An Overview of the Parallel Virtual File System

Walter B. Ligon III
Parallel Architecture Research Lab
Clemson University
102 Riggs Hall, Box 340915
Clemson, SC 29634-0915
walt@ces.clemson.edu

Robert B. Ross
Parallel Architecture Research Lab
Clemson University
102 Riggs Hall, Box 340915
Clemson, SC 29634-0915
rbross@parl.clemson.edu

Abstract

As the PC cluster has grown in popularity as a parallel computing platform, the demand for system software for this platform has grown as well. One common piece of system software available for many commercial parallel machines is the parallel file system. Parallel file systems offer higher I/O performance than single disk or RAID systems, provide users with a convenient and consistent name space across the parallel machine, support physical distribution of data across multiple disks and network entities (I/O nodes), and typically include additional I/O interfaces to support larger files and control of file parameters.

The Parallel Virtual File System (PVFS) Project is an effort to provide a parallel file system for PC clusters. As a parallel file system, PVFS provides a global name space, striping of data across multiple I/O nodes, and multiple user interfaces. The system is implemented at the user level, so no kernel modifications are necessary to install or run the system. All communication is performed using TCP/IP, so no additional message passing libraries are needed, and support is included for using existing binaries on PVFS files. This paper describes the key aspects of the PVFS system and presents recent performance results on a 64 node Beowulf workstation. Conclusions are drawn and areas of future work are discussed.

1 Introduction

One common piece of system software available for many commercial parallel machines is the parallel file system. Parallel file systems typically provide users with three services:

- a consistent name space across the machine,
- physical distribution of data across disks and network entities, and
- additional I/O interfaces.

The consistent name space aids programmers in accessing file data on multiple nodes. The physical distribution of data eliminates bottlenecks both at the disk interface and the network, providing more effective bandwidth to the I/O resources. Additional I/O interfaces allow the user to control how data is distributed, enable new access modes, and in some cases allow for larger files to be stored than possible on many, more traditional, file systems. PFS for the Intel Paragon, PIOFS for the IBM SP, HFS for the HP Exemplar, and XFS for the SGI Origin2000 are all examples of commercial support for parallel I/O.

As the use of PC clusters has grown, it has become obvious that system software support is necessary for parallel computing to continue to grow on this popular platform. In particular, the file systems that have been commonly used on this type of machine (AFS, NFS) do not provide the services needed by parallel applications, especially as machine sizes grow beyond only a few nodes. Thus new solutions are necessary to fill the need for true parallel file systems in PC clusters.

The Parallel Virtual File System (PVFS) project began in the early 1990's as an experiment in user-level parallel file systems for clusters of workstations but has since grown into a very usable, freely available, high performance parallel file system for PC clusters. This work will provide an overview of the design and features of PVFS, present some of our latest measurements of performance on a Beowulf workstation [3], and point to future work in the development of PVFS.

2 PVFS Design

As a parallel file system, the primary goal of the PVFS system is to provide high-speed access to file data for parallel applications. PVFS is a user-level implementation with the following features:

- provides cluster-wide consistent name space,
- enables user-controlled striping of data across nodes, and
- allows existing binaries to operate on PVFS files.

Because PVFS is a user-level implementation, no kernel modifications are necessary to install or operate the file system. The system uses TCP/IP to pass file data, so there are no dependencies on message passing libraries.

The PVFS system consists of three components: the manager daemon, which runs on a single node, the I/O daemons, one of which runs on each I/O nodes, and the application library, through which applications communicate with the PVFS daemons. The manager daemon handles permission checking for file creation, open, close, and remove operations. The I/O daemons handle all file I/O without the intervention of the manager.

In the following sections we will describe how PVFS stores file data, how it stores file metadata, one method by which applications can interface to the system, how data is transferred between applications and I/O nodes, and how existing binaries can operate on PVFS files.

2.1 Storing File Data

File data is stored on local file systems on I/O nodes. For each file striped across N I/O nodes, there will be N files, one per I/O node, holding that file's data. A unique identifier, supplied by another part of the file system, ensures that the names of these files will not conflict on the I/O nodes.

As a result of this, the UNIX mmap(), read(), and write() calls can be used directly by the I/O daemons to perform file I/O. The operating system will also cache file data for PVFS, so this is not performed in the PVFS code proper. A disadvantage of this approach is that we give up control of block allocation and cannot directly control what data is cached.

2.2 Storing Metadata

In the context of a parallel file system, metadata is information describing the characteristics of a file. This includes permissions, the owner, and most importantly a description of the physical distribution of the file data.

With PVFS, an NFS-mounted file system is used to store the metadata. We found that this scheme was more convenient than our previous attempts in that it gives us our unique name space, provides a directory structure for applications to see, and lightens the load on the manager.

2.3 Application Interfaces

The first interface to PVFS was heavily influenced by two research projects in progress at the time, the Charisma Project [2] and the Vesta Project [1]. The Charisma Project focused on characterizing the I/O of workloads on parallel machines. One of their observations was that a large fraction of I/O accesses occurred in what they called "simple strided" patterns. These patterns are characterized by fixed-size accesses which are spread apart by a fixed distance in the file.

The Vesta parallel file system was originally designed for the Vulcan parallel computer. Its interface was interesting in that it allowed processes to "partition" a file in such a way that the process would only see a subset of the file data.

We combined these two concepts into a "Partitioned File Interface", which forms the basic interface to PVFS. With this interface applications can specify partitions on files which allow them to access simple strided regions of the file with single read() and write() calls, reducing the number of I/O calls for many common applications.

2.4 Transferring File Data

PVFS relies on TCP to transfer data. During the early phases of the design, TCP performance tests indicated that we could utilize most available bandwidth using TCP. This has continued to be the case, although there are some situations where characteristics of TCP such as slow start and delayed acknowledgement result in poor performance.

Application tasks communicate directly with I/O nodes when file data is transferred, and connections are kept open between applications and I/O nodes throughout the lifetime of the application in order to avoid the time penalty of opening TCP connections multiple times. A predetermined ordering is imposed on all data transfers to minimize control messages, and simple-strided requests are supported by the I/O nodes directly to allow for larger request sizes.

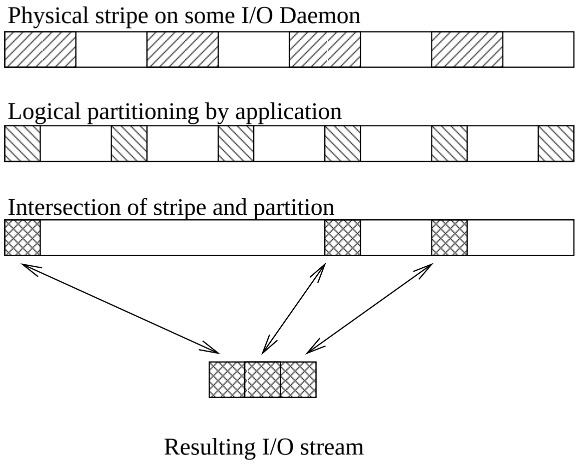

Figure 1: Example of an I/O Stream

Figure 1 shows an example of the I/O stream between an application and an I/O node resulting from a strided request. Each side calculates the intersection of physical stripe and the strided request. The data is always passed in ascending byte order and is packed into TCP packets by the underlying networking software.

2.5 Using Existing Binaries

We take advantage of the LD_PRELOAD variable to allow existing binaries to operate on PVFS files. A collection of system call wrappers are preloaded allowing us to catch I/O calls before they pass into the operating system. The state of open files is kept in user space, and accesses using file descriptors referring to PVFS files are handled by the PVFS library. All other calls are passed through to the appropriate system call.

3 PVFS Performance

In this section we examine the performance of PVFS on a Beowulf workstation while running a simple parallel test application. The purpose of these tests was twofold; we want to show the potential of PVFS, but we also want to point out areas where improvement is still warranted.

3.1 Test Environment

The Beowulf workstation used in these tests resides at the NASA Goddard Space Flight Center. It is a 64 node system, each housing:

- dual-processor Pentium Pro 200MHz, 128MB RAM
- 6 GB Seagate IDE drive
- 100Mbit Intel EtherExpress Pro in full duplex mode

A Foundry Network FastIron II switch connects the nodes. A separate front-end node is connected to the switch via a 1Gbit full duplex connection. The nodes were running Linux 2.2.5, MPICH 1.1.2, and PVFS 1.3.1.

Two different models of Seagate disks were used in the system, with advertised sustained transfer rates of 5.0 and 7.9MB/sec. Testing using the Bonnie disk benchmark showed 8.81MB/sec writing with 27.1% CPU utilization and 7.51MB/sec reading with 17.3% CPU utilization. Using ttcp version 1.1 TCP throughput was measured at 11.0MBps.

The 64 nodes in the system were divided into 16 I/O and 48 compute nodes for the purposes of these tests. The number of I/O and compute nodes used was varied throughout the tests. The test application, run under MPICH, performed the following operations:

- Create new PVFS file
- Simultaneously write data blocks to disjoint regions
- Close and reopen the file
- Simultaneously read same data blocks back from the file
- Close the file

The file was removed and the disks synchronized between each test iteration.

3.2 Scaling and I/O Nodes

The first set of test runs were designed to test the performance of PVFS as the number of I/O nodes was scaled. The amount of data written on each I/O node is held constant for each number of application tasks.

Here we see that we reach a maximum of around 30MB/sec for 4 I/O nodes, 60MB/sec for 8 I/O nodes, and 120MB/sec for 16 I/O nodes. These values closely match the maximum performance we would expect to get out of the disks on each node, although it is likely that at the points where these peaks are occurring we are mostly working from cache. In all cases we find that network performance is a bottleneck for small numbers of application tasks, but it

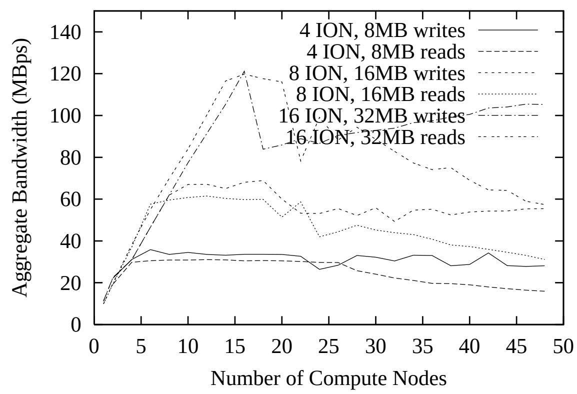

Figure 2: Effects of Increasing Number of I/O Nodes

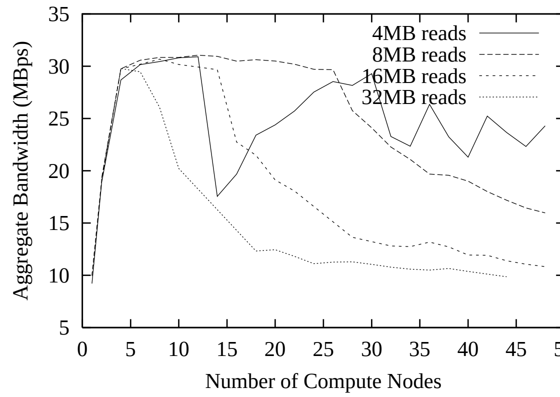

Figure 4: Read Performance for 4 I/O Nodes

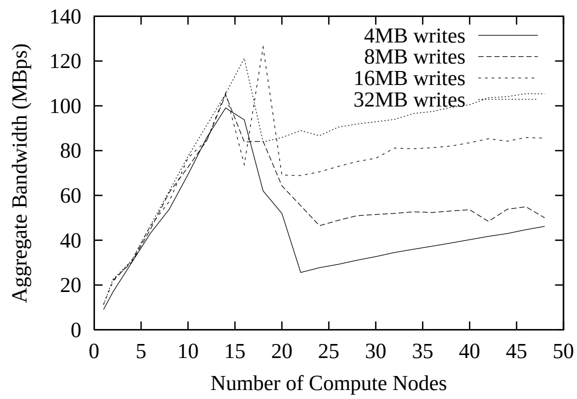

Figure 3: Write Performance for 16 I/O Nodes

appears that disk is the bottleneck for larger numbers of application tasks (and thus larger amounts of data).

3.3 PVFS Write Performance

Figure 3 focuses on PVFS write performance using 16 I/O nodes. We see a significant drop-off in performance for 16-20 application tasks, but performance climbs again after this point. The subsequent rise in performance indicates that we have not hit the limit in performance of disk or network, but rather that we are inappropriately using one or both of these resources.

3.4 PVFS Read Performance

In Figure 4 we examine read performance for PVFS using 4 I/O nodes. Here we can see two effects. First, for 4MB accesses performance is erratic. Second, for the larger accesses, when total access size exceeds 60MB we see a significant drop-off in performance. Here it is possible that we are limited by disk performance; however, this has not been established and further testing will be required to determine if this is in fact the case. It is also possible that our technique for file reads is not optimal and can be improved to lessen or eliminate this drop-off.

4 Conclusions and Future Work

PVFS is a work in progress. The focus of current development is on scalability, reliability, and support of additional interfaces. We hope to extend the capabilities of PVFS so that it can support systems of many hundreds of nodes, and we are actively working with the authors of the ROMIO MPI-IO implementation [4] in order to provide a high performance MPI-IO option for Beowulf workstations. We are also actively investigating the variances in performance that are obvious from the tests presented here in an effort to provide more predictable performance.

At the same time, PVFS is stable enough for regular use and provides compatibility with existing binaries, which makes parallel I/O a real option for Beowulf workstations and Piles-of-PCs. For more information on obtaining and installing PVFS, see the PVFS Project pages at http://ece.clemson.edu/parl/pvfs/.

References

[1] Peter F. Corbett, Dror G. Feitelson, Jean-Pierre Prost, and Sandra Johnson Baylor. Parallel ac-

cess to files in the Vesta file system. In *Proceedings of Supercomputing '93*, pages 472–481, Portland, OR, 1993. IEEE Computer Society Press.

[2] Nils Nieuwejaar, David Kotz, Apratim Purakayastha, Carla Schlatter Ellis, and Michael Best. File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1075–1089, October 1996.

[3] Daniel Ridge, Donald Becker, Phillip Merkey, and Thomas Sterling. Beowulf: Harnessing the power of parallelism in a pile-of-pcs. In *Proceedings of the 1997 IEEE Aerospace Conference*, 1997.

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