NetworkAppliance®

The evolution of storage.™

NFSv4 Extensions to Support Parallelism

Peter Corbett

December 4, 2003

Introduction

- **Proposition:**
  - **NFSv4 can be simply extended to support parallelism, data distribution, and high performance I/O**
- **Goal is to meet needs of**
  - **HPC community**
  - **Linux clusters**
  - **Database and enterprise computing**
- **Leverage existing V4 implementations with a very small set of protocol extensions**
- **Separation of metadata, data function at server not required**

Overview

- Propose to extend NFSv4 in two ways
- Extensions are suitable for inclusion in NFSv4.1
  - Small set of extensions
  - Optional for clients and servers
  - Strict superset of existing protocol
- Affects client/server protocol only
  - No specification of parallel server implementation or server-to-server protocol
- Compatible with NFS RDMA

Two Sets of Protocol Changes

- Support for unordered operations within compounds
- Support for data distribution within files and directories
- These two sets of changes are independent of each other
  - Can be considered separately for inclusion in protocol

Unordered Operation

Single NFSv4.0 compound can specify multiple I/O operations to server

- **Proposal: Allow unordered set of operations in a compound**
  - **Ops can be performed concurrently or in arbitrary order at the server**
- **Provides support for I/O list and batch I/O operations**
- **Useful for strided access and for cache prefetchers and cleaners**

Unordered Operation Set

- Add two new operators to v4
  - UNORDEREDBEGIN
  - UNORDEREDEND
- Specify beginning and end of a set of unordered operations within a single compound
- Both have void parameter list
- MUST not be nested (NFS4ERR_UNORDERED_INVAL)

Some Semantics

- PUTFH applies to succeeding operations within unordered set
- Server is free to ignore unordered directive and perform ops in order
- Server can reject an ambiguous unordered sequence (NFS4ERR_UNORDERED_INVAL)
- Overlapping read/write and write/write conflicts are allowed
  - Server not required to check
  - Can execute I/O ops in any order

Unordered Error Handling

- Status of UNORDEREDBEGIN is always NFS4OK or NFS4ERR_UNORDERED_INVAL
- Status of UNORDEREDEND is always NFS4OK, NFS4ERR_UNORDERED_INVAL or NFS4ERR_UNORDERED_FAILED
- Return FAILED if any op in unordered sequence fails
- More than one op can fail

More Unordered Error Handling

- Status returned for all ops up to UNORDEREDEND, or for first failing op
- If there is a failed op, status of UNORDEREDEND and of COMPOUND is NFS4ERR_UNORDERED_FAILED

Data Distribution

- A single file system (FSID) can span multiple servers
  - Or SSI server with multiple access points
- Named objects (directories and files) are always local to the server that contains directories with hard links to them
- New type of named object: “Metafile”
  - Basically a file that has its data elsewhere
- Second new type of unnamed object: “Data Fork”
  - Data portion associated with a metafile

Metafiles and Data Forks

- Data Forks can be located on different servers from their metafiles
- Allows data distribution
  - I/O can be separated from metadata operations
  - Provides “directory scaling”
    - Increases aggregate I/O bandwidth to the files in a directory
    - Does not improve throughput of namespace operations within a directory

Linkage

- Each data fork has only one metafile
- Metafiles can be multiply linked into namespace, just like ordinary files
- Add a new optional attribute to metafile
  - data_locations
  - Similar to fs_locations
  - Contents are a list of server name strings and file handles
  - File handles are handles of data forks

Data Forks

- Only way to get data fork FH is through GETATTR on metafile
- Client performs GETATTR to retrieve data_locations attribute
- New variant OPEN DISTRIBUTED
  - similar to OPEN RECLAIM
  - takes a file handle as an argument

Transparency

- Data is accessible through metafile
- Server proxies data from data fork
- File appears normal to client
- Client has to explicitly look for non-empty data_locations attribute to take advantage of distribution

Parallel Files

- Can have more than one data fork per file
  - File Scaling
- V4 client can stripe data across the data forks
  - Data forks are sparse
  - Overlay of all data forks is complete file
- Client is free to expose the data forks to the application as a collection of parallel data containers
  - Suitable for parallel I/O libraries such as MPI-IO
  - No need to hide the inherent parallelism
  - Exposure is outside scope of protocol spec

data_locations attribute

- data_locations is a list of server name strings and filehandles, one per data fork
- Can have multiple data forks of same file on a server
  - Completely up to the server to distribute data forks
- Client can request that a file have multiple data forks by specifying a non-empty list of null entries for data_locations in CREATE, SETATTR
- Client can request change in number of data forks via SETATTR
- Server not required to comply
  - Can simply create a normal file, with empty data_locations attribute

Single Use Volatile File Handles

- Propose a new flavor of volatile fh
  - NF4_VOLATILE_SINGLE_USE
  - data fork FH returned from GETATTR usable by calling client to open data fork exactly once
  - FH expires after lease period if not used in OPEN DISTRIBUTED
- Allows server to know when there are no outstanding FHs for a data fork
  - Facilitates restriping, migration, etc.
- Proof at data fork that metafile access control was checked
  - Handle all access control and access denial at metafile

File Handle as Secure Capability

- struct FH {
  - expiry time (for use in open)
  - data fork id
  - OWF(expiry time, data fork id, client cred, server secret)

  }

  Can't be forged
  Limited lifetime
  Server can invalidate it

data_distribution attribute

- Add another attribute data_distribution
  - Variable length array of uint4s
- Required attribute if data_locations length > 1
- First uint4 is stripe factor
- Default is zero-based round robin striping
- Subsequent uint4s allow other distributions to be specified
  - Useful for restriping
- Distribution description can be standardized in V4 spec

Operation Semantics

- Operations that affect metafile and data forks go to metafile
  - CREATE, REMOVE, SETATTR, etc.
- Data forks share owner and acls with metafile
- Data forks can be separately secured from metafile
  - E.g. different encryption level on metadata and data
  - Operations on data fork can provoke a SECINFO

Locking

- Data forks can be directly locked
  - These locks are held locally only
  - Only apply to single data fork even though byte range includes sparse regions held in other data forks
- Metafile can be locked
  - These locks must be propagated to the affected data forks
  - Can conflict with local data fork locks

GETATTR

- GETATTR of data fork does not retrieve information for whole file
- GETATTR of meta file retrieves correct size, mtime, atime, ctime for entire file

Server-to-Server Protocol

- Server-to-server communication is implied by data distribution
- Beyond scope of v4 spec
  - Should not be considered for inclusion in v4 spec
- Servers may use v4 to implement some functionality
- Server may use proprietary internal methods and protocols
- Possible to define a companion spec to v4 that specifies inter-server operation or some aspects of it

Compatibility and Leverage

- Simple set of extensions to V4
- Maps quite closely to some parallel server architectures
  - Would be relatively easy to use V4 with extensions as the client/server wire protocol
- Leverages existing V4 implementations, as well as current and planned parallel server implementations

Conclusions

- Two sets of simple extensions to V4
- Unordered operations support HPC, database I/O
  - Facilitate higher performance and throughput
- Data distribution allows directory and file scaling
  - Highly parallel I/O
  - Transparent or explicit parallelism at application
- Leverages existing V4 and parallel server implementations