Exporting Storage Systems in a Scalable Manner with pNFS

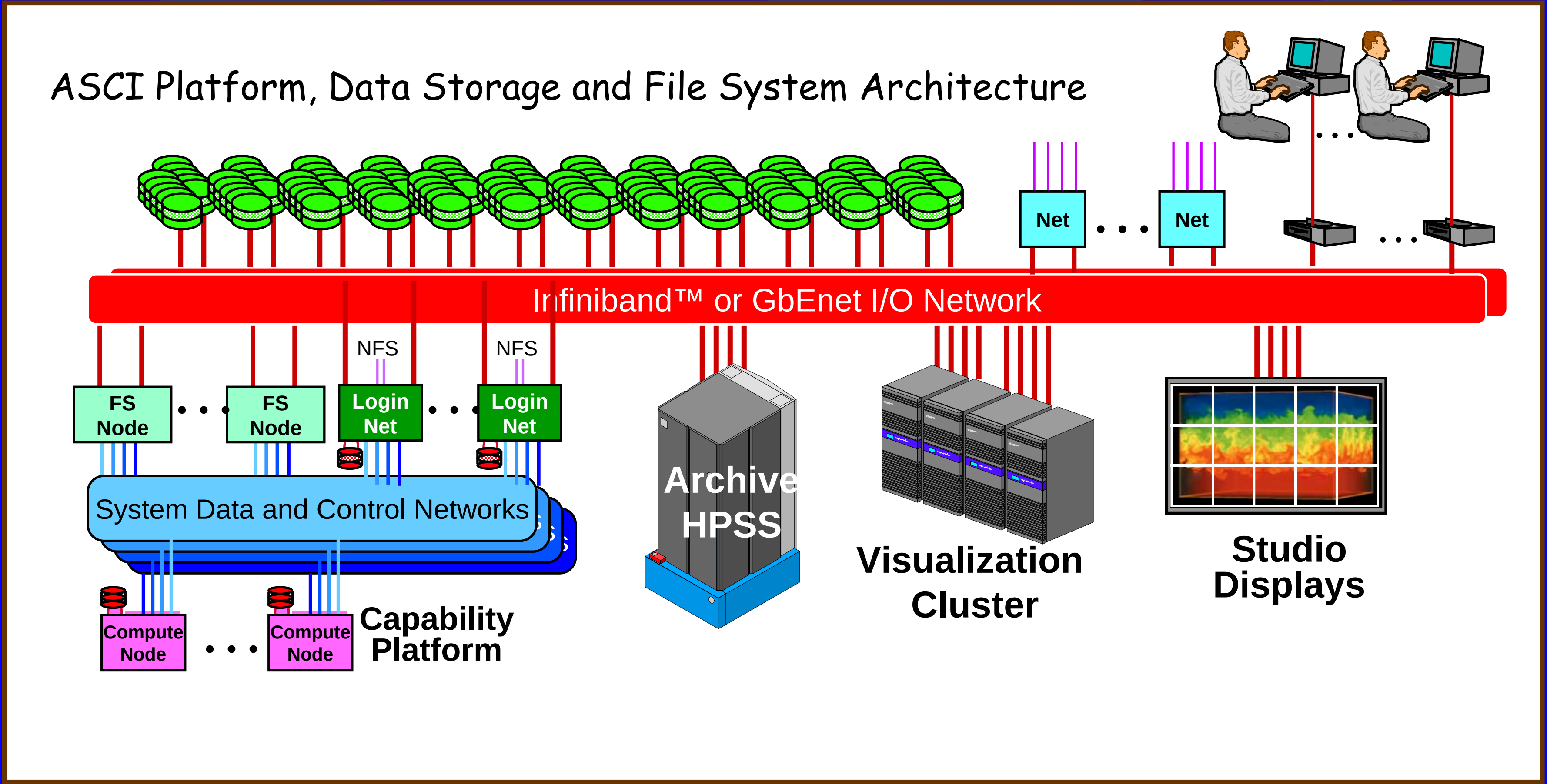
Dean Hildebrand

Advisor: Peter Honeyman
Center For Information Technology Integration
University of Michigan

Outline

- Motivation
- pNFS Overview
- pNFS Prototype
- Experiments

Motivation: ASCI Example

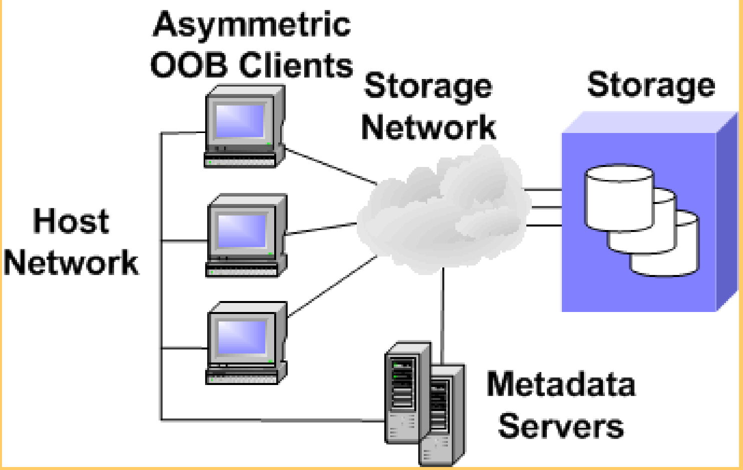


from ASCI Technology Prospectus, July 2001

Motivation: HPC Out-Of-Band File Systems

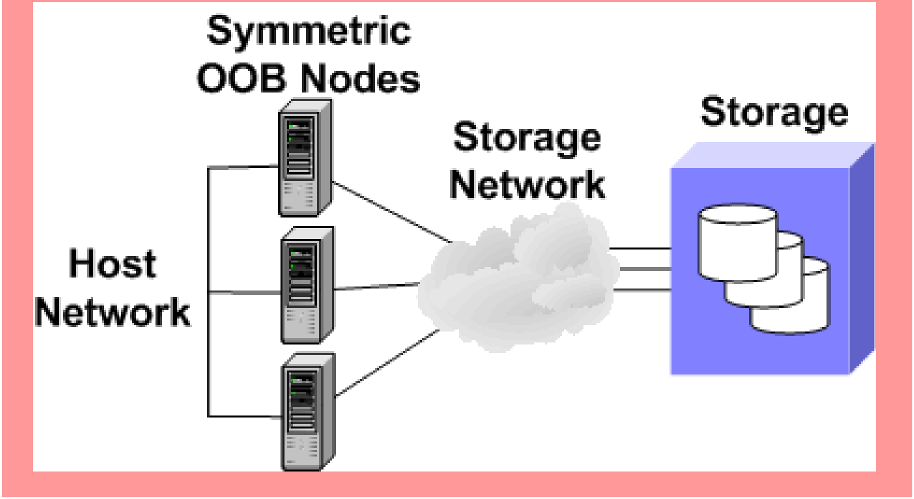
Asymmetric

- Direct storage access
- Separate metadata server(s)
- Object Based:
  Lustre, Panasas ActiveScale
- Block Based:
  EMC High Road, IBM SAN FS
- File Based: NASD NFS



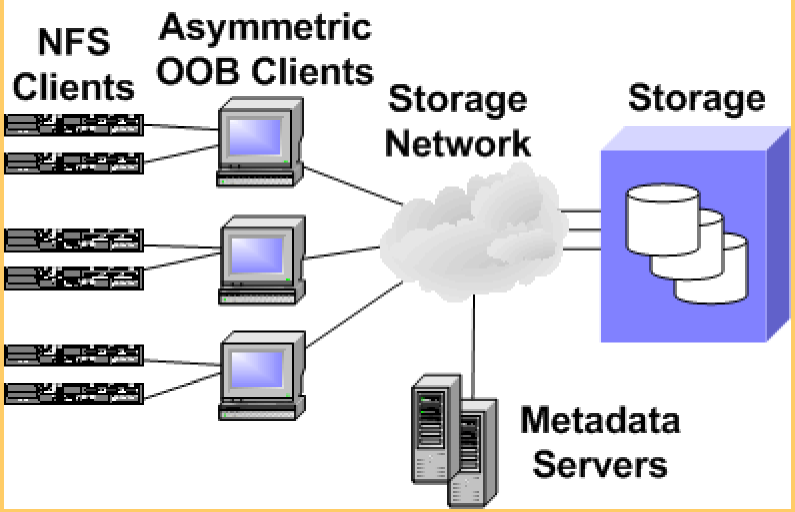
Symmetric

- Direct storage access
- Each node is a fully capable client and metadata server
- Ex: IBM GPFS, Redhat GFS, Polyserve Matrix Server

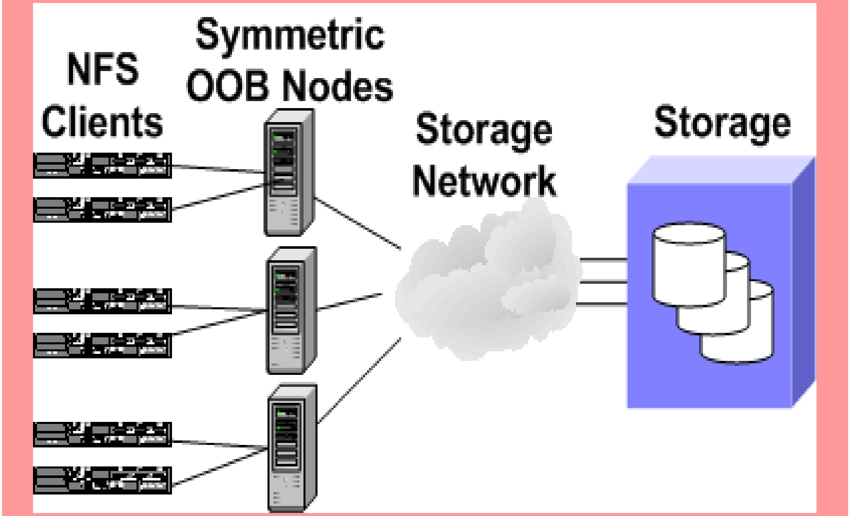


Motivation:
NFS and OOB File Systems

Asymmetric OOB File Systems



Symmetric OOB File Systems



Issues:

- Single Server Bottleneck
- Extra level of indirection

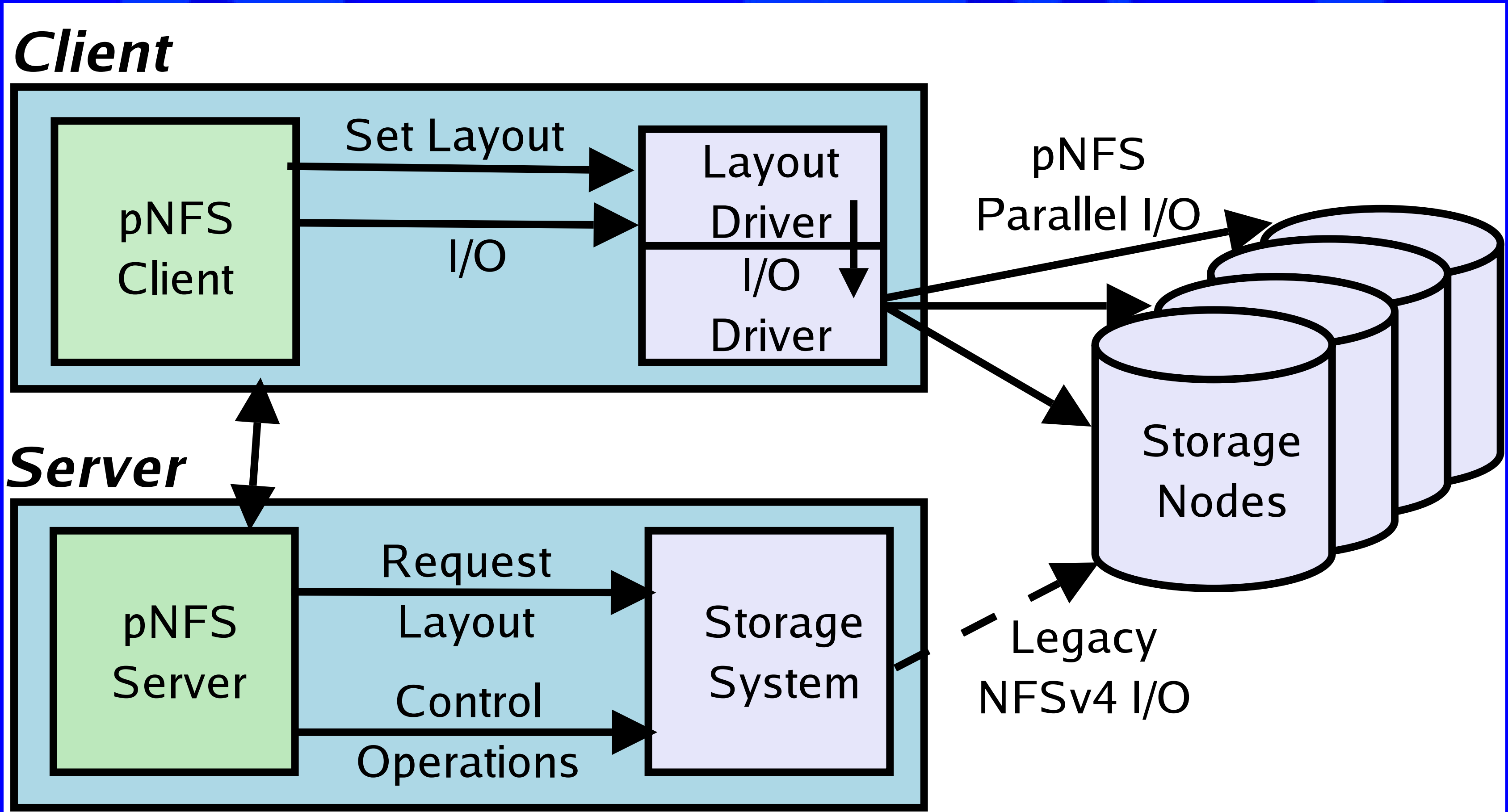
Problem Statement

- HPC OOB File System Issues
  - Interoperability
  - Cost
  - Proprietary
  - Remote access performance (NFS, CIFS)
- NFSv4 Issues
  - Many-to-one relationship of NFSv4 clients to server
  - Cannot scale with exported storage

pNFS

- IETF NFSv4 protocol extension
- Scale with underlying file system
  - Clients performs direct I/O to storage
  - Escape NFSv4 block size restrictions
  - Single file access
- File system independent
  - Support all layout maps (block, object, file, etc)
  - Create global namespace of disparate HPC file systems
- Interoperate with standard NFSv4 clients and servers
  - Storage still accessible through NFSv4 server
- Operate over any NFSv4 infrastructure
- Support existing storage protocols and infrastructures
  - Examples: SBC on Fibre Channel on iSCSI, NFSv4

pNFS Architecture



NFSv4 Extensions (1/2)

- LAYOUTGET operation
  - Retrieves file layout information
  - Valid until returned or file close

Arguments:

- File handle
- Offset
- Extent
- I/O type
- State identifier
- Maximum count and cookie

Results:

- Offset
- Extent
- Cookie
- **Opaque** layout

- Other Operations:

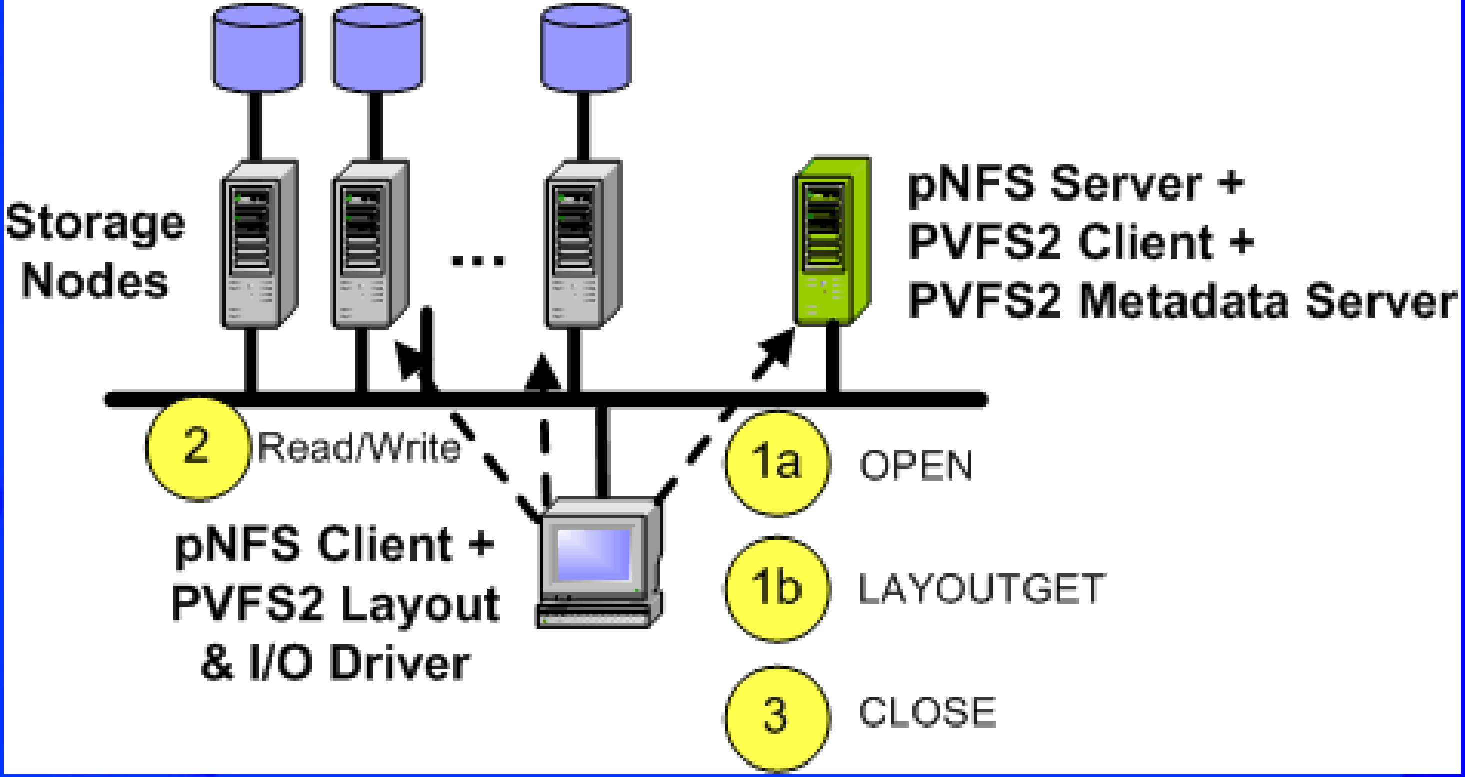
LAYOUTCOMMIT, LAYOUTRETURN, CB_LAYOUTRECALL, GETDEVICEINFO, GETDEVICELIST

NFSv4 Extensions (2/2)

- Layout Driver
  - Interprets layout information
  - File layout protocol specific
  - Support standard and non-standard storage protocols
  - Multiple per client
  - Multiple per file system
    - LAYOUT_CLASSES file system attribute
- I/O Driver
  - Performs raw I/O to storage nodes
  - Examples: Myrinet GM, Infiniband

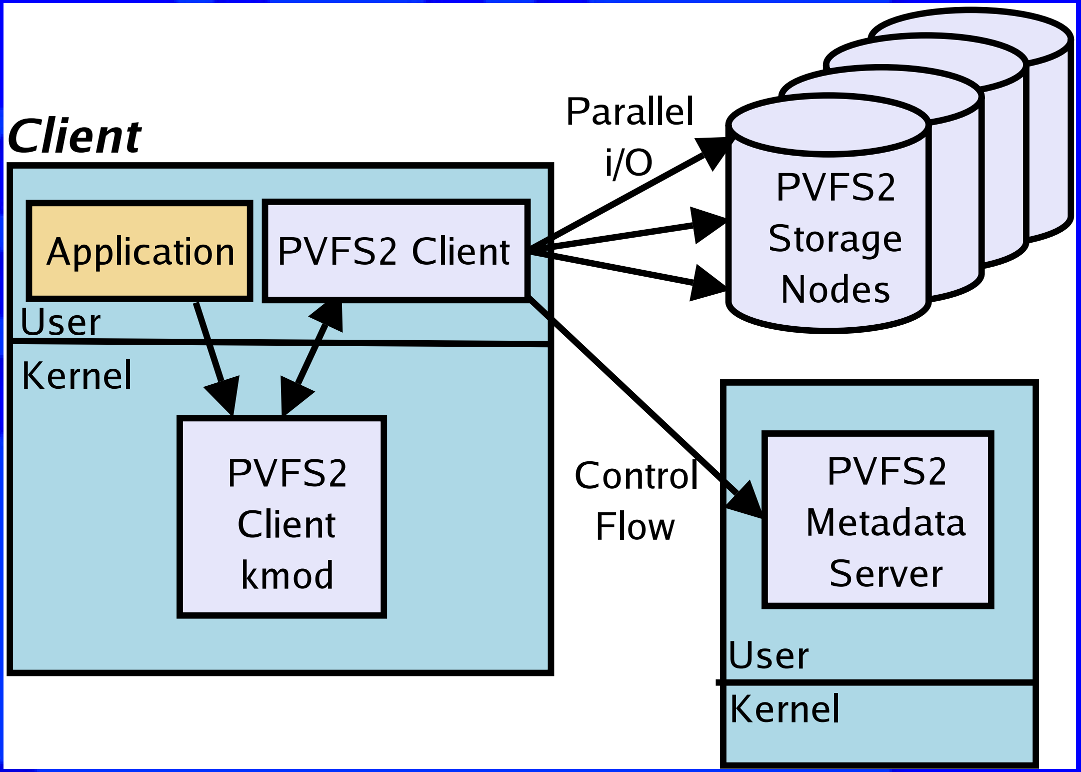
pNFS Prototype

- NFSv4 on Linux 2.6.9-rc3
- Exports PVFS2 1.0.1 OOB file system


Storage Nodes
...
pNFS Server + PVFS2 Client + PVFS2 Metadata Server
2 Read/Write
1a OPEN
1b LAYOUTGET
3 CLOSE
pNFS Client + PVFS2 Layout & I/O Driver

PVFS2 Overview

- Developed at Argonne National Laboratory
- Algorithmic file layout
  - Currently supports round robin striping
  - LAYOUTCOMMIT, LAYOUTRETURN, CB_LAYOUTRECALL not required
- No locking subsystem
- No data caching



File Layout Retrieval Mechanism

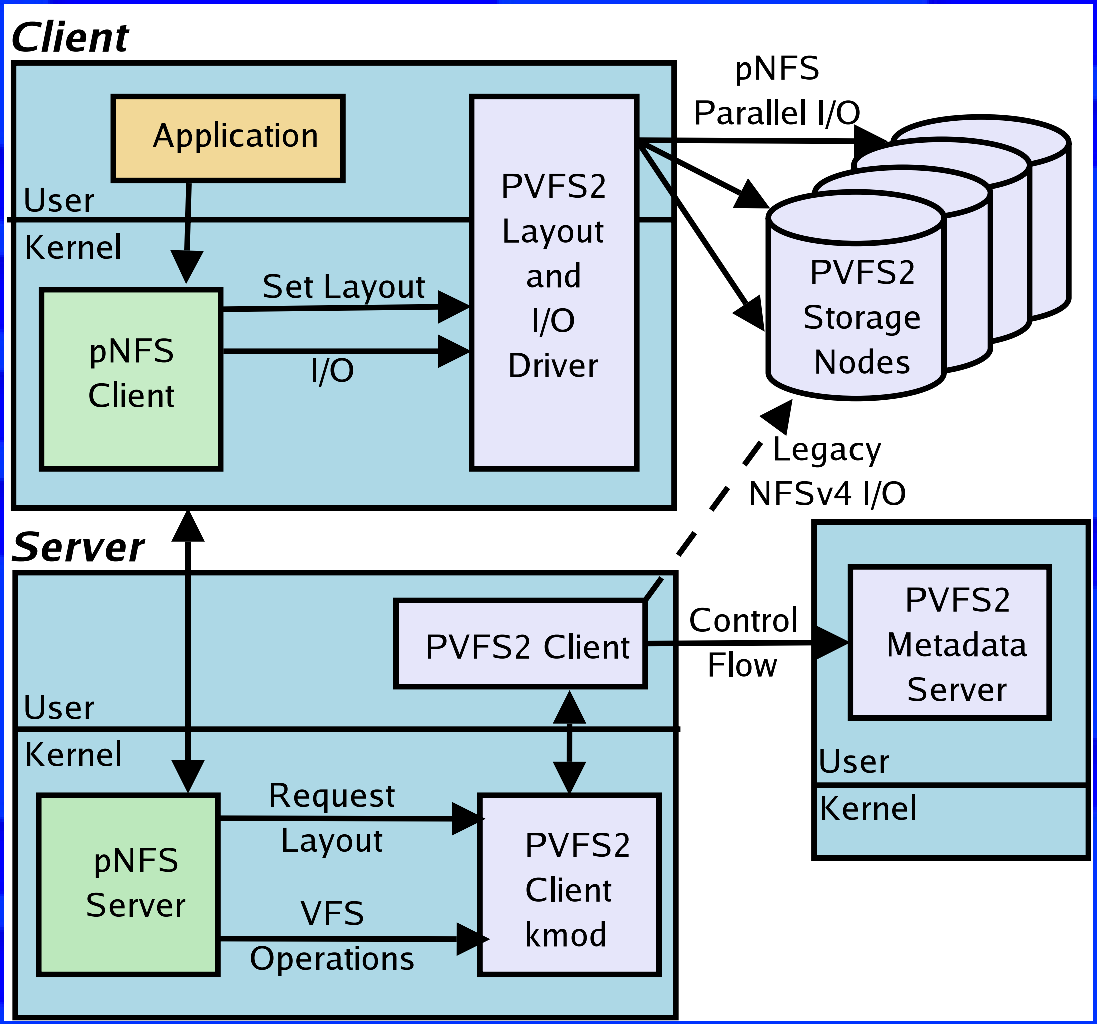
- LAYOUTGET sent on either:
  - Application read/write request
  - File open
- pNFS server uses ioctl to retrieve layout
- Client caches layout information
- PVFS2 file layout consists of:
  - Size of layout
  - File system id
  - Number of file handles
  - Set of file handles (includes storage node info)
  - Distribution id
  - Distribution parameters, e.g., stripe size

PVFS2 Layout and I/O Driver

- Registers on load
- Standard Linux API
  - file_operations structure
- Ioctl injects opaque layout
- I/O Driver uses specialized protocol with TCP/IP

```
ssize_t (*read)  (struct file* filp, char __user* buf,
                   size_t count, loff_t* pos);
ssize_t (*write) (struct file* filp, const char __user* buf,
                   size_t count, loff_t* pos);
int     (*ioctl) (struct inode* inode, struct file* file,
                   unsigned int cmd, unsigned long arg);
```

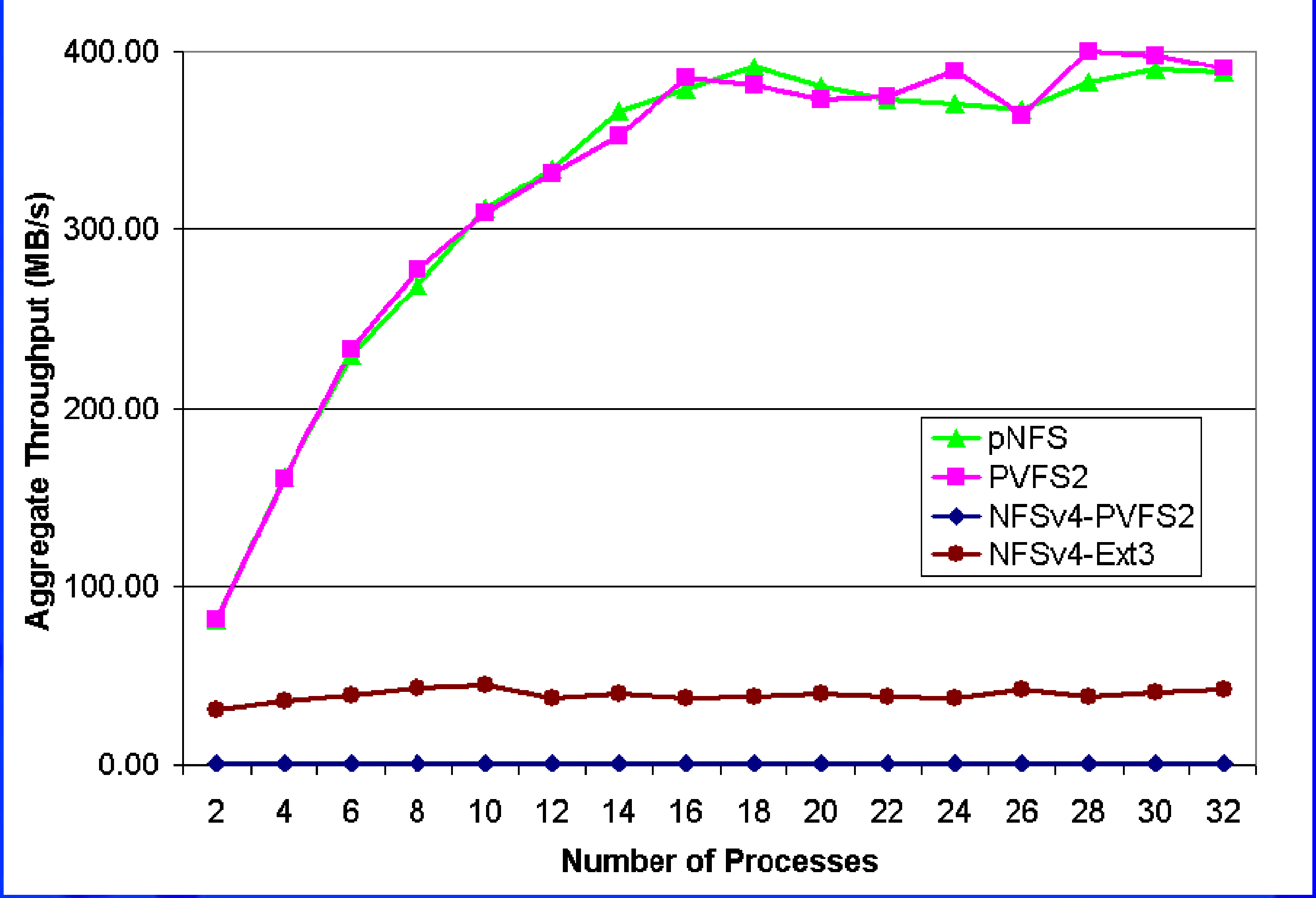
pNFS Prototype Architecture



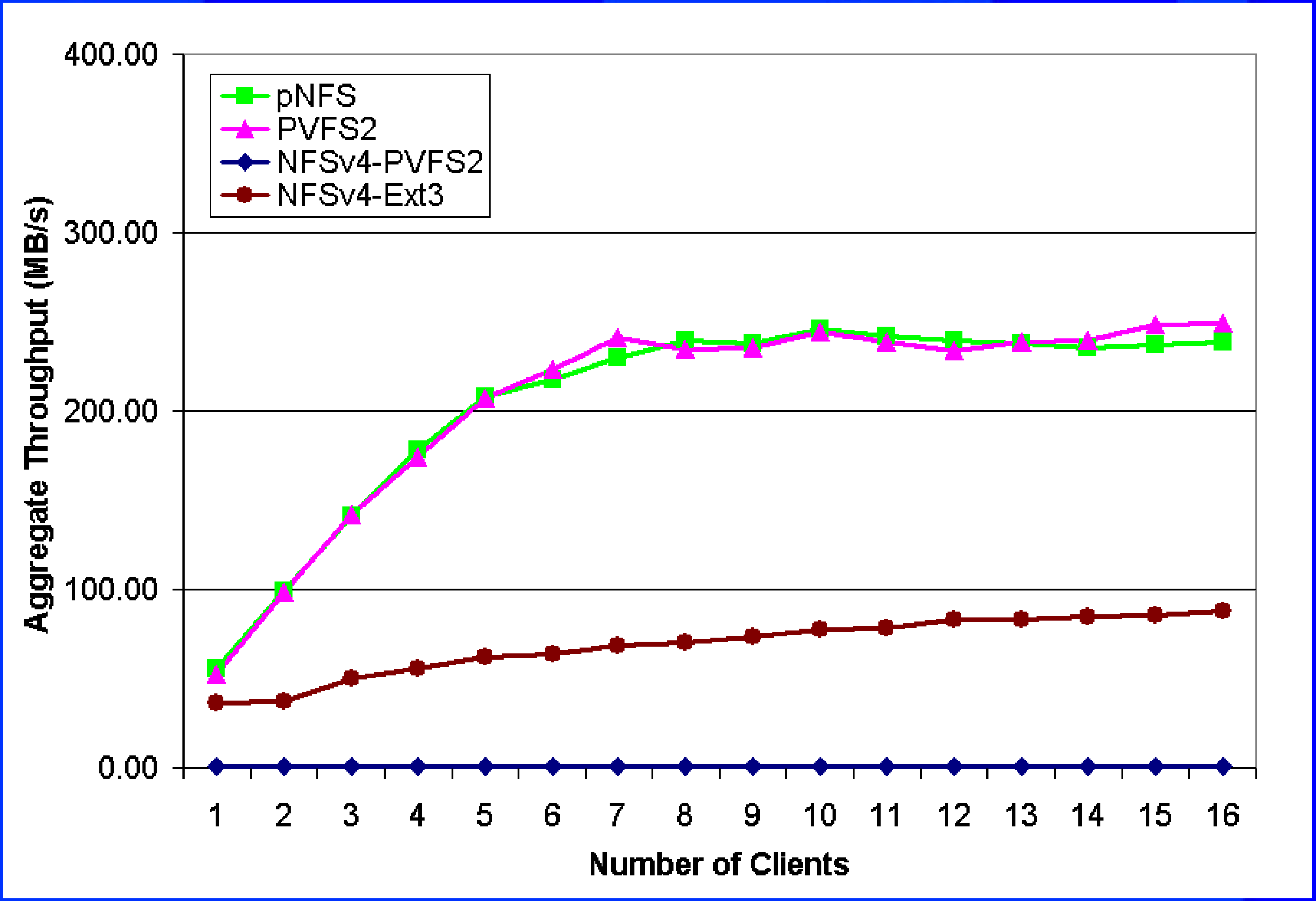
Experimental Setup

- Forty 2GHz dual Opteron nodes with 2GB RAM
- PVFS2 storage nodes utilize software RAID 0 with four ATA drives
- PVFS2 1.0.1 with 16 storage nodes, 1 metadata server
    - pNFS server, PVFS2 client, and PVFS2 metadata server on single machine
- Maximum 23 clients (46 processes)
- Gigabit Ethernet
- Linux 2.6.9-rc3
- IOZone to measure aggregate I/O throughput
- Warm read cache
- Writes are immediately committed to disk

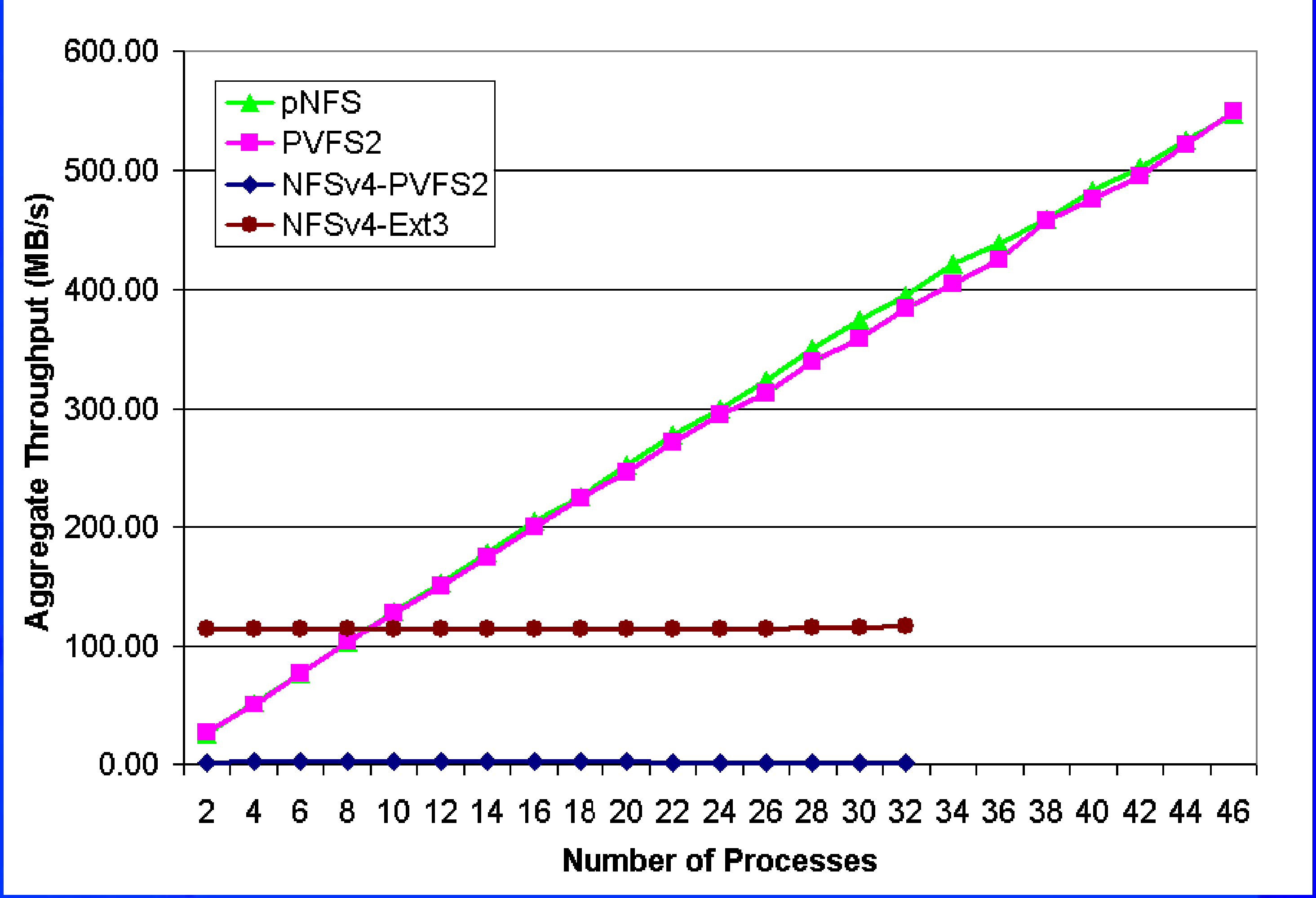
Write Experiments - Separate Files



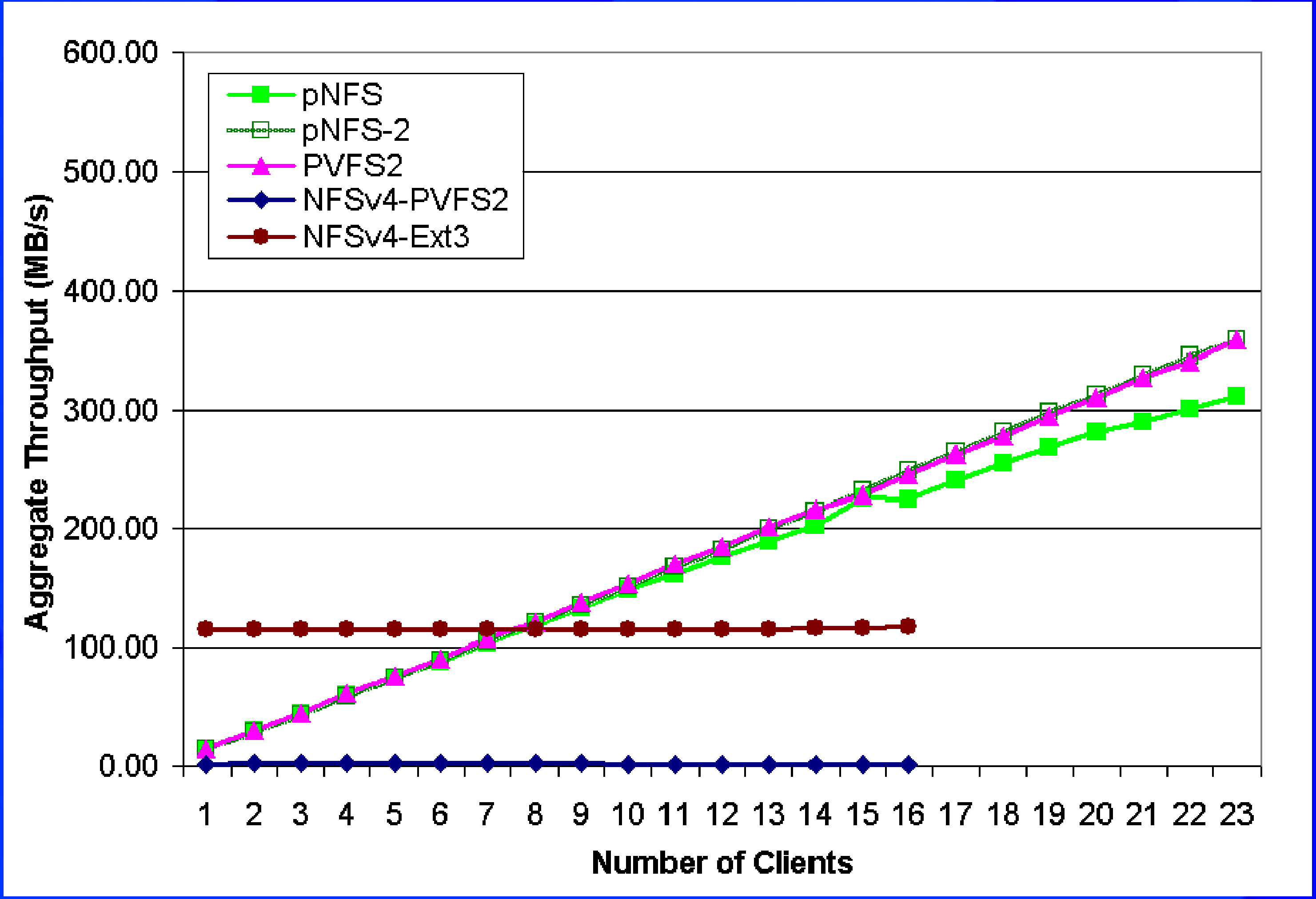
Write Experiment - Single File



Read Experiment - Separate Files



Read Experiment – Single File



Discussion

- Layout and I/O drivers enable scalability
  - Avoid indirection penalty and single server bottleneck of NFSv4
- Standard I/O protocol reduces development and support cost
  - Validated opaque layout design
  - Opaque file layout information and standard layout driver interface enables underlying file system independence
  - Obviates proprietary file system client
- A single pNFS client can interact with multiple parallel file systems on multiple platforms

Issue:

- Scalability of LAYOUTGET operation

Related Work

- Scalability of NFS
  - Bigfoot-NFS, Expand, nfsp, NASD NFS
- EMC's HighRoad
  - NFS- or CIFS-based control channel and block-based data channel
  - Facilitates data sharing
  - Limited to block-based EMC Symmetrix storage system
- Storage Resource Broker (SRB)
  - Lacks parallel data access to multiple storage endpoints
  - Lacks integration with local file system
- GridFTP
  - No consistency protocol
  - Lacks integration with local file system

Future Work

pNFS Protocol :

- Locking
- Security
  - Different issues for file-, object-, and block-based systems
- Layout management and delegation
- Client data cache consistency

pNFS/PVFS2 Prototype:

- MPI-IO support

General:

- Large file layouts
- Strided file layouts
- Multiple pNFS servers

More Information

- pNFS –
  http://www.pdl.cmu.edu/pNFS/
- NFSv4 –
  http://www.nfsv4.org/
- CITI Linux NFSv4 Projects –
  http://www.citi.umich.edu/