Design and Implementation of the Cooperative Cache for PVFS

In-Chul Hwang, Hojoong Kim, Hanjo Jung, Dong-Hwan Kim, Hojin Ghim, Seung-Ryoul Maeng, and Jung-Wan Cho

Division of Computer Science, Dept. of Electrical Engineering & Computer Science, KAIST, 373-1 Kusung-dong Yusong-gu, Taejon, 305-701, Republic of Korea
{ichwang, hjkim, hanjo, dhkim, hojin, maeng, jwcho}
@calab.kaist.ac.kr

Abstract. Recently, there have been many efforts to get high performance in cluster computing with inexpensive PCs connected through high-speed networks. Some of them were to provide high bandwidth and parallelism in file service using a distributed file system. Other researches for distributed file systems include the cooperative cache that reduces servers' load and improves overall performance. The cooperative cache shares file caches among clients so that a client can request a file to another client, not to the server, through inter-client message passing. In various distributed file systems, PVFS (Parallel Virtual File System) provides high performance with parallel I/O in Linux widely used in cluster computing. However, PVFS doesn't support any file cache facility. This paper describes the design and implementation of the cooperative cache for PVFS (Coopc-PVFS). We show the efficiency of Coopc-PVFS in comparison to original PVFS. As a result, the response time of Coopc-PVFS is shorter than or similar to that of original PVFS.

1 Introduction

Recently, there have been many efforts to get high performance in cluster computing with inexpensive PCs connected through high-speed networks. It is necessary to connect PCs with efficient inter-connection network and to support applications efficiently in operating system to get high performance. Among these efforts, there have been many researches about distributed file systems which access disks much slower than any other component in cluster computing.

Among researches for distributed file systems, the cooperative cache [4,5,6] was proposed to reduce servers' load and to get high performance. Because the access time of another client's memory is faster than that of a server's disk, to get the block from another client's memory has faster response time than to get a block from a server's disk. In the cooperative cache, a client finds a block first from its own file system cache, and then the other clients' file system caches before finding the block from servers' disks.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3036, pp. 43–50, 2004.

In various distributed file systems, PVFS (Parallel Virtual File System) [1,2], which supports parallel I/O on Linux which is widely used in cluster computing, was developed in Clemson University. PVFS can get high bandwidth by stripping files over I/O servers. However, PVFS doesn't support any file system caching facility but only supports applications with transfer of data from/to I/O servers.

In this paper, we describe the design and implementation of the cooperative cache for PVFS (Coopc-PVFS). We also present various performance results with Coopc-PVFS and PVFS on the CAN cluster [8] at KAIST. We present the result of executing a simple matrix multiplication program. Then, we show the result for the BTIO benchmark programs [9].

The rest of this paper is organized as follows. In the next section, we discuss the related work of PVFS and cooperative cache. In section 3, we describe the design and implementation of Coopc-PVFS. In section 4, we present and discuss the performance results. In section 5, we summarize major contributions of this work and discuss future work.

2 Related Work

2.1 PVFS (Parallel Virtual File System)

PVFS [1,2], which supports parallel I/O on Linux widely used in cluster computing, was developed in Clemson university.

PVFS is composed of compute nodes, single metadata manager and I/O servers. The compute nodes are clients that use PVFS services. The metadata manager manages the metadata of PVFS files. The I/O servers store the actual data of PVFS files. In PVFS, a file data is stripped over I/O servers.

There are two schemes by which users can access files in PVFS. First, users can access files by recompiling their application codes with the PVFS user-level library – the PVFS library scheme. Another scheme is that users can access files through UNIX I/O system call using the PVFS kernel module – the PVFS kernel module scheme

There was a research for the file system caching effect of PVFS. Vilayannur et al. [3] designed and implemented a file system cache of a client in the PVFS library scheme. They showed that a file system cache in a client is efficient if many applications in the client share files among them. But their research was limited to a file system cache in a single node. Because many users share files in cluster environments, the cooperative cache is more appropriate than a file system cache in a client.

2.2 Cooperative Cache

The cooperative cache [4,5,6] was proposed to reduce servers' load and to get high performance. In the cooperative cache, if a file system cache in a client doesn't handle a request to a file, the client sends the request to the other client's cache that caches the file rather than to the server because the access time of another client's memory is faster than that of the server's disk. Servers' load can be reduced in the

cooperative cache so that it is scalable as the number of clients increases. Because there is much more memory in the cooperative cache than in a single file system cache, the cooperative cache can handle more requests and improve overall system performance.

There have been many researches about the cooperative caching. Dahlin et al. [4] suggested the efficient cache management scheme called N-chance algorithm, Feeley et al. [5] suggested another efficient cache management scheme called modified N-chance algorithm in GMS (Global Memory Service). Sarkar et al. [6] suggested the hint-based cooperative caching to reduce the management overhead using hint. Thus, the hint-based cooperative cache is scalable and can be adopted in the large-scale system such as cluster computer. Because Sarkar's idea uses a file as a file system management unit, it can not be applicable to parallel file systems in which many users share large files concurrently.

3 Design and Implementation of Coopc-PVFS

3.1 Overview of Coopc-PVFS

In the PVFS kernel module scheme, an application reads a file from I/O servers through the PVFS kernel module without a file system caching facility. We added the cooperative caching to the PVFS kernel module scheme in computing nodes.

In figure 1, we present the workflow of Coopc-PVFS added in the PVFS kernel module scheme.

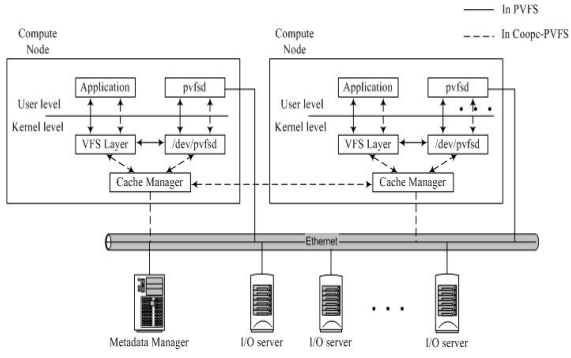

Fig. 1. Workflow of Coopc-PVFS

In the cooperative cache added to PVFS kernel module scheme, when an application reads a file, the cache manager in the client looks up whether the requested block is in its own cache. If the block is in the cache manager, the cache manager copies the block to the application. If the block is not found in its own cache manager, the cache manager looks up whether there is any client that caches the block. If the block is found in the other clients' cache managers, the cache manager gets the block from one of them, and then caches the block and copies the block to the application. If the

block is not found in other clients' cache manager, the cache manager gets the block from I/O servers, and then caches the block and copies the block to the application.

3.2 Design of Coopc-PVFS

3.2.1 Information Management

Because of large overhead to maintain accurate information about cached blocks, we designed Coopc-PVFS as a hint-based cooperative cache. To maintain the hint – opened clients list, we added new function to the metadata manager to keep the clients list that contains information of clients that opened the file before. Whenever a client opens a file, the client gets both the metadata and the opened clients list of the file from the metadata manager.

To accurately look up a block whether other clients have it, the client must know the information about cached blocks in other clients. To maintain this information, we used the methods like below:

- In PVFS, when an application opens a file, the application gets metadata of the file from the metadata manager. Using this mechanism, the metadata manager manages the hint per file – the IPs of the clients which opened the file before, named opened clients list. When an application opens a file in Coopc-PVFS, the application gets not only the metadata of the file but also opened clients list.
- To maintain the information about cached blocks, when an application reads a block that is not in its own cache, the cache manager exchanges its own information (bitmap) about cached blocks with the information of other client's cache manager. After many accesses to the file which clients exchange the information with each other, the client maintains approximately accurate information about cached blocks in Coopc-PVFS.
- When an application closes a file, the cache manager doesn't do anything. Because the cache manager caches blocks of the file, the metadata manager remains the client which closes the file in opened clients list.

Unlike previous hint-based cooperative cache research [6], we managed information and cached blocks per block, not per file. Because many clients share large files among them in a parallel file system, it is more adaptable to manage information and to cache per block than per file in Coopc-PVFS.

3.2.2 Consistency

In PVFS, all the accesses to files go through the I/O servers. To preserve the consistency likewise in PVFS, the cache manager must invalidate blocks cached in other clients before writing the block to the I/O server in Coopc-PVFS. To do that, whenever an application writes a block, the cache manager sends the block invalidation propagation request to the metadata manager before sending the written block to the I/O server. When the metadata manager gets the block invalidation propagation re-

quest, it sends the block invalidation messages to the clients in opened clients list of the file then all of the clients that receive the block invalidation message invalidate the block. Therefore, all of cached blocks in other clients are invalidated before sending the written block to the I/O server and we can preserve the consistency in Coopc-PVFS the same as in PVFS.

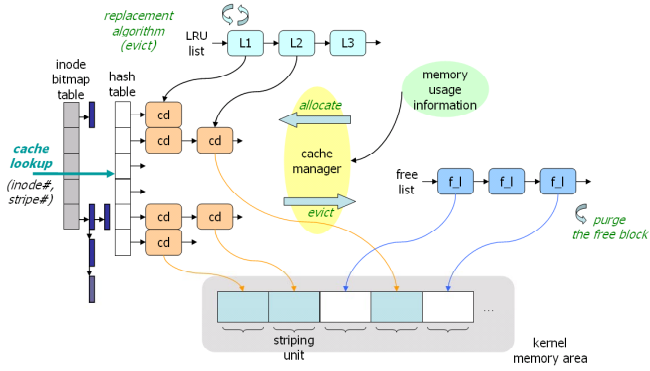

Fig. 2. Data structures used by a cache manager in Coopc-PVFS

3.3 Implementation of Coopc-PVFS

In figure 2, we present data structures used by the cache manager in Coopc-PVFS.

We implemented most of data structures using linked lists in Coopc-PVFS for dynamic addition and deletion of entries. For replacement of the cached block, we managed the LRU list. For unused blocks allocated from system memory, we managed the free list. Each cached block is managed by the size of stripping unit used in PVFS. In PVFS, the size of stripping unit is 64KB by default and it can be changed for various system configurations.

In the metadata manager, we manage opened clients lists of files using linked lists. Each opened clients list of a file has its own lock in order to access it concurrently.

For allocating cache blocks from system memory, we didn't use the page cache system in Linux for the cache manager. If we use the page cache system in Linux, we don't know which blocks are cached in the cache manager – If we use, cache blocks can be freed by the Linux memory manager so that we can't know about that event. Therefore, we must allocate cache blocks from kernel memory and do memory management for cache blocks. For doing memory management, we implemented the cache replacement manager which can do memory management according to the amount of free kernel memory in the system as a kernel thread.

4 Performance Evaluation

We used CAN cluster [8] in KAIST to evaluate the performance of Coopc-PVFS. The system configuration of CAN cluster is presented in table 1.

Table 1. System configuration

CPU	Pentium IV 1.8GHz
Memory	512MByte 266MHz DDR
Disk	IBM 60G 7200rpm
Network	3c996B-T(Gigabit Ethernet) 3c17701-ME(24port Gigabit Ethernet Switch)
OS , PVFS	Linux(Kernel version 2.4.18) , 1.5.3

The metadata manager was allocated in one node and the I/O server was allocated in another node. And one to four other clients were used to execute the test applications –a simple matrix multiplication program and BTIO benchmark programs. Each program operates like below:

- Matrix multiplication program: Applications in four clients read two input files of 1024*1024 matrix and calculate the matrix multiplication and write the result to the output file.
- BTIO benchmark programs: BTIO [9] is a parallel file system benchmark. BTIO contains four programs. In table 2, we present each four programs. We can evaluate the parallel I/O performance of Coopc-PVFS using four clients with smallest sized class s in BTIO.

Table 2. BTIO benchmark programs

Full (*mpi_io_full*)	MPI I/O with collective buffering
Simple (*mpi_io_simple*)	MPI I/O without collective buffering
Fortran(*fortran_io*)	Fortran 77 file operations used
Epi (*ep_io*)	Each process writes the data belonging to its part of the domain to a separate file

Table 3. Execution time of the matrix multiplication program

PVFS	I/O server doesn't caching data(*iod_cool*)	209.174 secs
	I/O server caches data(*iod_hot*)	128.283 secs
Coopc-PVFS	anyone doesn't cache data(*iod_cool*)	120.451 secs
	I/O server cache data(*iod_hot*)	120.005 secs
	Coopc-PVFS cache data(*coopc_hot*)	120.029 secs

4.1 Execution Time of Matrix Multiplication Program

The execution time of the matrix multiplication program is in table 3.

To analysis of total read/write time of this program, we present the time breakdown of average execution time in figure 3.

The matrix multiplication program is a read-dominant program so that total read time is much longer than total write time. In Coopc-PVFS, we can reduce the read time to approximately zero because the file is cached in Coopc-PVFS after once a file is read. When the I/O server doesn't cache the file in PVFS, the waiting time is much

larger than any other case because the variation of read time is much larger than any other case. The write time in Coopc-PVFS is a little longer than that in PVFS because the write in Coopc-PVFS has slightly overhead than in PVFS.

Fig. 3. Execution time breakdown of the matrix multiplication program

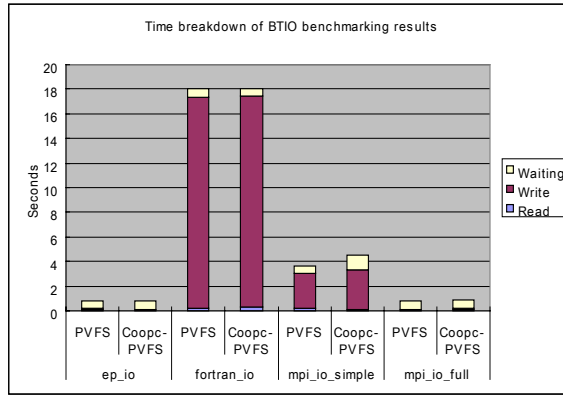

Fig. 4. Execution time breakdown of BTIO benchmark programs

4.2 Performance Evaluation using Benchmark Programs

In figure 4, we present the time breakdown of BTIO benchmarking results.

BTIO benchmark programs are write-dominant programs so that total write time is much longer than total read time in the results. Using MPI, we can get much shorter write time. Collective I/O reduces almost part of write time. In most cases, write time of Coopc-PVFS is longer than in PVFS because the write in Coopc-PVFS has more overhead than the write in PVFS and read time of Coopc-PVFS is shorter than in PVFS because cli-ents cache all files in Coopc-PVFS. Totally, the execution time in

Coopc-PVFS is a little longer than in PVFS. Therefore, we can know that the performance improvement of writing is needed in Coopc-PVFS.

5 Conclusion and Future Work

In this paper, we describe the design and implementation of the cooperative cache for efficient data sharing that is not supported in PVFS.

We evaluated Coopc-PVFS with many programs. In a matrix multiplication program, we can execute the program in Coopc-PVFS faster than in PVFS about 6%~50%. When we executed write-dominant BTIO benchmark programs, we can know that using Coopc-PVFS has a little worse than using PVFS.

In the future, we will evaluate the performance of Coopc-PVFS using many scientific applications in cluster. Using the cooperative cache can improve the performance of reading, but the cooperative cache can not improve the performance of writing. Therefore, we will support the write buffering and develop new write schemes to improve the write performance. And we will adopt the collective I/O request technique which the cache manager sends many requests at a time instead of sending a request at a time.

References

1. P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A Parallel File System For Linux Clusters", Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000, pp. 317-327
2. R. B. Ross, "Providing Parallel I/O on Linux Clusters" ,Second Annual Linux Storage Management Workshop, Miami, FL, October 2000.
3. M.Vilayannur,M.Kandemir, A.Sivasubramaniam, "Kernel-Level Caching for Optimizing I/O by Exploiting Inter-Application Data Sharing", IEEE International Conference on Cluster Computing (CLUSTER'02),September 2002
4. Dahlin, M., Wang, R., Anderson, T., and Patterson, D. 1994. "Cooperative Caching: Using remote client memory to improve file system performance", In Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation. USENIX Assoc., Berkeley, CA, 267-280
5. Feeley, M. J., Morgan, W. E., Pighin, F. H., Karlin, A. R., and Levy, H. M. 1995. "Implementing global memory management in a workstation cluster", In Proceedings of the 15th symposium on Operating System Principles(SOSP). ACM Press, New york, NY, 201-212
6. Prasenjit Sarkar , John Hartman, "Efficient cooperative caching using hints", Proceedings of the second USENIX symposium on Operating systems design and implementation, p.35-46, October 29-November 01, 1996, Seattle, Washington, United States
7. "Linux Kernel Threads in Device Drivers", `http://www.scs.ch/~frey/linux/kernelthreads.html`
8. Can cluster, http://camars.kaist.ac.kr/~nrl
9. Parkson Wong, Rob F. Van der Wijngaart, NAS Parallel Benchmark I/O Version 2.4, NAS Technical Report NAS-03-002, NASA Ames Research Center, Moffett Field, CA 94035-1000