PVFS: A Parallel File System for Linux Clusters

Authors:
Philip H. Carns Walter B. Ligon III Robert B. Ross Rajeev Thakur

Presented by: Pooja Nilangekar
poojan@umd.edu

Motivation

- Cluster computing is a mainstream method for parallel computing.
- Linux is the most common OS for cluster computing.
- Lack of production-quality performant parallel file system for Linux clusters.
- Linux clusters cannot be used for large I/O-intensive parallel applications.

Solution: Parallel Virtual File System (PVFS) for linux clusters.

Used at: Argonne National Laboratory, NASA Goddard Space Flight Center, and Oak Ridge National Laboratory.

Objectives for PVFS

1. Basic software platform for pursuing further research in parallel I/O and parallel file systems in the context of Linux clusters.
    a. Requires a stable, full-featured parallel file system to begin with.
2. Meet the need for a parallel file system for Linux clusters.

- High bandwidth for concurrent read/write operations.
- Multiple API support: a native PVFS API, the UNIX/POSIX I/O API, other APIS MPI-IO.
- Common UNIX shell commands, such as ls, cp, and rm.
- Compatible with applications developed with the UNIX I/O API.
- Robust and scalable.
- Easy to install and use.
- Distribute the software as open source.

Related Work

- Commercial parallel file systems
  - PFS (Intel Paragon), PIOFS & GPFS (IBM SP), HFS (HP Exemplar), XFS (SGI Origin2000).
  - High performance and functionality desired for I/O-intensive applications.
  - Available only on the specific platforms.
- Distributed file systems
  - NFS, AFS/Coda, InterMezzo, xFS, and GFS.
  - Distributed access to files from multiple client machines.
  - Varying consistency semantics and caching behavior.
  - Not designed for high-bandwidth concurrent writes required for parallel scientific applications.
- Research projects
  - PIOUS: views I/O as transactions.
  - PPFS: adaptive caching and prefetching.
  - Galley: disk-access optimization and alternative file organizations.
  - Freely available but are mostly research prototypes.

PVFS Design

- High-speed access to file data for parallel applications.
- Clusterwide consistent name space.
- Enables user-controlled striping of data across disks on different I/O nodes.
- Client-Server architecture.
- Multiple servers - I/O daemons (nodes that have disks attached to them).
- Applications interact with PVFS via a client library.
- Manager daemon - handles only metadata operations such as permission checking for file creation, open, close, and remove operations.
- Higher performance if client, I/O daemons and managers are run on different nodes.
- Primarily a user-level implementation

PVFS Manager and Metadata

- Single manager daemon is responsible for the storage of and access to all the metadata.
- PVFS stores both file data and metadata in files on existing local file systems.
- Files are striped across I/O nodes to facilitate parallel access.
- Metadata parameters: base I/O node number, number of I/O nodes, and stripe size.
- Application processes communicate directly with the PVFS manager (via TCP) for metadata operations.
- Applications communicate directly with I/O nodes when file data is accessed.
- A mapping routine to determine whether a PVFS directory is being accessed; redirected to the PVFS manager.

I/O Daemons and Data Storage

- Nodes to be operated as I/O nodes are specified by the user at installation.
- Ordered set of PVFS I/O daemons runs on the I/O nodes.
- Use local disk on the I/O node for storing file data.
- Client nodes interact with I/O daemons directly.

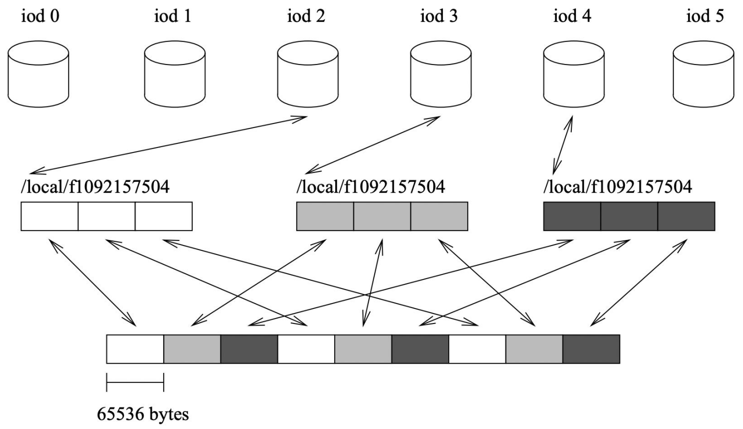
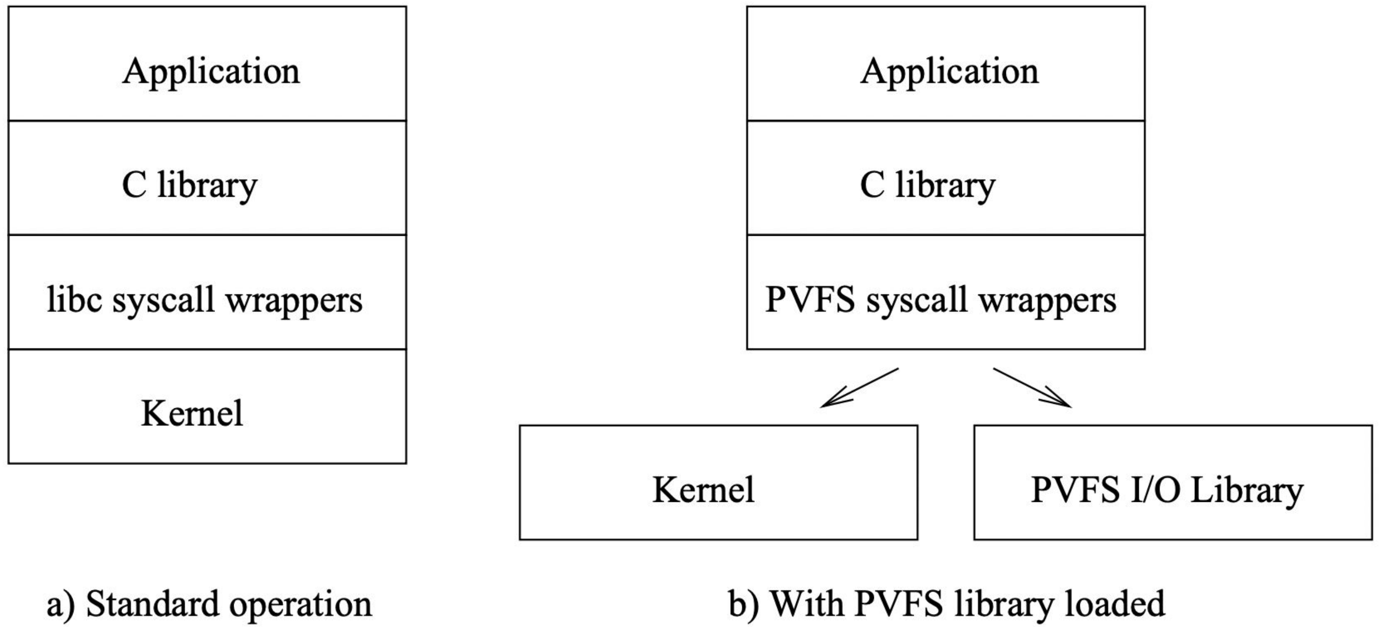


Figure 1: File-striping example

Application Programming Interfaces

- Multiple (APIs): a native API, the UNIX/POSIX API, and MPI-IO.
- Communication with I/O daemons and the manager is handled transparently.
- Partitioned API for noncontiguous file regions to be accessed with a single function call.
- Partition specified by: offset, gsize, and stride.
- MPI-IO interface on top of PVFS by using the ROMIO implementation.
- Noncontiguous MPI-IO using data sieving - makes large contiguous I/O requests and extracts the necessary data.
-

Trapping UNIX I/O Calls

- Using traps to identify PVFS I/O
- 

Linux Kernel VFS Module

Shortcomings of syscalls:

- A call to exec() will destroy the state that we save in user space, and the new process will therefore not be able to use file descriptors used for PVFS files
- Porting this feature to new architectures and operating systems is nontrivial.

The Linux kernel provides the necessary hooks for adding new file-system support via loadable modules without recompiling the kernel.

Performance Results

- Setup
  - Experiments conducted on Chiba City cluster at Argonne National Laboratory.
  - 256 nodes, each with two 500- MHz Pentium III processors, 512 Mbytes of RAM, a 9 Gbyte Quantum Atlas IV SCSI disk, a 100 Mbits/sec Intel EtherExpress Pro fast-ethernet network card operating in full-duplex mode, and a 64-bit Myrinet card.
  - 60 nodes were used for experiments.
- Experiments
  - concurrent reads and writes with native PVFS calls.
  - concurrent reads and writes with MPI-IO.
  - the BTIO benchmark.
- Varying number of I/O nodes, compute nodes, and I/O size. Using fast fast ethernet and Myrinet.

Concurrent Read/Write Performance

- MPI program that concurrently writes/reads data to disjoint regions of the file.
- Scalability limits of fast ethernet:
  - Peak read performance 222 Mbytes/sec for 24 I/O nodes.
  - Peak write performance 226 Mbytes/sec for 24 I/O nodes.
- Slower rate of increase in bandwidth indicates that the benchmark hit the maximum number of sockets across which it is efficient to service requests on the client side.
- Significant performance improvement with Myrinet instead of fast ethernet.
  - Read performance with 32 I/O nodes, peak bandwidth 687 Mbytes/sec.
  - Write performance with 32 I/O nodes, peak bandwidth 670 Mbytes/sec.
- Myrinet maintained performance consistency as the number of compute nodes was increased beyond the number of I/O nodes.

MPI-IO Performance

- Same MPI program as native PVFS benchmark.
- MPI-IO added a small overhead of at most 7–8% on top of native PVFS.

BTIO Benchmark

- I/O required by a time-stepping flow solver that periodically writes its solution matrix.
- “full MPI-IO” version of this benchmark uses MPI derived datatypes to describe noncontiguity in memory and file and uses a single collective I/O function to perform the entire I/O.
- Modified benchmark to perform reads and writes.
- 16 I/O nodes and varying compute nodes (16, 25, and 36).
- With more compute nodes, the smaller granularity of each I/O access resulted in lower performance.
- For Myrinet, the maximum performance at 36 compute nodes.

Conclusion and Future work

- PVFS: high-performance parallel file systems to Linux clusters.
- Compatible with applications using MPI-IO API.
- Future work
  - With fast gigabit networks, performance is limited by TCP.
  - Performance can be improved by tuning some parameters in PVFS and TCP, either a priori or dynamically at run time.
  - General file-partitioning interface to handle the noncontiguous accesses supported in MPI-IO.
  - Improve the client-server interface to better fit the expectations of kernel interfaces.
  - Flexible I/O-description format that is more flexible than the existing partitioning scheme.
  - Adding redundancy support.
  - Scheduling algorithms for use in the I/O daemons.

http://www.parl.clemson.edu/pvfs/