Parallel NFS (pNFS)

Garth Goodson

Introduction

- **Proposition:**
  - Extend NFSv4 to support data parallelism
- **Goals:**
  - Remove single server bottleneck
  - Standardize client interface for parallel data access
  - Meet needs of HPC, and Linux cluster communities
- **Leverage NFSv4 implementations**
  - Add small set of protocol extension (pNFS)
  - Provide protocol for parallel data access by clients

Background

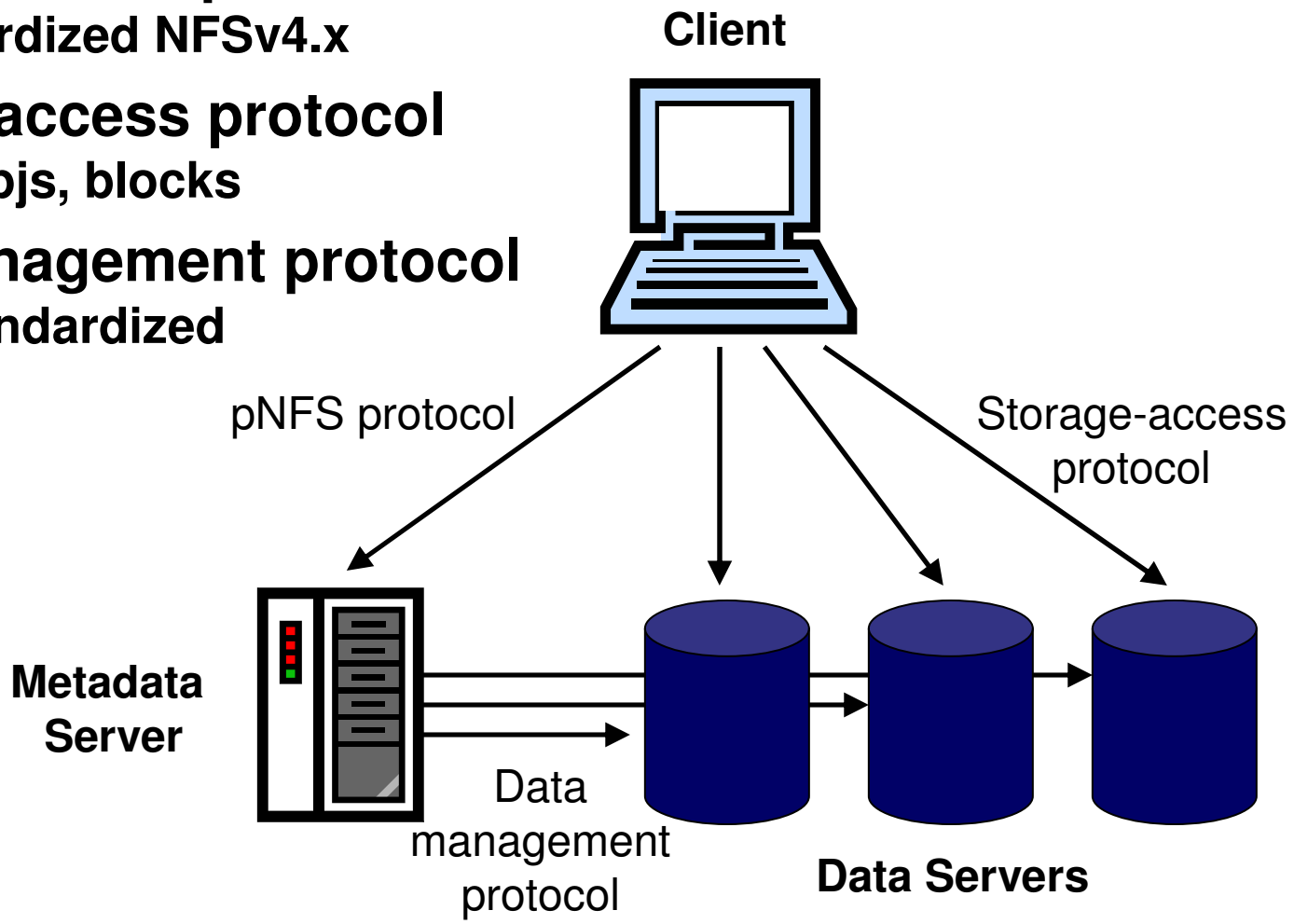
- Started as discussion at CITI Dec. 2003
- Since then many informal meetings
- Major industry interest
  - Panasas, EMC, Sun, IBM, NetApp, …
- Introduced to NFSv4 WG Nov. 2004
  - Problem statement
  - Requirements document
  - Operations proposal

Parallel NFS (pNFS)

- Separates data and metadata accesses
  - Standardized metadata protocol
  - Different variants for storage access protocol
    - Files, Blocks, Objects
- Files striped across multiple servers
  - High aggregate bandwidth
  - Data can be accessed in parallel (for a single file)
  - Should allow for multiple backend storage protocols
- Standardized client
  - Proposed as IETF NFSv4 minor version
  - Can fall back to regular NFSv4
  - Should promote interoperability

pNFS Architecture

- **pNFS metadata protocol**
  - Standardized NFSv4.x
- **Storage-access protocol**
  - files, objs, blocks
- **Data-management protocol**
  - not standardized



pNFS Requirements

- **Scalability**
  - Extend parallelism to client
  - Directories are not distributed, files are
- **Interoperability**
  - Client must be able to fall back to standard NFSv4
  - Server must support standard NFSv4 clients
  - Storage-access protocols must be well-defined
- **Concurrent-sharing**
  - Shared direct access to storage by multiple clients
  - Not further addressing caching/coherency

pNFS Requirements

- **Error recovery**
  - Should be able to fall back to something ‘simple’
    - At the cost of performance
- **Security**
  - Should be comparable to NFSv4 security
  - Files protocols can use V4 w/o impacting security
  - Other protocols control security outside of V4 spec
    - Object protocols can use capabilities
    - Block protocols must rely on SAN-based security
      - Changes security model to include client

pNFS metadata protocol extensions

- A handful of new ops for manipulating layouts
  - `LAYOUTGET, LAYOUTCOMMIT, LAYOUTRETURN`
- Some ops for mapping devices to IDs
  - `GETDEVICEINFO, GETDEVICELIST`
- A few attributes for determining pNFS support
  - `LAYOUT_CLASSES, LAYOUT_TYPE, LAYOUT_HINT`
- And a callback
  - `CB_LAYOUTRECALL`
- Also, IANA consideration for defining types…

What is a layout?

- **Layouts define the data mapping to the client**
  - **E.g., enumerates servers data is mapped across**
- **Layout based on storage access protocol used**
  - **May be small/static for file/obj-based protocols**
  - **May be large/dynamic for block-based protocols**
  - **Identified by class and type (sub-class)**

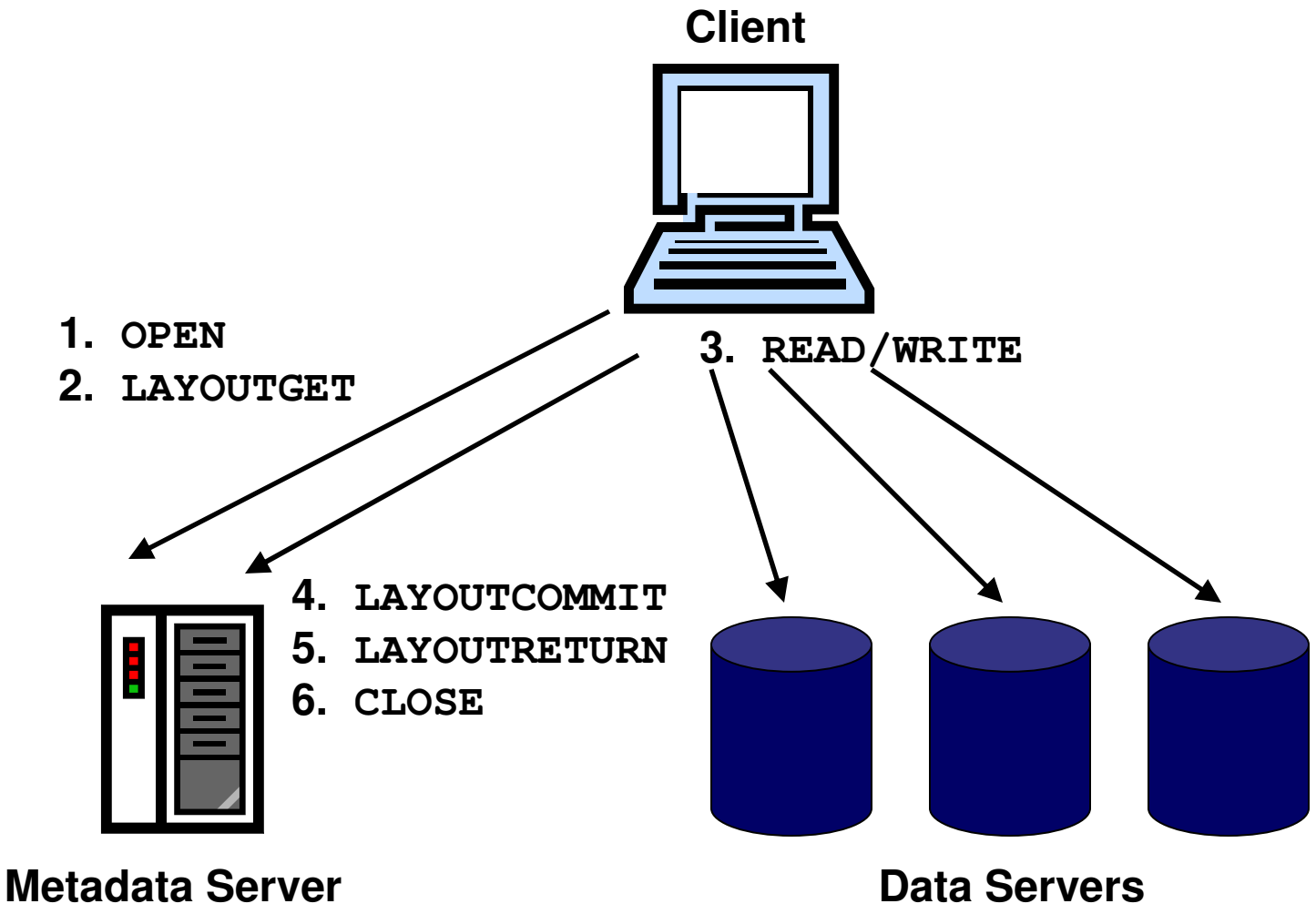
Example:

```
union pnfs_layout4 switch (pnfs_layoutclass4 class) {
        case LAYOUT_FILES_NFSV4:
                pnfs_nfsv4_layouttype4 file_layout;
        default:
                opaque layout_data<>;
};
```

Layouts

- Layouts may be delegated to clients
  - Not for caching, for out-of-band data modification
  - Sharing mode specified in `LAYOUTGET`
    - `IOMODE: READ/WRITE/RW`
    - `SHAREMODE: SHARED/EXCLUSIVE`
  - Layout updated on `LAYOUTCOMMIT`
    - Can also update size attribute (EOF)
    - Ensures size attribute is consistent
  - Layouts may be recalled (`CB_LAYOUTRECALL`)
  - Return layout with `LAYOUTRETURN`

Basic operation flow



Our prototype

- Basic premise: start simple
  - Basic prototype to show parallel access
  - Not concerned w/locking, delegations, security, etc.
- Based on pNFS draft operations proposal
- Using NFSv4 files storage-access protocol
- Implemented in NetApp filer
- Sun involved in client prototype work
  - Interoperability demonstrated at connectathon

Prototype: requirements

- NFSv4 files based layout for striped data
- Single `OPEN/REMOVE` at the metadata server
- Size attribute made visible on `LAYOUTCOMMIT`
- Simple mgmt protocol, using stock NFSv4

Prototype: non-requirements

- ACLs on data servers
- Mandatory locking on data servers
- Implicit lease renewal based on data I/Os
- Immediate reflection of attrs based on I/Os

Prototype: file layout definition

- Layout is an array of device layouts & stripe size
  - `dev_id` names data server (shorthand uint32)
  - `fh` names file on data server
  - `stateid` state required for data access, set to all 0's

```
struct pnfs_nfsv4_layout {
        pnfs_deviceid4      dev_id;
        nfs_fh4             fh;
        stateid4            stateid;
};

struct pnfs_nfsv4_layouttype4 {
        uint64_t            stripe_size;
        pnfs_nfsv4_layout   dev_list<>;
};
```

Prototype: handling of file layouts

- Option on filer sets default layout
  - Stripe size and set of data servers
  - Easily changed
  - Inherited by metadata file upon creation
- Layout stored as named attr on metadata file
  - Default layout copied into named attr on creation
  - Metadata file is otherwise empty, w/correct attrs
- Client should not modify layout
  - In future, restriping may modify layout
- Delegated layouts not yet supported

Prototype: storage access protocol

- Storage access uses standard NFSv4
  - No `OPEN`, `LOOKUP`, `SETATTR`, etc.
  - Only `READ` and `WRITE`/`COMMIT` at data servers
  - `FH` and `stateid` returned in layout
- `GETDEVICEINFO` translates `dev_id` to address
  - For prototype simplicity, `dev_id` stores IP address
- `GETDEVICELIST` returns list of default `dev_ids`
  - Based on current default layout

Prototype: management protocol

- Responsible for managing state
  - Setting attrs, creating state, removing files, etc.
- Current metadata ops that invoke mgmt
  - `OPEN` w/create
  - `REMOVE`
  - `SETATTR` of size
  - `LAYOUTCOMMIT`
- Future:
  - ACLs, mandatory locks, delegated layouts, etc.

Prototype: management protocol

- `OPEN` w/create creates state on data servers
  - Performs create on metadata server
    - Inherits default layout (stripe size + data servers)
  - Creates data files on remote data servers
    - Data files stored in root dir, named by inode #
    - Data files created w/same mode, uid/gid, size
    - Returns filehandles (stateid in future)
  - Creates layout as named attr on metadata server
    - Stores layout with returned `FH`s

Prototype: management protocol

- `REMOVE` removes state on data servers
  - Opens layout stored on metadata server
  - Issues `REMOVE`s for each data file
  - Metadata server removes layout and local file
  - Future: async `REMOVE`s to data servers?
- `SETATTR` may need to truncate/grow data files
  - `SETATTR`s that affect size are sent to data servers

Prototype: management protocol

- `LAYOUTCOMMIT` ensures size is consistent
  - Currently EOF flag and length provided as args
  - Prototype uses this as a hint
  - If EOF is set and len is > metadata file length
    - Metadata server issues SETATTRs of new length
    - This may sparsely extend data file(s)
  - Can use to update mtime
  - Future: handle error cases
    - E.g., server failures before `LAYOUTCOMMIT`

Standardization

- Just moved into NFSv4 working group
  - Still at beginning of process (individual drafts)
  - At least 1½ years to standardization
- Focus on NFSv4 files-based protocol
  - Easiest to get through WG (wrt. security, etc…)
  - Design must support multiple storage protocols
- Requires
  - Thorough understanding of error conditions
  - Fully specified files-based access protocol
  - Working strawman implementation

For more information

- NFSv4 WG mailing list
  - http://www1.ietf.org/mail-archive/web/nfsv4/current/index.html
- Current proposals & drafts (from CMU's pNFS)
  - http://www.pdl.cmu.edu/pNFS/
- My contact info:
  - Garth Goodson
  - Network Appliance
  - goodson@netapp.com