Providing Parallel I/O on Linux Clusters

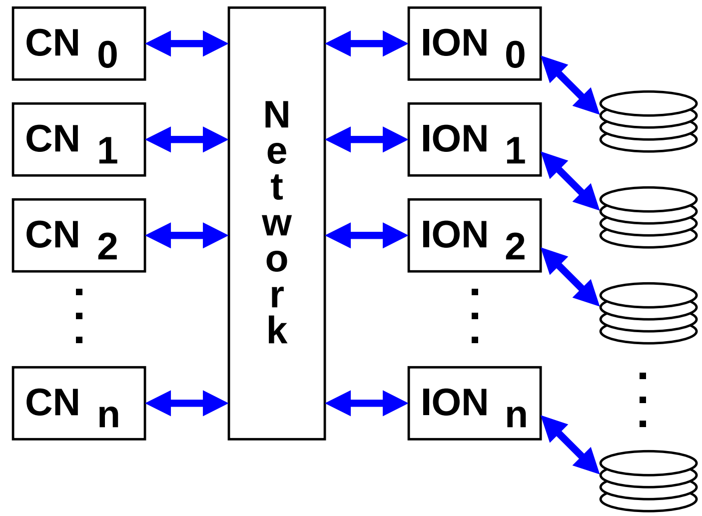
Rob Ross
Argonne National Laboratory
Mathematics and Computer Science Division
http://www.mcs.anl.gov/

Overview of Presentation

- A little background on parallel I/O
- Providing parallel I/O on Linux clusters
- Components:
  - ROMIO
  - Parallel Virtual File System (PVFS)
  - PVFS client-side VFS support
- Opportunities for improvement
- Final notes

Parallel I/O

- Use of multiple distributed I/O resources by a parallel application



- Goal is to increase aggregate I/O performance

- Accomplished by reducing bottlenecks in I/O path
  - no single I/O device
  - no single I/O bus
  - no single network path

- Target is medium to large clusters (64 or more nodes)

Providing Parallel I/O under Linux

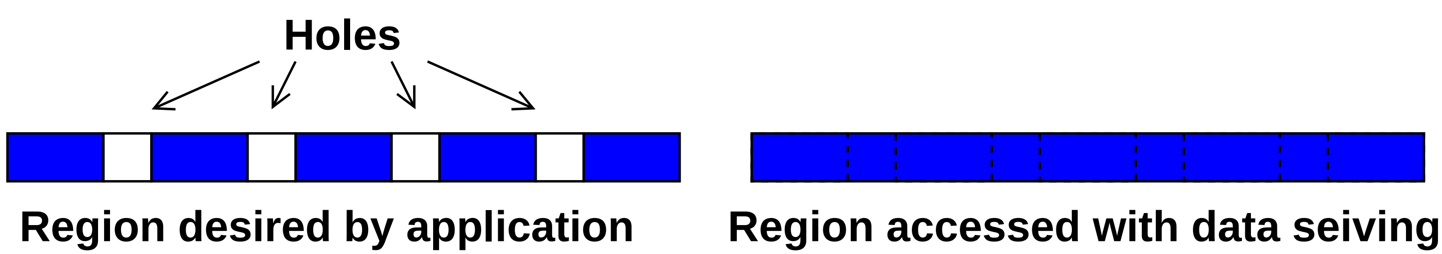
- Three software requirements:
  - Usable application interface
  - Underlying high performance data storage mechanism
  - Tools for every day file manipulation (e.g. `cp`, `rm`, `ls`)
- ROMIO provides the interface, MPI-IO
- Parallel Virtual File System (PVFS) provides data storage
- PVFS client-side VFS support allows existing tools to manipulate PVFS files

ROMIO MPI-IO Implementation

- Implementation of MPI-2 I/O standard
- Developed at Argonne National Lab
- Includes bindings for Fortran and C
- Allows for multiple underlying file systems via ADIO layer
- Supports PVFS, NFS, PIOFS, PFS, HFS, XFS, and others
- Provides optimizations for noncontiguous accesses and collective I/O

ROMIO Noncontiguous Accesses

- MPI-IO allows users to define "derived datatypes"
- These datatypes can have unaccessed regions, or "holes"
- To avoid multiple accesses for such a region, ROMIO uses *data sieving*
- Writes performed with read/modify/write



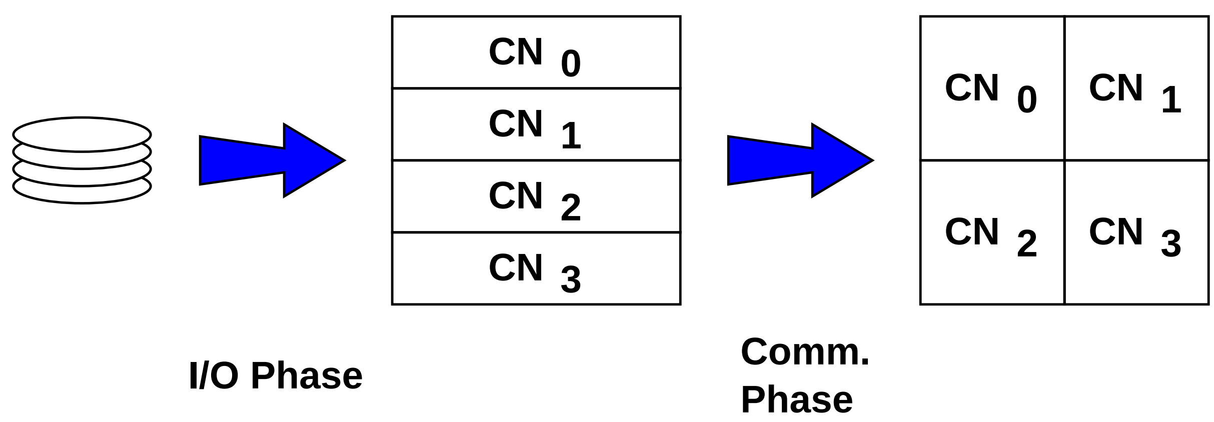
ROMIO Collective I/O

- ROMIO provides *two-phase* optimized collective I/O
- I/O performed in two steps:

  I/O Phase: Read data from disk in large contiguous chunks

  Comm. Phase: Shuffle among clients to obtain desired distribution

- Example: reading 2D array from disk (stored row-major) with block distribution



Parallel Virtual File System

- File System – allows users to store and retrieve data using common file access methods (open, close, read, write)

- Parallel – stores data on multiple independent machines with separate network connections

- Virtual – exists as a set of user-space daemons storing data on local file systems

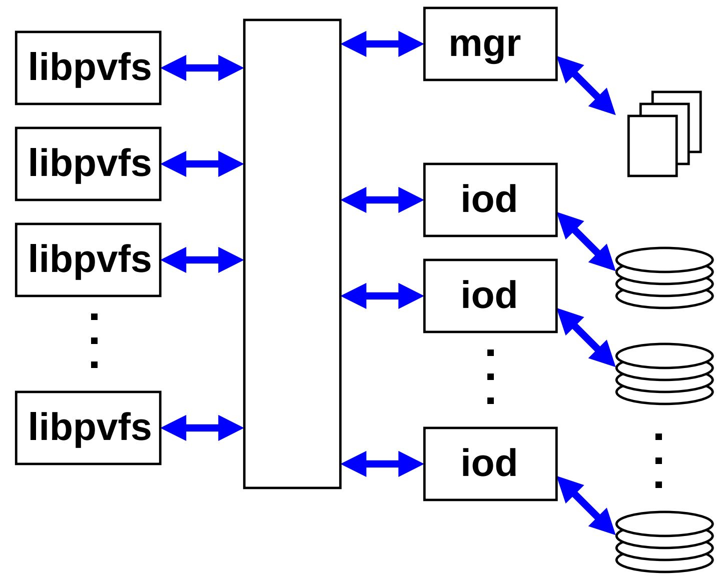
PVFS Components

Two server types:

- mgr – file manager, handles metadata for files
- iods – I/O servers, store and retrieve file data

Client-side library:

- libpvfs – links clients to PVFS servers



- libpvfs hides details of PVFS access from application tasks

- Multiple interfaces utilize libpvfs, including ROMIO

PVFS Server Design

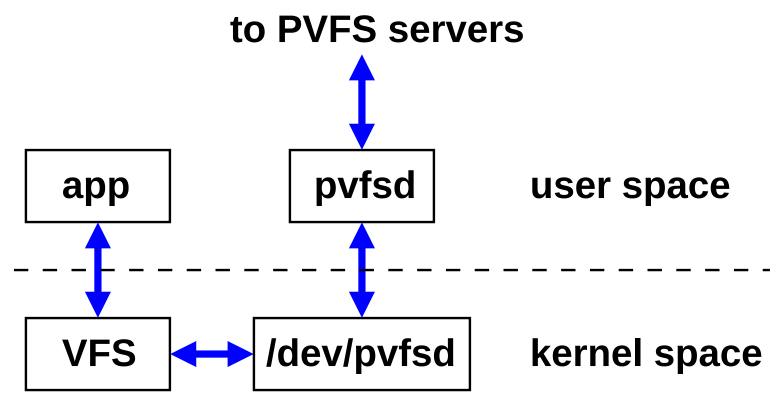
- Single-threaded, `select` driven
- Use non-blocking reads and writes for socket I/O
- Store file data on a local file systems
- Read-only `mmap` used for reading file data
- For writes, data is read from socket into buffer and then written

Linux VFS Support

- PVFS kernel module registers PVFS file system type
- PVFS file systems can then be mounted
- Coda implementation used as example:
  - PVFS code converts VFS operations to PVFS operations
  - Client-side daemon handles network I/O
  - Requests passed through device file

Accessing PVFS Files Through VFS

- I/O operations pass through VFS



- PVFS code in kernel passes operation through device
- pvfsd reads requests from /dev/pvfsd
- Requests converted to PVFS operations, sent to servers
- data passed back through device
- Optionally use `map_user_kiobuf` to map user's buffer into kernel space and avoid one copy

PVFS Current State

- Linux 2.2 kernel support
- TCP data transfer only
- $2N$ Gbyte file size limit (N = # of I/O servers)
- Use UNIX interface to store data on local file systems (e.g. ext2fs, reiserfs)

Opportunities for Development

- High performance networking technologies
- Multi-threading to better overlap disk and network I/O
- Improved ordering of request service
- More direct data access (i.e. avoiding buffer cache)

Improving PVFS Data Storage

- Almost anything would be better :)
- More direct access to disk
- Control over cache
- Suggestions of approaches would be appreciated

Chiba City – The Argonne Scalability Testbed

- 256 nodes total
- We had 60 nodes to play with

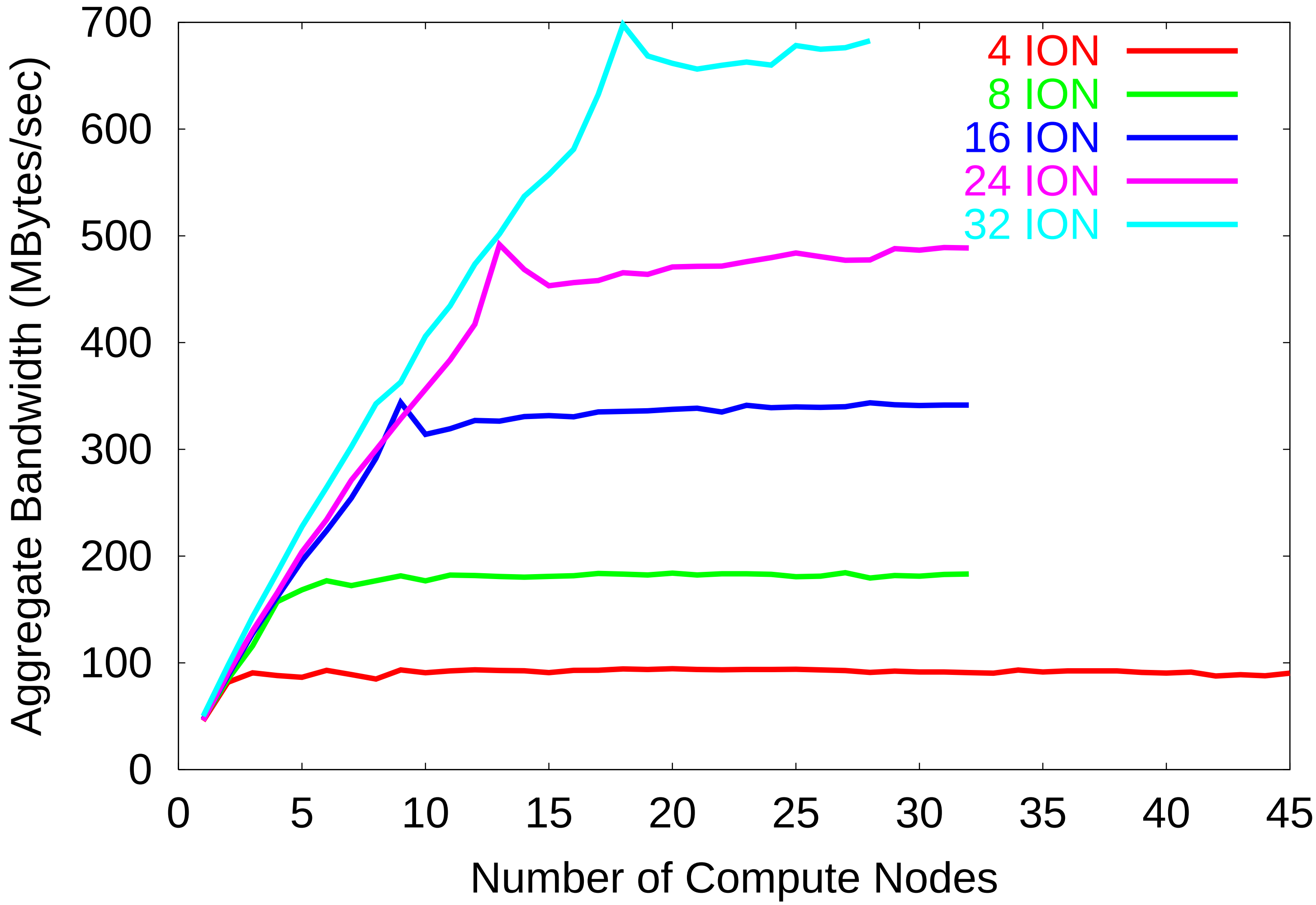
Hardware

- 500 MHz Pentium III
- 512 Mbytes RAM
- Myrinet (Rev. 3)
- 9 Gbyte SCSI disk
- NCR 53c875 based SCSI (40 Mbytes/sec)

Software

- Linux 2.2.15pre4
- PVFS 1.4.3
- MPICH-GM 1.1.2
- GM Driver 1.2pre2

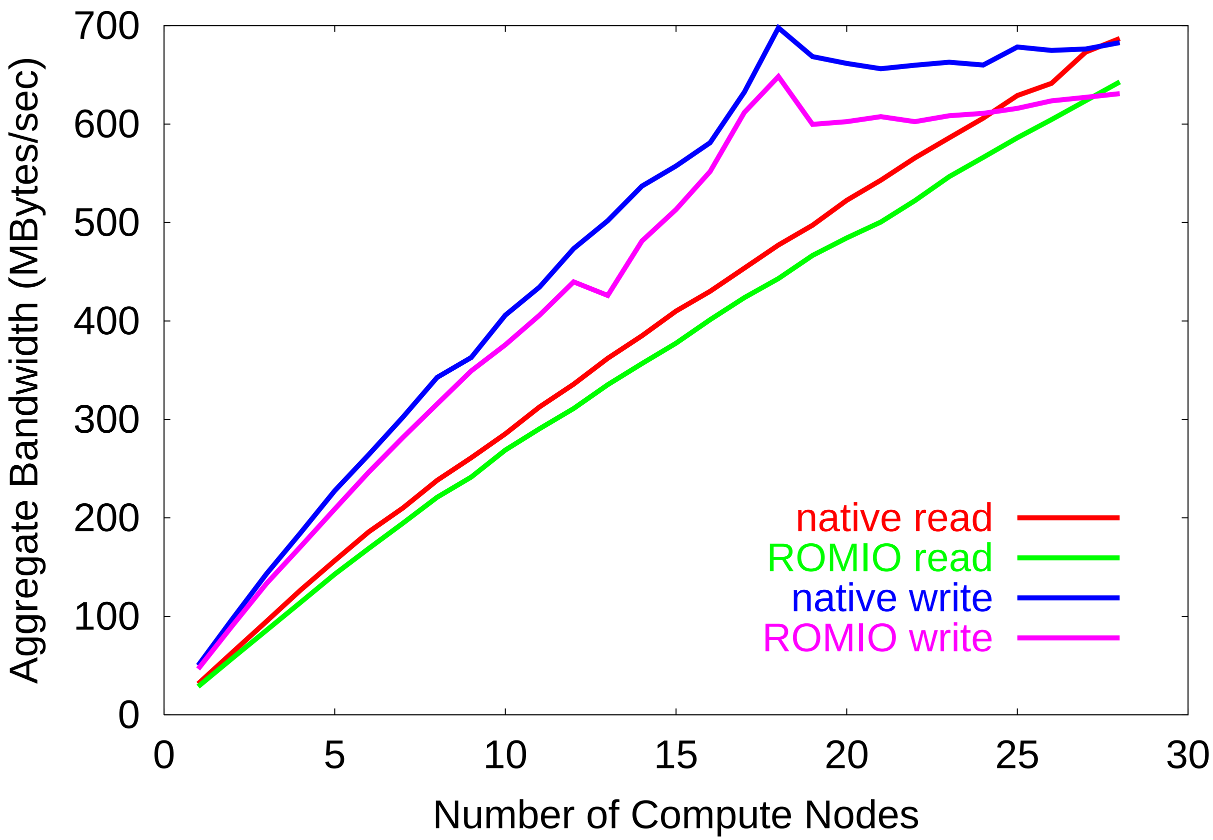
Native Write Performance



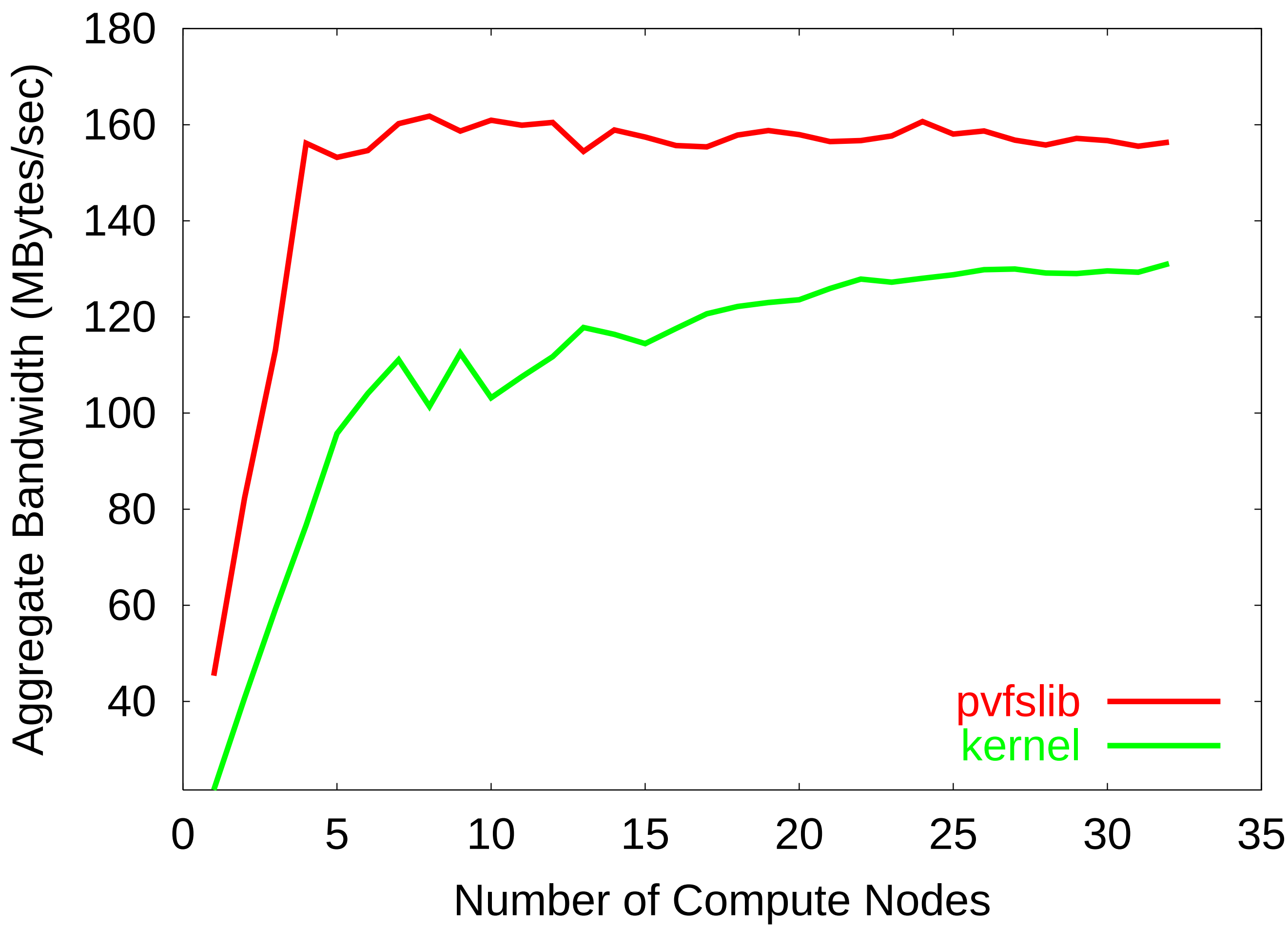
- 10/10/2000 - 112 CNs, 48 IONs, 21 Gbytes @ 1.05 Gbytes/sec

ROMIO Performance

- Using 32 I/O nodes, data sizes identical to concurrent tests
- At worst 8% overhead for using ROMIO



PVFS Write Performance through Kernel



- As much as 50% loss in bandwidth, stabilizes at 16% loss

Summary

- There is a parallel I/O solution for Linux clusters
- There are many potential development directions
- Some of these aren't likely to be pursued by commercial entities in the near term
- Obligatory web pages:
  - http://www.mcs.anl.gov/romio
  - http://www.parl.clemson.edu/pvfs