Development of PVFS V2 Parallel File System

W. B. Ligon

June 2, 2003

Abstract

Parallel I/O remains a critical problem for cluster computing. A significant number of important applications need high performance parallel I/O and most cluster systems provide enough hardware to deliver the required performance. System software for achieving the desired goals remains in the research and development stage. A number of parallel file systems have achieved remarkable goals in one or more of several key areas related to parallel I/O, but there is still great reluctance to commit to any file system currently available. This is mostly due to the fact that these file systems do not address enough issues at once in a package that is robust enough for widespread use. In this paper we will present the design of the PVFS V2 file system. PVFS V2 is the second generation of a parallel file system for Beowulf clusters developed at Clemson University and Argonne National Labs with support from Goddard Space Flight Center.

Introduction

The Parallel Virtual File System (PVFS) is a parallel file system designed for Beowulf class parallel computers [1][2]. PVFS provides distribution of file data across nodes in a cluster architecture and high performance access to data by tasks in parallel applications. PVFS is open source software designed for high-throughput access to large data sets by parallel applications running on Beowulf computers and the Linux operating system.

PVFS consists of two server daemons: a data server or I/O daemon (IOD) and a metadata server or manager daemon. Under PVFS, there is one manager daemon an on or more IODs. Clients access PVFS via the pvfslib, a library of Posix-like I/O function such as `pvfs_open()`, `pvfs_read()`, and `pvfs_write()`. Clients can also access PVFS via the kernel's VFS layer and the pvfs-kernel kernel module and pvfsd a client-side daemon which uses the pvfslib to access PVFS for the kernel.

PVFS uses tcp/ip to transport data between clients and servers. Data in PVFS is striped across IODs running on distinct nodes. Each file can select its own striping parameters including the number of nodes the file should be distributed to and the size of the striping unit. Metadata for all files is stored on one node where the manager daemon is running. I/O requests in PVFS consist of strided access patterns, where multiple blocks of contiguous bytes that are regularly spaced in the file can be accessed with a single read or write request.

PVFS was designed to be an experimental file system, used to research critical characteristics if applications with large I/O requirements and to prototype different file system features that might address those characteristics. Regardless, PVFS has become very popular as a parallel file system not so much as a defacto standard, but as a viable default. This is to say that even though PVFS has many faults and there have been many potential alternatives, PVFS remains places the solution for parallel file I/O on Beowulf clusters. It is important to note that PVFS is designed to provide high performance I/O for large parallel applications, and not necessarily security, high availability, or even high performance for small, random file access. PVFS itself does not utilize any kind of redundancy in storing files, though the underlying file system on the nodes can provide this. The PVFS manager daemon becomes a bottleneck when a large number of small files are accessed, or in the face of a large number of metadata operations.

Still, all things considered, PVFS has done well to live up to its purpose and has pointed the way to new innovations in parallel file system design. In order to begin realizing those innovations, it has become apparent that a new file system is needed, thus beginning in 2000, the PVFS development team began work on a then next version of PVFS: PVFSv2.

PVFSv2 Goals and Features

PVFSv2 is an effort to address many of the deficiencies in parallel file systems of today, including the dependence on particular network and storage hardware or interfaces, scalability limitations imposed on traditional consistency management approaches (e.g. locks), and the lack of application-specific metadata storage options at the file system layer. But first, the core infrastructure of the file system has undergone a complete overhaul intended to provide ready access to those internal components implicated in these issues and to make the software more manageable in a production environment. PVFSv2 represents a complete re-write of the original PVFS. While The original implementation provided simple striping, simple strided access requests, and tcp/ip as a transport mechanism, PVFSv2 is considerably more flexible. New design features include:

- Modular networking and storage subsystems,
- A powerful request format for non-contiguous requests based on MPI Datatypes,
- Flexible and extensible data distribution modules,
- Support for data redundancy, and
- Redesigned client and server codes.

PVFSv2 is the culmination of a 3 year effort to redesign PVFS as a production capable parallel file system based on experience gained in the design and operation of the original PVFS. The goals of the PVFSv2 project are:

- Provide a production ready parallel file system infrastructure,
- Use sound design practices, extensive documentation, and liberal code reviews in the development process,
- Portability, reliability, and performance,
- Expandable and customizable for research purposes.

In this paper we present an overview of the design of PVFSv2. As of this writing pre-release versions of the file system are running and various libraries and components are nearing completion. Major components are briefly highlighted.

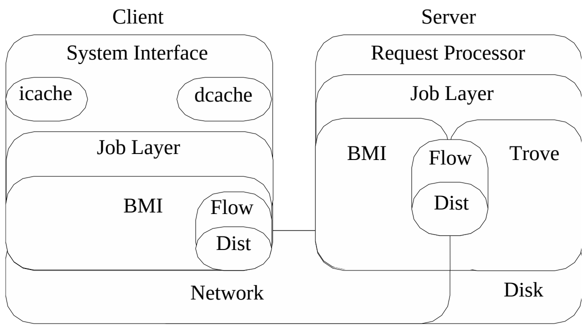

Figure 1: Software Architecture of PVFSv2.

PVFSv2 System Architecture

The PVFSv2 server is a combined metadata/data server. It is carefully constructed of several abstraction layers that facilitate the transfer of data in and out of storage and to and from the network interface. The PVFSv2 client is constructed from the same code up to the point that specific transfer requests are prepared. The purpose of the client is to translate kernel VFS-like calls into requests to a collection of servers that manage the file system. The shared code between the client and server goes a long way toward making the PVFSv2 code more manageable and compact.

A critical area of design in PVFSv2 is in abstraction layers for networking and storage. Three components of PVFS are BMI, Trove, and Flows. BMI and Trove abstract networks and storage, respectively, while Flows combine the two to abstract transfers between network and storage.

The Buffered Messaging Interface (BMI) provides a non-blocking network interface that can be used with a variety of high performance network fabrics and is tailored for use in file system servers and clients. Currently there are BMI modules for both TCP/IP and GM networks. Trove provides a non-blocking storage interface that can be used with a number of underlying storage mechanisms. Trove storage objects are called data spaces an each data space can hold both byte-stream data and key/value pairs. Byte-streams typically hold file data and my be implemented with host files or raw disk partitions or similar mechanisms. Key/value pairs are typically used to implement metadata and can be implemented using different types of database management tools. The current implementation uses Unix files and Berkeley db4. Thus, one of Trove's benefits is that metadata can be easily expanded, accessed, and searched.

A key feature of BMI and Trove are that they provide portability. In contrast, the original PVFS was limited to using tcp/ip and Unix file system for networking and storage. In the time since its development, a number of good alternatives have appeared, particularly in networking. PVFSv2 is more flexible in this regard. Use of BMI and Trove also represent information hiding, improve code quality, and increase reuse.

Flows combine the functionality of the network and storage subsystems by providing a mechanism to specify a flow of data between network objects and storage objects. Flows are an optimization structure that allows complex data transfers with a low level of control, without the overhead of those parts of the server that deal with requests and scheduling. Flows also incorporate the request and distribution processing system that allows PVFSv2 to handle highly complex access patterns.

The flow system provides a mechanism for specifying methods for transferring data between specific storage and network endpoints. Using this, a generic set of methods can be used to process BMI to Trove flows, but specialized, optimized methods can also be used for special purposes, like small I/O requests, thus providing better performance than the original PVFS and providing a means for research and expandability.

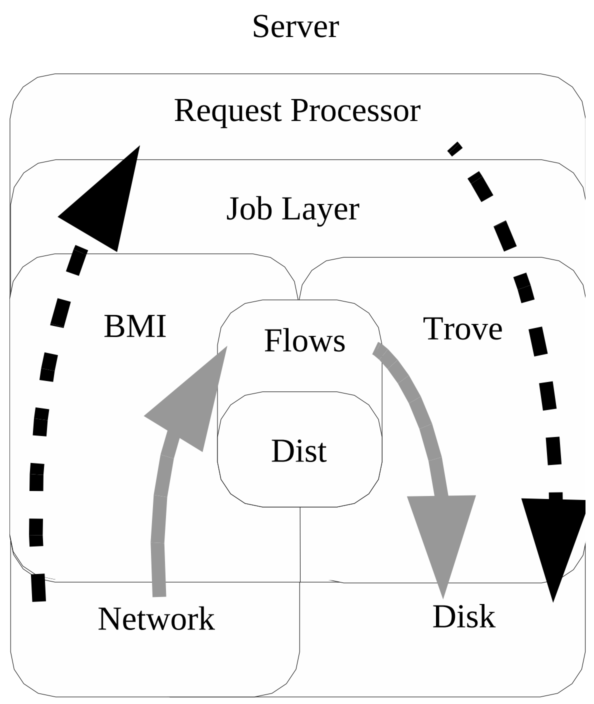

Figure 2: Flows provide a shorter path between storage and network.

One of the more important lessons learned from the design of the original PVFS was that the ordering and scheduling of transfers to and from the network and to and from storage can play a key role in file system performance. With the network and storage abstractions in place, and Flows as a means of transfer, the job layer provides this scheduling and task management.

The Job Layer provides a common interface to BMI, Trove, and Flow tasks. The Job Layer manages all of the tasks or jobs that are presently active and also manages asynchrony through the use of threads. Finally, the Job Layer performs dependency analysis and scheduling by deciding when each job is serviced. The Job Layer simplifies the use of the lower level facilities and controls task ordering for both performance an correctness.

The client and server code for PVFSv2 is the same from the Job layer down (except that the client doesn't need Trove for its function). This gives a reusable common interface to both codes, which greatly simplifies the structuring and maintenance of the high layers.

The System Interface is the interface between all higher level interfaces and PVFSv2. The system interface is designed to have other interfaces built on top of it. Each API call has a wealth of arguments that allow all of the features of PVFSv2 to be accessed where as most user interfaces would not expose so many complex details. The System interface is designed to be similar to the VFS interface of Linux and other similar operating systems so that PVFSv2 can readily be integrated as an OS supported file system. The other interfaces planned include an MPI-IO [3] interface based on ROMIO [4] for use with MPI and a Posix-like interface.

The PVFSv2 Server is controlled by a request processor that responds to client requests and directs the storage and network subsystems to perform the necessary transfers. The request processor is built from a system state machine written and compiled using a simple language developed specifically for PVFSv2. This system provides a highly structured means of handling requests, a simple means to reuse code within the request processor, and a powerful mechanism for adding new requests to the server for extensions or experimental purposes.

The server state machine greatly simplifies the programming of often repetitive tasks in the server and simplifies the management of concurrent requests. This is also a design feature the provides for expandability, as the request processing system can easily accommodate new specialized requests. This system is also being studied for use in the kernel daemon that allows users to access PVFSv2 through the kernel VFS.

PVFSv2 allows complex non-contiguous I/O requests via a format based directly on MPI Datatypes. A set of datatype constructor functions identical in function to the equivalent MPI calls is provided, and the format can readily be translated from existing MPI datatype formats. This format allows a compact representation of arbitrarily complex cyclic non-contiguous requests as well as representation of non-cyclic and hybrid patterns. The commonality with MPI Datatypes ensures compatibility with existing MPI-IO interfaces. PVFSv2 servers directly process this format to service I/O requests.

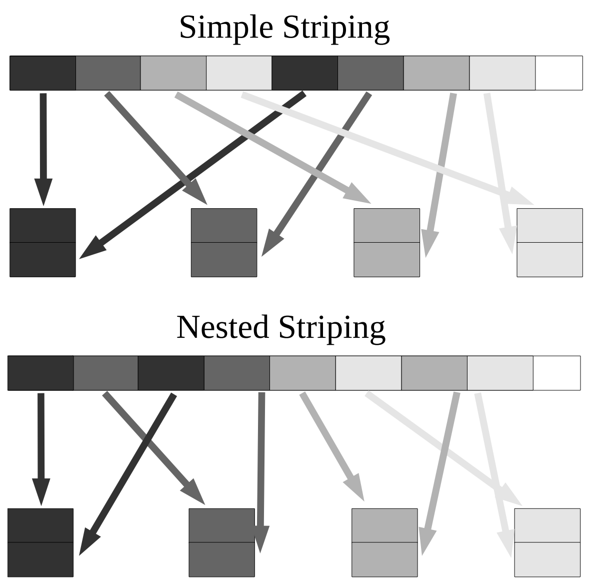

Figure 3: Example of two distributions.

The main feature of a parallel file system is that it distributes file data across multiple nodes in a parallel system. Most parallel file systems use a distribution known as striping, whereby equal size chunks of data are stored on the disks in a round-robin fashion. In PVFSv2 the distribution mechanism has been abstracted so that different files can be stored with different distributions.

Virtually any distribution that can be described with the set of methods specified by PVFSv2 can be used. For example, mechanisms that change the order that chunks are assigned to servers can be used to help match data storage to the access patterns of specific applications. One such distribution pattern is nested striping.

Conclusion

The PVFS project is an on-going project. In the coming months we expect PVFSv2 to take over as the flagship code of the project. This will represent the next generation of infrastructure for the file system, upon which we anticipate active research in cross node redundancy, interfaces, semantics, synchronization, and other important issues. PVFSv2 has been designed to not only fuel this research, but to act as a production ready platform for parallel I/O on Beowulf architectures.

References

References

[1] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, October 2000. USENIX Association.

[2] R. Ross, D. Nurmi, A. Cheng, and M. Zingale. A case study in application I/O on Linux clusters. In *Proceedings of SC2001*, November 2001.

[3] William Gropp, Ewing Lusk, and Rajeev Thakur. *Using MPI-2: Advanced Features of the Message-Passing Interface.* MIT Press, Cambridge, MA, 1999.

[4] Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems*, pages 23–32. ACM Press, May 1999.