata and data network path split
- Volume/network movement works fine as well, thanks to pNFS layout recalls
- A glorified **data referral** i.e data path changes as cluster management constructs work

Unstriped volumes (pNFS)



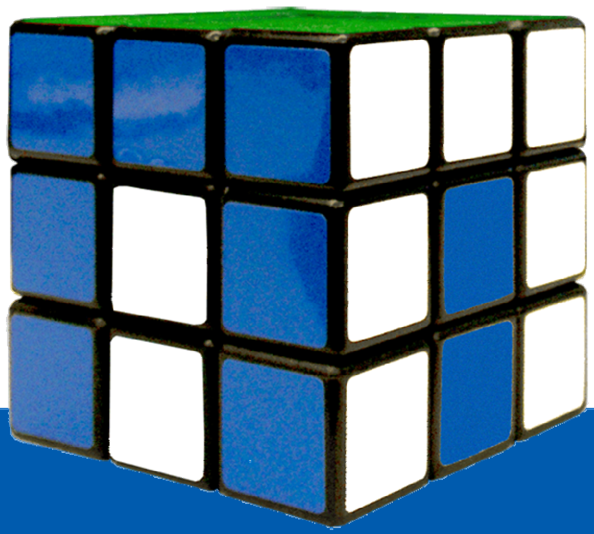




Protocol mechanics

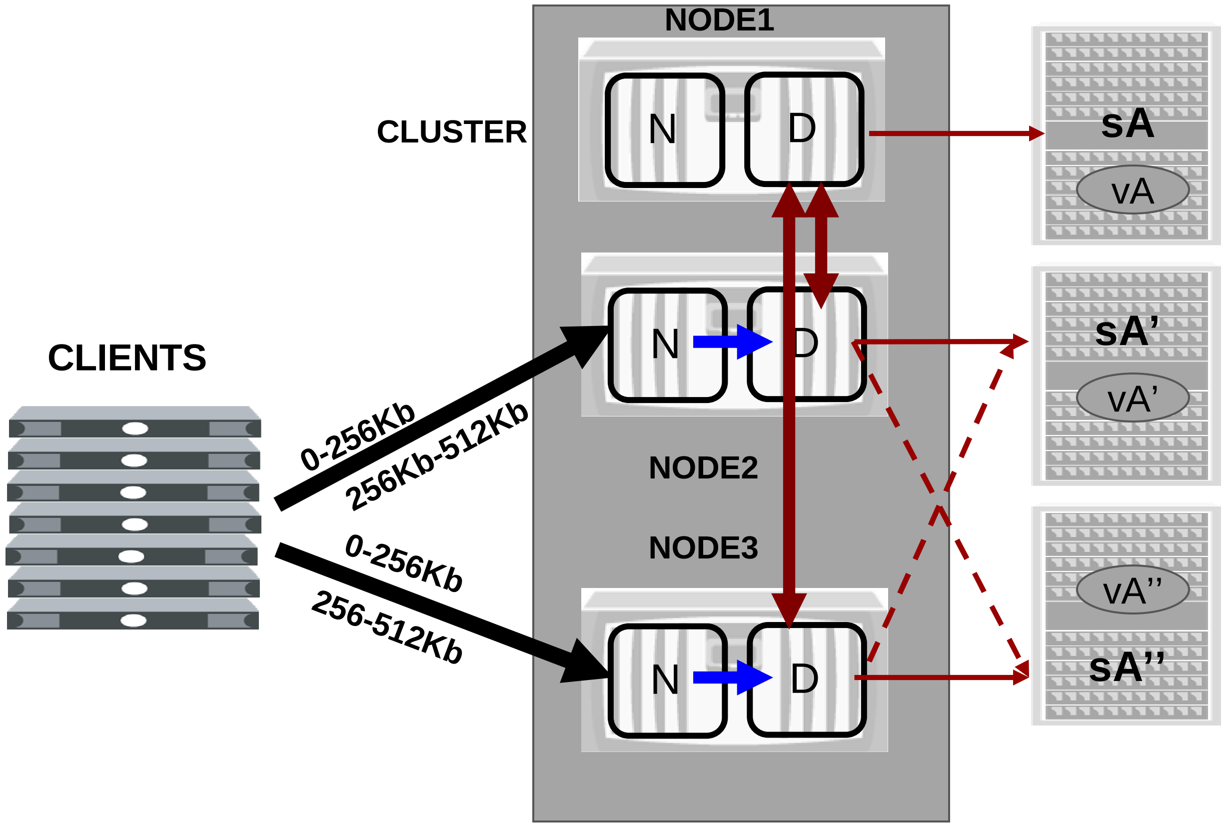
- No. of pNFS data servers = 1
  - No. of device addresses is also the same
  - When trunking is implemented, more addresses will be sent for every data server
- pNFS stripe (indices) count = 1
  - Yes, our GETDEVICEINFO response for Unstriped volumes is really small
- pNFS first stripe index
  - Same for each file, as all the files located on the same volume
  - Value 0 for each file, points to the only element referred above

Scale Out WAFL Entities



Scale Out Entities

- A Flexible volume (vA) has other corresponding scaleout entities (vA’, vA’’)
- vA’, vA’’ are essentially static or on-demand copies of the corresponding flexvol (vA)
- Use Cases: Mirrors, Caching (Tiering clusters)
- N-blade always routes client request to the local copy of volumes



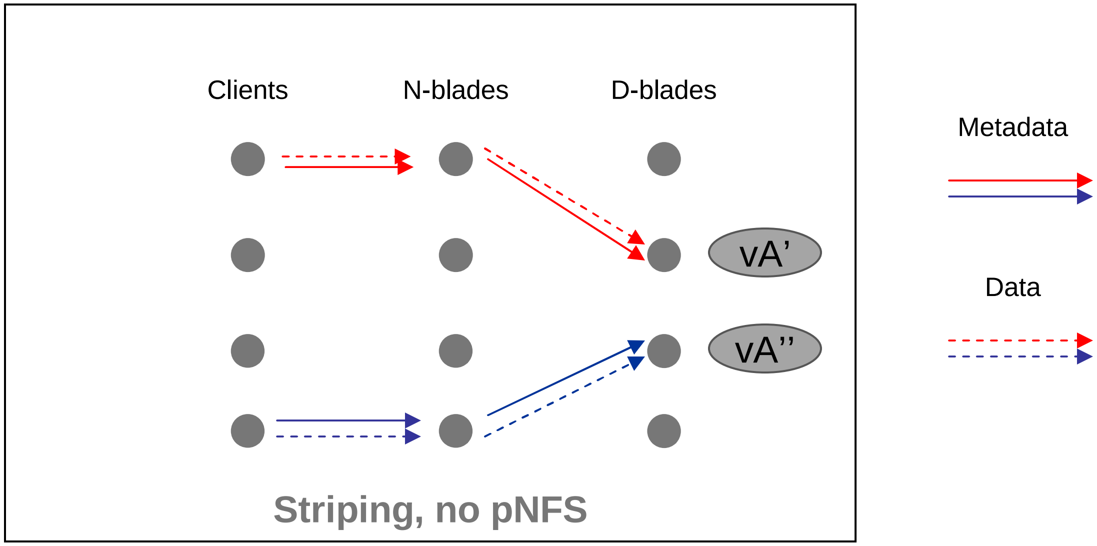
Scaleout Entities

- Basically, scale out entities leverage the cluster and help deal with bottlenecks
  - Mirrors to share load across the cluster
  - Caches to tier storage, within cluster and between clusters
- Technically, these scale out entities are static or on-demand copies of a source volume (to be tiered, or shared)
- The metadata and data is not striped on disk
- However, consistency is maintained by backend across primary source and copies
- Opportunity with pNFS
  - Stripe client requests across these entities for a volume
  - Help increase utilization of the cluster itself
  - Help client overcome single-server bottleneck and parallelize I/O requests
  - Help warm-up cluster tiered caches automatically

Scaleout Entities Terminology

- Constituent Count
  - Number of static or on-demand constituents
  - Logically ID'd from 0 to N - 1
- Stripe Width (pNFS stripe unit size)
  - Maximum number of data bytes written and read to and from a specific member-volume
  - Could be anything, as they are not inherently striped on-disk
- Striping Table (pNFS stripe indices array)
  - Size fixed at 4096 entries – irrespective of stripe count
  - Yep, Same as what we have for striped volumes
  - Elements in the table will be logical ID's assigned to constituent volumes
  - Proprietary algorithm – Not to be disclosed

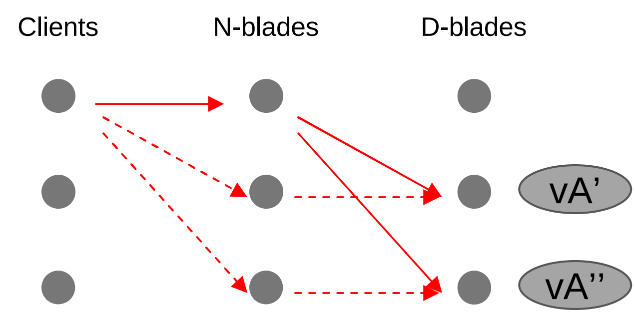
Scaleout WAFL Entities (no pNFS)



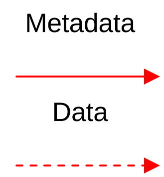
pNFS for Scale Out WAFL

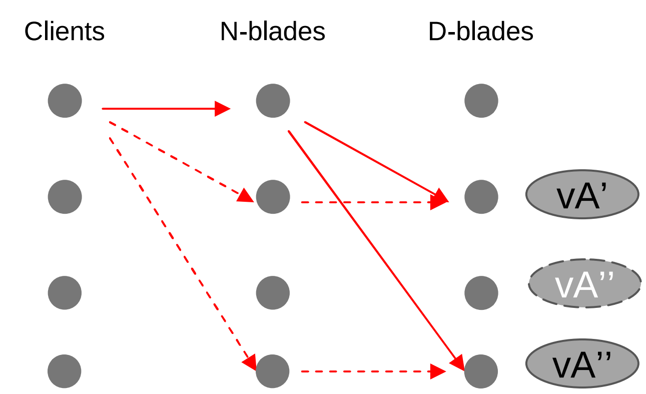
- Round Robin DNS - pick a network IP from a set of Lif's
- With pNFS, MDS could be any of the Lif's on any node
- Basically, ONTAP finds an IP that is local to each of the scale out entity for a volume
- If we project it as just another Unstriped volume, client does not have a benefit of parallelism
- So, there is a problem to solve i.e., spread each clients request across each of the D-blade hosting Scaleout WAFL entity
- Volume/network movement works fine as well, thanks to pNFS layout recalls
- Support sparse layouts
  - Routing within the cluster backend is based on logical file offset

Scale out WAFL (pNFS)



Scaleout, pNFS





Scaleout, pNFS (after constituent movement)

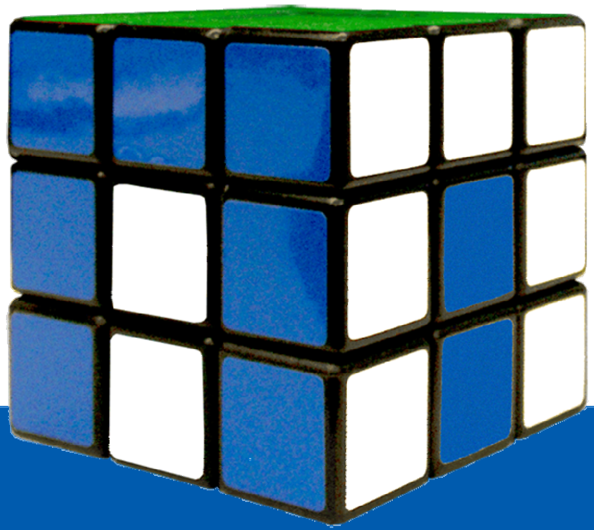
Protocol mechanics

- No. of pNFS device addresses = constituent count
  - Constituents mean scale out entities corresponding to a volume
  - Of course, does not consider multipathing
  - When trunking is implemented, more addresses will be sent for every data server
- pNFS stripe indices count = striped file system striping table size
  - i.e. always 4096 stripe indices (same as striped volumes)
  - Yes, our GETDEVICEINFO response is large
  - However, since there is no inherent file system striping, we could consider playing with various algorithms
  - Potentially even looking at multi-segment layouts with fancy configurations
- pNFS first stripe index
  - Varies for each file
  - An index into the striping table based on a proprietary hash
  - Thus, different file I/O get directed to different constituents

NetApp

Thank You !

Questions?



References

- pNFS problem statement
  - http://www.pdl.cmu.edu/pNFS/archive/gibson-pnfs-problem-statement.html
- NFSv4.1 Draft
  - http://tools.ietf.org/html/draft-ietf-nfsv4-minorversion1-29
- pNFS Tech ONTAP article
  - http://www.netapp.com/us/communities/tech-ontap/pnfs.html
- Clustered ONTAP pNFS Server Talk at Connectathon 2009
  - http://www.connectathon.org/talks09/ClusteredONTAPpNFSCthon2009.pdf
- Mike Eisler’s metadata striping proposal
  - http://tools.ietf.org/id/draft-eisler-nfsv4-pnfs-metastripe-01.txt
- Mike Eisler’s Blog
  - http://blogs.netapp.com/eislers_nfs_blog/