Design and implementation of PVFS-PM: a cluster file system on SCore

Koji SEGAWA Osamu TATEBE Yuetsu KODAMA Tomohiro KUDOH
Toshiyuki SHIMIZU
Grid Technology Research Center,
National Institute of Advanced Industrial Science and Technology (AIST)
k.segawa@aist.go.jp

Abstract

This paper discusses the design and implementation of a cluster file system, called PVFS-PM, on the SCore cluster system software. This is the first attempt to implement a cluster file system on the SCore system. It is based on the PVFS cluster file system but replaces TCP with the PMv2 communication library supported by SCore to provide a scalable, high-performance cluster file system. PVFS-PM improves the performance by factors of 1.07 and 1.93 for writing and reading, respectively, with 8 I/O nodes, compared with the original PVFS on TCP on a Gigabit Ethernet-connected SCore cluster.

1. Introduction

Recently, cluster systems are widely used as high performance computing environments. A cluster system can not only provide increased computing power by aggregating the performance of processors, but can also provide a large, high-performance file system by combining the local file systems of computing nodes. To provide such a combined file system, a parallel cluster file system software application required.

The Real World Computing Partnership (RWCP) developed cluster system software called SCore [3] to provide a parallel processing environment for cluster systems. As described in the next section, SCore provides a parallel processing environment, including various tools and commands. A high-performance, low-level communication library called PMv2 [7] is available on SCore. PMv2 uses a lighter communication protocol than TCP/IP. However, SCore does not yet have its own cluster file system that has performance scalable to the number of nodes.

PVFS [2] is an open-source cluster file system on Linux. PVFS is implemented on top of a TCP/IP communication stack. We ported PVFS to the SCore cluster system software. In this paper, we call the original PVFS, "PVFS-TCP," and the ported one, "PVFS-PM." PVFS-PM is the first cluster file system implemented on SCore. By using the PMv2 communication library instead of TCP/IP, PVFS-PM yields better performance than PVFS-TCP on the same hardware.

Since the PMv2 API for Gigabit Ethernet and Myrinet are the same, PVFS-PM can be used for both networks. PMv2 on Myrinet does not support TCP/IP communication. Therefore, when SCore is used on top of Myrinet, PVFS-PM is the only way to provide a cluster file system using Myrinet. When Gigabit Ethernet is used, communication using PMv2 and TCP/IP can use the same network. Therefore, not only users who use a SCore-based programming environments but also those who use the basic Linux environment with TCP/IP communication can enjoy the high performance of PVFS-PM without additional hardware.

2. PVFS and SCore

2.1. PVFS

PVFS is a freely distributed parallel file system for Linux clusters, and it has been developed mainly by Clemson University. It is a parallel file system using local disks distributed on each node of a PC cluster. PVFS is designed as a client/server system. This system consists of I/O daemons (iods) executed on the I/O nodes providing the local disks, a meta data manager (mgr) that is holding the meta data of the file system, and client libraries for accessing the parallel file system (Figure 1).

PVFS stripes files using uniform-sized stripes across the I/O nodes in the cluster. The local file system is used for storing the file stripes. Client libraries provide the PVFS I/O API. In the API, the client accesses the meta data manager to get file information, such as a list of I/O nodes and

Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)
0--7695-1919-9/03 $17.00 © 2003 IEEE

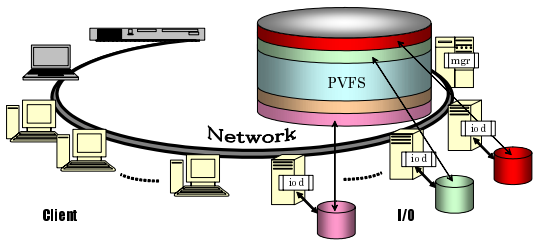

Figure 1. Configuration of PVFS

stripe size, when the file is opened. This information can be freely specified to set up file arrangement. When the file is accessed for read and write, the client accesses I/O nodes directly without accessing the meta data manager. Therefore PVFS can achieve efficient file I/O. PVFS also supports Linux VFS. By mounting PVFS using VFS, programs that use standard UNIX I/O API such as `open()`, `read()`, `write()`, `close()` can access files on PVFS without any changes.

NFS is a commonly used file system for clusters. Since NFS uses a single file server, its performance is limited and it becomes a bottleneck when many clients accesses the file server at the same time. On the other hand, since PVFS has multiple I/O nodes, it can achieve disk performance scalable to the number of I/O nodes involved, and can tolerate simultaneous access by many clients. PVFS is a user-level system and it can be used without any kernel modifications.

2.2. SCore

The SCore Cluster System Software is a high-performance parallel programming environment for workstations and PC clusters. It was developed by the Parallel Distributed System Software Laboratory of the RWCP, and is currently maintained and distributed by the PC Cluster Consortium [6].

SCore supports a single system image for a cluster. Using SCore, users are not aware of whether or not a system is a cluster of single/multi-processor computers or a cluster of clusters. A parallel application or an ordinary UNIX command may be run by just specifying a computer node group of such a cluster. A Unix command runs in the SIMD execution mode. To utilize processor resources and to enable an interactive programming environment, SCore multiplexes parallel processes in the processors' space and time domains simultaneously. Parallel processes are gang-scheduled when multiplexed in the time domain.

SCore uses a high-performance, low-level communication library called PMv2. PMv2 is described in detail in the next section.

3. Design and Implementation on PVFS-PM

PVFS-PM was designed based on the original framework of PVFS. The implementation is based on PMv2, version 2.1 and PVFS, version 1.5.5.

3.1. PMv2

PMv2 is a high-performance, low-level communication library dedicated to cluster computing using many types of networks. Currently, PMv2 drivers for Myrinet, Ethernet, UDP, and the Shmem shared memory interface have been implemented. To achieve low latency and high-throughput communication, PMv2 uses a lighter weight user-level communication protocol compared to TCP/IP, and eliminates kernel traps and data copies between kernel and user space. PMv2 provides not only point-to-point message passing APIs, but also remote memory operations.

PMv2 provides a virtual network mechanism called PMv2 channel in order to realize a multi-user environment on the SCore software. The channel provides reliable datagram communication, instead of connection-oriented communication such as that of TCP/IP. Each process of a parallel application uses the same PMv2 channel exclusively, which forms a virtual network. The PMv2 context stores the status of the PMv2 channel, and enables context switching on the SCore scheduler.

3.2. PVFS-PM

Since PMv2 does not support socket APIs, it is necessary to emulate the socket layer using PMv2 APIs. Basically, the PMv2 message-passing APIs are quite useful to emulate this layer, for instance, replacing `read()` and `write()` with `pmReceive()` and `pmSend()`, respectively, while taking the following considerations into account.

1. First, a program using PMv2 should initialize PMv2 devices. The program acquires a list of nodes that are used in the cluster system. It then opens a new context and associates this with nodes and channels. Lastly, it obtains its own node number in the context.

2. Replacing socket APIs with PMv2 messages-passing APIs is not enough to implement PVFS-PM. PVFS needs the source node id of the connection, which is not obtained by PMv2 APIs because it is not included in the PMv2 packet header. In PVFS-PM, the source node id , *Sender*, of each packet is added in the packet data format, which is depicted by Figure 2.

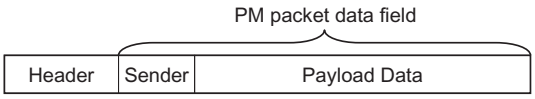

Figure 2. PVFS-PM packet format

3. In PVFS-TCP, the `select()` or `poll()` system call is used for processing remote I/O requests or acknowledgments basically in order of data arrival. For instance, an iod may receive requests from multiple clients, and a client may either wait for data or acknowledge requests from multiple iods. Since `pmReceive()` receives messages from multiple senders in order of arrival, `select()` and `poll()` can be replaced with `pmReceive()`.

4. Performance evaluation for large files

4.1. Evaluation environment

We compared the performance of PVFS-TCP and PVFS-PM by measuring the write and read bandwidth of large files. The performance of PVFS-TCP was evaluated using Linux 2.4.18 and Linux 2.4.19. Since the performance using 2.4.19 was much higher (especially for write) than that using 2.4.18, we only show the results using 2.4.19 in this paper. For PVFS-PM, Linux 2.4.18-2SCore, which is used in the SCore 5.2 package, is used.

Other elements of the evaluation environment are as follows (Figure 3).

- cluster node
  - Fujitsu PRIMERGY L200 $\times 17$
    $chipset : ServerWorksHE-SL$
    $CPU : PentiumIII 1.13GHz \times 1$
    $memory : 1GB$
    $HDD : FujitsuMAN3184MC \times 2, SeagateST373307L\ C \times 1 SCSI$
    - System files and the transferred data are stored on different disks.

network

- NIC: 3Com 3C996B Gigabit Server NIC
- Switch: 3Com SuperStack 3 4924 (non-blocking Gigabit Ethernet switch)

Gigabit Ethernet driver

- Broadcom Gigabit Ethernet Driver bcm5700 with Broadcom NIC Extension (NICE), ver. 2.0.18 (08/24/01)

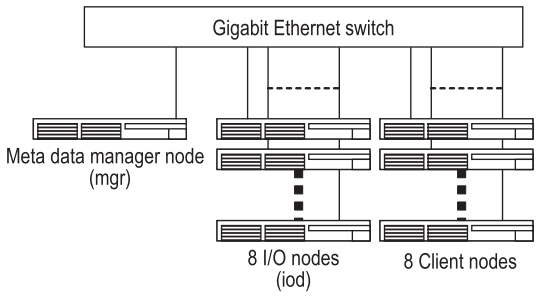

Figure 3. Evaluation environment

4.2. Benchmark program

To evaluate bandwidth, we wrote a disk I/O benchmarking program which uses PVFS client libraries. This program executes read/write operations of files on the PVFS file system from a client node using the native PVFS API. It accurately measures elapsed time using the processor's timer register to investigate the cause of a performance bottleneck. File size, Read/write block size, PVFS stripe size and the number of I/O nodes can be specified as parameters of this program.

A read or write operation is executed as follows. First, the client calls `pvfs_read()`/`pvfs_write()` with a specified *block size* value. Then, the PVFS library function divides the *block size* into units of the *stripe size*, and assigns the units to I/O nodes and communicates with the I/O daemons (iods) in the order described in a configuration file. This process is repeated until the specified *file size* is accessed. In this program, a meta data manager process, an I/O daemon process, or a client process is run on each node.

Clients execute this benchmarking program as follows. (Client processes access different files.)

1. Each client node starts writing a file simultaneously, and repeats writing three times without synchronization. We measured the elapsed time of the second write to get the write bandwidth. By skipping the first write, the effects of write buffers are eliminated. Linux uses unused memory as write buffer. I/O daemons receive a certain amount of written data and stores it in the write buffer, without writing to the hard disk. This write buffer is filled up during the first write, and then the data received (or an amount of data equal to it) is actually written to the hard disk. In doing the third write, the network utilized resources uniformly during the measurement.
2. After all processes finish the previous operation, all the client nodes start reading the written file simultaneously. Reading is repeated three times, and the elapsed time of the second read is measured.

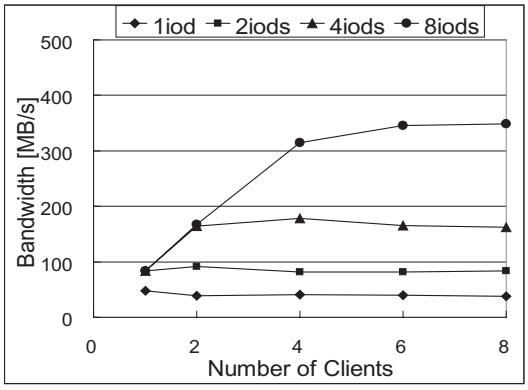

Figure 4. PVFS-TCP write bandwidth

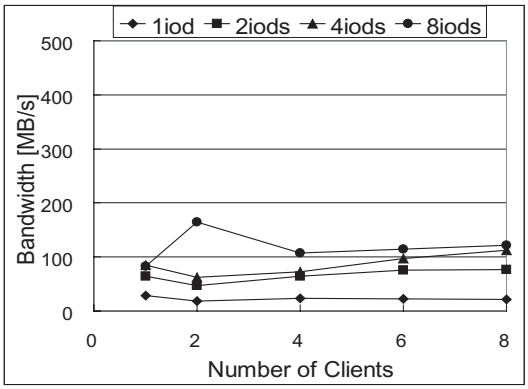

Figure 5. PVFS-TCP read bandwidth

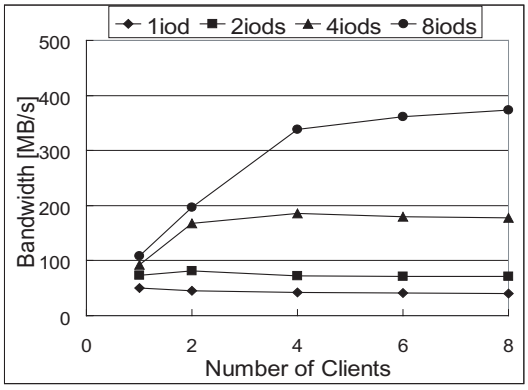

Figure 6. PVFS-PM write bandwidth

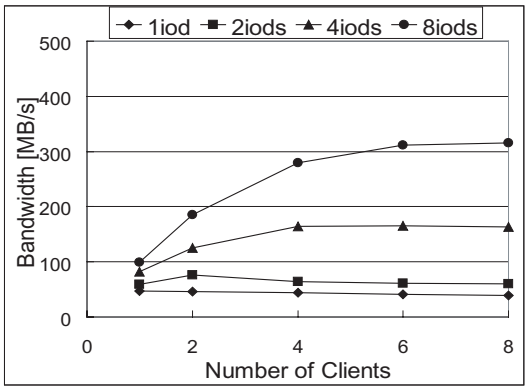

Figure 7. PVFS-PM read bandwidth

4.3. Evaluation Results

Bandwidth used for writing and reading with PVFS-TCP, and writing and reading with PVFS-PM are shown in Figures 4 and 5, and Figures 6 and 7, respectively. The file size was 2 Gbytes and the stripe size was 64 Kbytes. Bandwidths shown are the aggregate bandwidth of all clients. In this measurement, the bandwidth of each client was almost the same. The highest bandwidth with PVFS-TCP was 349 Mbytes/sec (8 iods, 8 clients) and 164 Mbytes/sec (8 iods, 2 clients) for writing and reading, respectively. The highest bandwidth with PVFS-PM was 373 Mbytes/sec (8 iods, 8 clients) and 316 Mbytes/sec (8 iods, 8 clients) for writing and reading, respectively.

Without the effects of write buffers, theoretical maximum bandwidth is limited by the aggregate network bandwidth or the aggregate hard disk I/O bandwidth, whichever is smaller. Gigabit Ethernet, the link bandwidth of which is 125 Mbytes/sec, is used as the network, and the disk I/O bandwidth of each node is about 50 Mbytes/sec. Therefore, the aggregate network bandwidth is calculated as

$$\textit{number of clients} \times 125 \text{ Mbytes/sec},$$

and the aggregate hard disk bandwidth is calculated as

$$\textit{number of iods} \times 50 \text{ Mbytes/sec}.$$

When the number of iods is smaller than the number of clients, the network bandwidth is limited by the number of iods. However, in such a case, the total bandwidth is limited by the hard disk bandwidth. Figure 8 shows the theoretical maximum bandwidth and the measured bandwidth when the number of iods is 8.

Considering that the theoretical maximum bandwidth is calculated without taking any overhead into account, PVFS-PM almost fully utilizes the bandwidth of the network and hard disks. The read bandwidth of PVFS-TCP does not scale to the number of client nodes. The overhead of TCP alone is not sufficient to explain this large bandwidth difference between PVFS-TCP and PVFS-PM. Further investigation is needed to identify the cause of this bandwidth degradation with PVFS-TCP.

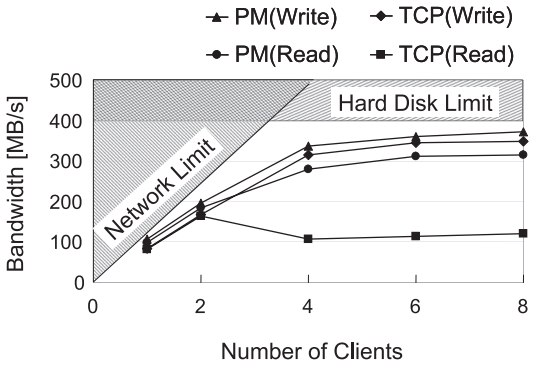

Figure 8. Comparison of theoretical maximum and measured bandwidth

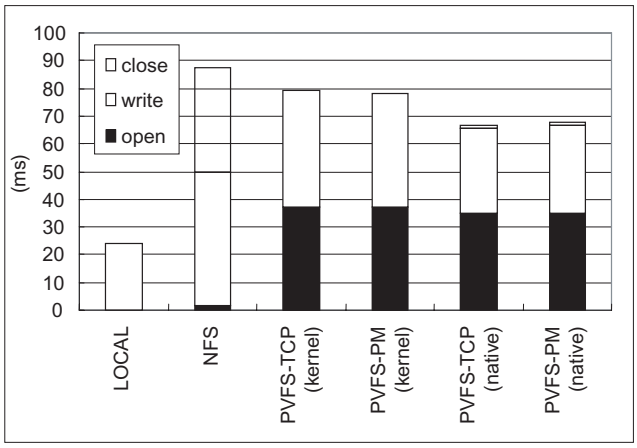

Figure 9. Breakdown of write accesses for 1M-byte file

5. Performance evaluation for small files

5.1. Breakdown of file creation and file access

Figure 9 and Figure 10 shows the breakdown of the file creation and file access time with 1 Mbyte file size. When the file size is changed, the `open()` and `close()` time is almost the same and the `read()/write()` time is changed in proportion to the file size. We compared the six types of file systems; local, NFS, PVFS-TCP kernel, PVFS-PM kernel, PVFS-TCP native and PVFS-PM native. The local is the result of the file I/O on the local disk, and the NFS is the result of the file I/O on the NFS file system. The PVFS kernel is the result of the file I/O on the PVFS file system that is mounted using the kernel module, and the PVFS native is the result of file I/O using the native PVFS API. In the results, PVFS uses a mgr, an iod and a client, which are run on the different hosts. There is no major difference between PVFS-TCP and PVFS-PM.

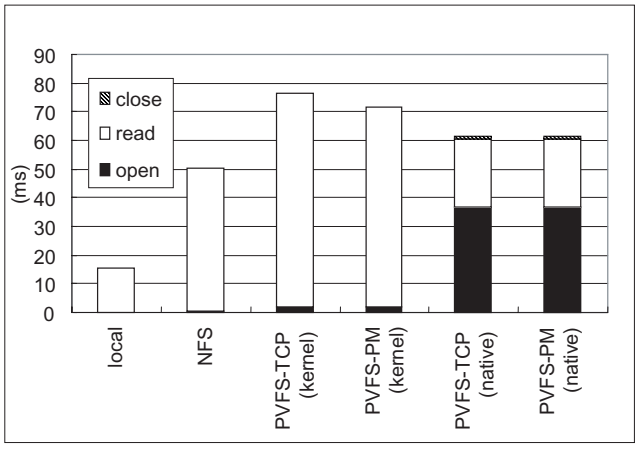

Figure 10. Breakdown of read accesses for 1M-byte file

PVFS takes about 37 msec in `open()` for both reading and writing, actually which is comparable with the time for `read()` and `write()` in this case. In the result, PVFS is about three times slower than the local in write and four times slower than the local in read. It is a little faster than the NFS in write and a little slower than the NFS in read. When writing, NFS spends a lot of time in `close()` operations compared to others. For writing, NFS is provided with a write buffer, and thus a part of the actual writing operation to the disk may be postponed and processed when `close()` is invoked.

The breakdowns of write access on PVFS native and PVFS kernel are similar. The total writing time of PVFS kernel is 1.2 times larger than that of the PVFS native. This may be caused by the overhead of the kernel module of VFS. On the other hand, the breakdown of read access in PVFS kernel is quite different from that of PVFS native. The `open()` time of PVFS kernel is quite a bit smaller than that of PVFS native, but the `read()` time of PVFS kernel is more than three times larger than that of PVFS native. This is caused by the difference of file management. PVFS native call always obtains the file system metadata from a meta data manager when opening a file regardless of the open mode for reading or writing because there is no metadata cache at the client side. On the other hand, the PVFS kernel module utilizes file system inodes in the Linux VFS at the client side, and does not need to obtain the updated file system metadata especially when opening a file for only reading not for writing. That is why opening a file for reading takes only 0.8 msec using the PVFS kernel module, while it takes 38 msec using the PVFS native API.

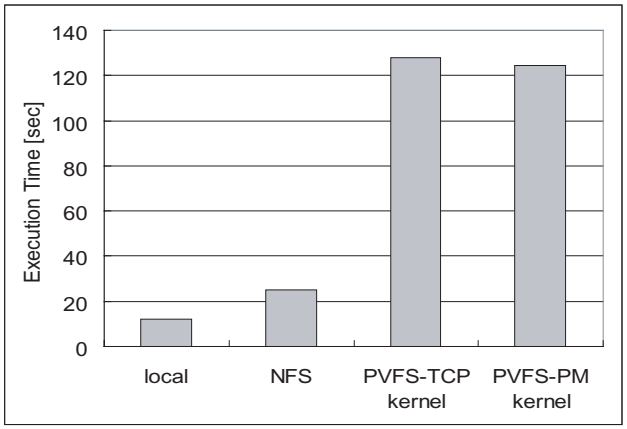

Figure 11. Building time of the PVFS version 1.5.5

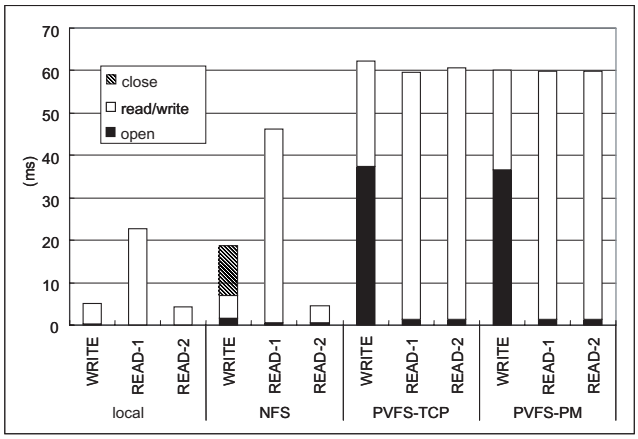

Figure 12. File cache effects in small file access

5.2. PVFS compilation and file cache effect

From the above breakdown, it is obvious that the performance of small files on PVFS is lower than the local and NFS file systems because of the overhead of `open()`. We also evaluated the performance by compiling PVFS source code on the file systems. By using the kernel module of PVFS, we can use the PVFS file system via the VFS interface. We measured the execution time of the *make* command which compiles the pvfs-1.5.5 source tree on local, NFS, PVFS-TCP and PVFS-PM file systems. It consists of 68 C language source files, of which the average file size is 5 Kbytes. The file sizes differ from 300 bytes to 57 Kbytes. Figure 11 shows the results.

Here, PVFS uses one iod node. While the execution time on NFS is double that of local, PVFS, both of PVFS-TCP and PVFS-PM, is about ten times that of local.

The poor performance of the PVFS seen in the compilation process comes from the effect of the file cache.

Figure 12 shows the breakdown of file creation and read access for relatively small number of small files which fits the in-core cache enough. In this evaluation, twenty 1-MB files are created, all of which will be read twice. Between the write test and the first read test, all in-core cache is copied to disk using fsync(2). In the local file system, the bandwidth of write and the second read achieved about 200 Mbytes/sec on average due to the in-core cache, while the bandwidth of the first read achieved only 44 Mbytes/sec. In the case of the NFS, the second read performed almost the same as the case of the local file system. On the other hand, the PVFS kernel module does not utilize the in-core cache at all even in the second read case, which makes the difference of the performance more than 12 times compared with both the local file system and the NFS.

This result shows that the current PVFS is not suitable for daily use, such as compilation of small files.

6. Related Work

Vollestad [8] ported PVFS to SCI (Scalable Coherent Interface). He measured the result using 2 iods and 1 client, and achieved 52.4 Mbytes/sec and 29.9 Mbytes/sec for write and read, respectively.

Carns et. al. [2] evaluated the performance of PVFS on the Chiba City cluster using TCP/IP on Myrinet [5]. However, their Myrinet only achieves 37.7 Mbytes/sec as measured by the `ttcp` test. Therefore, with 8 iods, the aggregate bandwidth of PVFS is 180 to 255 Mbytes/sec for read and write, respectively. They achieved 687 Mbytes/sec aggregate read bandwidth using 32 iods and 28 clients.

In [1], the performance of PVFS over TCP/IP is evaluated, using Myrinet. TCP on top of Myricom's GM message passing system is used. They achieved about 260 Mbytes/sec aggregate bandwidth when the number of iods is seven. Since they assigned both an iod and a client to each node, the result can not be directly compared with our result. No performance degradation such as that shown in our measurement of PVFS-TCP was reported.

Mache et. al. [4] reported that they achieved over 1 Gbyte/sec I/O throughput by using a 32-node cluster, Gigabit Ethernet and PVFS. However, most disk accesses were local in this benchmark.

7. Conclusion

We implemented a cluster file system, PVFS-PM, on the SCore cluster system software. Compared with PVFS-TCP, PVFS-PM shows scalability up to eight iods, and improved

the performance by factors of 1.07 and 1.93 for writing and reading, respectively, with eight iods. PVFS-PM provides users a high-performance cluster file system which fully utilizes the network and hard disk performance.

On the other hand, the performance of the meta data manager is insufficient for use as a general file system, and improvement of the performance seems to be necessary in future.

We evaluated PVFS performance using Gigabit Ethernet. We plan to measure the performance using Myrinet both with TCP and with PMv2 to investigate the essential advantage of using a light-weight communication library for PVFS implementation.

Since PVFS-PM is implemented using PMv2 APIs, it can be made available on a cluster connected by Gigabit Ethernet and Myrinet.

Acknowledgment

We would like to acknowledge Mr. Satoshi Sekiguchi, Director of the Grid Technology Research Center, AIST, for supporting the research presented in this report.

References

[1] A. W. Apon, P. D. Wolinski, and G. M. Amerson. Sensitivity of Cluster File System Access to I/O Server Selection. In *Proceedings of CCGrid2002*, pages 183–192, May 2002.

[2] P. H. Carns, W. B. L. III, R. B. Ross, and R. 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.

[3] A. Hori, H. Tezuka, and Y. Ishikawa. User-level Parallel Operating System for Clustered Commodity Computers. In *Proceedings of Cluster Computing Conference 1997*, March 1997.

[4] J. Mache, J. Bower-Cooley, J. Guchereau, P. Thomas, and M. Wilkinson. How to achieve 1 gbyte/sec i/o throughput with commodity ide disks. In *Poster presentation of SC2001*, November 2001.

[5] Myricom, Inc. *http://www.myri.com/*.

[6] PC Cluster Consortium. *http://www.pccluster.org/*.

[7] S. Sumimoto. *A Study of High Performance Communication Using a Commodity Network of Parallel Computers*. PhD thesis, Keio, 2000.

[8] J. E. Vollestad. A high performance cluster file system using sci. In *Master's Thesis, Department of Informatics, University of Oslo*, 2002.