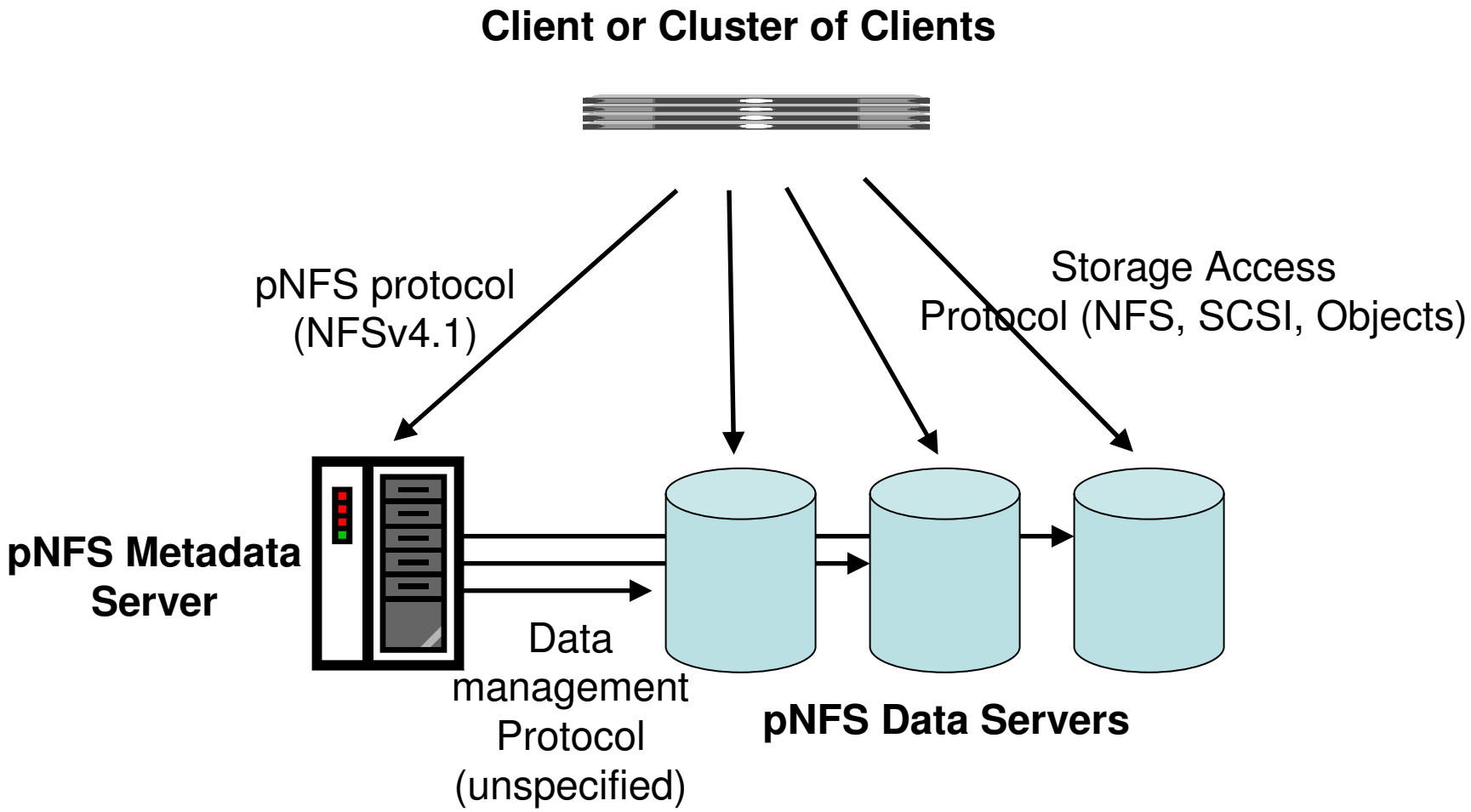
pNFS File Layout

Linux Storage and Filesystem Workshop
February 25, 2008

Tom Talpey
Ricardo Labiaga
Trond Myklebust

pNFS Architecture



pNFS users

- Gaining wide interest in HPC and cluster computing
- Stripe-aware parallel applications, parallel client clusters
- Linear file performance scaling, especially
- RDMA-capable cluster fabrics
  - NFS/RDMA transport offers Nx10Gb+ pipes
  - One-wire storage/cluster solution

pNFS deployment

- No natively pNFS-aware filesystems
- Typically cluster backends, with pNFS export
- Which works fine, but why put a cluster in front of a cluster?
  - Difficult for user adoption
- Seek easy-to-deploy parallel solution

spNFS

- “Simple pNFS” – NetApp developed/contributed
- Metadata server (only) with static layouts
- Implemented in **user space**
- Hooks incoming pNFS-eligible client requests
- Consults /etc/spnfsd.conf to find Data Servers
- Uses NFS to create and manage stripes in private directories on them
- Returns these layouts via pNFS
- Leverages existing pNFS server code

spNFS Advantages

- Simple!
- No (zero) changes to Data Servers
  - Unmodified NFSv3 servers, for example
  - Welll… a small stateid check must be added if NFSv4
  - Minimal changes to MDS, too
- Completely filesystem-agnostic
  - Any back-end filesystem can work
  - Cluster filesystems provide some additional flexibility (layout management)
- Linear scaling
  - Full benefit of pNFS

spNFS Disadvantages

- Simple!
- Manual layout configuration
- Difficult to restripe/modify layouts
- No client writethrough-MDS
  - Possible, but painful
- Clumsy hooks in MDS filesystem/exports

Export hooking (XXX!)

```
@@ -436,6 +437,45 @@ static int check_export(struct inode *in
+       /*
+        * spnfs_enabled() indicates we're an MDS.
+        */
+       if (spnfs_enabled()) {
+                       printk("set spnfs export structure...¥n");
+                       if (!inode->i_sb->s_export_op->layout_type)
+                               inode->i_sb->s_export_op->layout_type = spnfs_layout_type;
+                       if (!inode->i_sb->s_export_op->get_devicelist)
+                               inode->i_sb->s_export_op->get_devicelist = spnfs_getdevicelist;
+                       if (!inode->i_sb->s_export_op->get_deviceinfo)
+                               inode->i_sb->s_export_op->get_deviceinfo = spnfs_getdeviceinfo;
+                       if (!inode->i_sb->s_export_op->propagate_open)
+                               inode->i_sb->s_export_op->propagate_open = spnfs_open;
+                       if (!inode->i_sb->s_export_op->layout_get)
+                               inode->i_sb->s_export_op->layout_get = spnfs_layoutget;
+                       if (!inode->i_sb->s_export_op->layout_return)
+                               inode->i_sb->s_export_op->layout_return = spnfs_layoutreturn;
+                       printk("Done setting spnfs export structure¥n");
+       } else {
+                       printk("%s spnfs not in use¥n", __FUNCTION__);
+       }
+       /*
+        * get_state is needed if we're a DS using spnfs.
+        */
+       if (!inode->i_sb->s_export_op->get_state)
+                       inode->i_sb->s_export_op->get_state = spnfs_get_state;
```

Where we want it to be

- Goal: to generalize the spNFS approach
- Increase filesystem agnosticism
- Use arbitrary layout management
  - i.e. on arbitrary back-end filesystems
  - Whether they're layout-aware or not

Do an spNFS-FS?

- Stacking pseudo-FS
- Layout driver hook
  - Parses layout requests by probing lower FS
  - Or, making it up! (like current code)
  - Forwards most other requests
- Doesn't require lower FS to change for pNFS
- But, no general way to query layout
  - > Changes are in spNFS-FS, ok

Alternatives?

- Add layout op(s) to FS export API?
  - Requires changes to eligible filesystems
- Add layout op(s) to VFS switch?
  - Requires changes to all filesystems
- Attempt to create a layout library/API?
  - Is this even possible?