Mechanisms of client-side caching on high performance computers

Yu-Sheng Liu¹, Guang-Ming Liu¹ and Jie Yu¹

¹ College of Computer, National University of Defense Technology, Changsha, Hunan Province, China

Abstract. With the continuous development of high-performance computing (HPC), I/O demand for high-performance computing applications is growing rapidly, and storage systems are under severe I/O pressure. In the file system client developed based on FUSE, VFS page cache and FUSE kernel became the important bottlenecks restricting system performance. Cache technology is a classic technology to alleviate the I/O bottleneck of HPC systems. This paper proposes a client-side caching mechanism that takes the advantage of high-performance computers supporting RDMA communication to improve the I/O performance of masFS client. Finally, we deployed and tested the improved file system client on the TH-1A system. The test results show that the client cache can significantly improve the application client I/O performance. For a single client, it increases both sequential read and write speed by a factor of 4.8 and 2.3.

1. Introduction

The ever-increasing complexity and scale of computing in HPC applications, many applications are becoming I/O-intensive, like oil seismic applications and weather research and forecasting, which tend to do big I/O, sequential reads, and require file systems to achieve high bandwidth and low latency. The client-side caching system uses a fast storage device on the compute node to build a cache space close to the application. The application can use the client-side cache to temporarily store data. The page cache in distributed file system client will limit the performance of typical HPC application. This paper use masFS (File System Based on Memory and SSD) as the storage system[1], there are severe performance bottlenecks in the page cache and the FUSE kernel during I/O in masFS client. This paper proposes a client-side caching mechanism that is managed by the RDMA buffer to improve the I/O performance of the application. The test results show that the client-side caching mechanism proposed in this paper can significantly improve the sequential read/write aggregate bandwidth of the client. The maximum sequential read and write bandwidth of the client can reach 4.8 and 2.3 times of the original system respectively.

2. Related Work

2.1 Overview of masFS

In recent years, the continuous development of high-performance applications has led to continuous improvement of distributed file systems. Among them, MooseFS (MFS) is an open source system based on GFS architecture, which is widely used in large-scale distributed cluster. masFS improved
MooseFS, use node memory and SSD as storage, and optimized the.socket communication mechanism of MFS with RDMA communication. The architecture of masFS mainly consists of three components:

1. Master, the main server responsible for managing the entire Masfs file system. It stores metadata information of all files in the file system.
2. Data Servers (referred to as DS) is the object storage server of the file system.
3. The client is mounted on the node that provides POSIX semantics through the FUSE (File system in Userspace) interface.

2.2 I/O features of typical high performance applications
HPC applications tend to be highly phased, use intermediate output results as next input iterations. HPC applications typically have two important I/O features: (1) sequential reads of large files and no repeated reading of the same file; (2) the resulting file is read again as input.

The oil seismic application is a typical data-intensive application. The gathering program is to reorder and partially superimpose the imaging data. Its I/O process is: (1) each process first reads the gather file data independently; (2) sent the calculation of the resulting imaging data to the main process, and the main process written out to the image files. WRF is a numerical climate prediction application. The WRF workflow is a typical data-intensive application, consisting of multiple simulation cycles, each cycle consisting of two data I/O processes: (1) reading two different data sets and passing WRF applications to perform simulations and output two result data sets; (2) exchange part of the data of the first stage result data set as the input data set into the next simulation cycle.

2.3 Related research
Cache technology has been widely studied as a technology to alleviate the I/O bottleneck of high-performance computer systems. Some typical parallel file systems such as Lustre and Ceph support client-side caching.

Dong et al. build a general distributed cache system SFDC on a supercomputer based on the Memcached extension[2]. SFDC distributes the data hash to all nodes, and obtains the data location by calculating the hash value, then obtain data through the network, it can load balance effectively, but need to get data from the far end. Congiu G et al. solve the problem that the aggregated I/O performance in parallel programs is limited by the slowest process. By adding the MPI-IO interface function, the SSD of the compute node is used to cache the data of the aggregator process, thereby improving the aggregation I/O performance [3].

3. I/O process in masFS client
When the client processes the read/write request, first the VFS allocates a corresponding number of pages with a granularity of 4 KB from the page cache to receiving the request data, and the request is sent to the FUSE kernel module and then split into a FUSE format request. The client designs a multi-channel strategy when processing the request: (1) when the request data is greater than 1 MB, the RBB (RDMA Buffer Block) is directly allocated from the RBP (RDMA Buffer Pool) for receiving data; (2) when the request data is less than 1 MB, then allocate memory space as a RBB to buffering data; (3) when the request data is less than 32 KB, the socket buffer is used. The client receives the data and status from the RBB. Return to the Fuse kernel Buffer, and finally return to the application via VFS page cache.

Typically, VFS uses the page cache to prefetch file data. The page size is 4KB, and the prefetch window grows exponentially according to the continuous condition of the requested data (up to 128KB). Page cache increases the number of copies during data transmission. Moreover, typical large-scale scientific applications tend to do large I/O, showing the characteristics of sequential read and read after write of large files. Page cache is difficult to bring performance optimization.

In the framework of FUSE, the request is forwarded to the FUSE kernel via VFS. After analyzing the FUSE kernel, we found that the FUSE kernel splits the VFS forwarded request into a FUSE format sub-request of 128KB size (controlled by the parameter FUSE_MAX_PAGES_PER_REQ). After the
sub-request arrives at the client, it needs to obtain data from the underlying file system or the remote storage device, which will cause a certain data acquisition delay, resulting in an inevitable interval between the sub-requests. Especially for typical large-scale scientific applications that tend to do large I/O requests, the latency caused by the FUSE kernel splits request mechanism can severely impact the application's I/O performance.

4. Optimize I/O performance with client-side cache

4.1 Architecture of client cache

To fully utilize the mechanism of RDMA data transfer between masFS client and DS, the client cache proposed in this paper is managed in a unified manner with the RDMA buffer, and is organized by the CBB (Client Buffer Block). Three LRU lists related to read and write are designed: Prefetch list, Read LRU list and Write LRU list, so that different management strategies are used to process read and write requests. The Prefetch list holds the CBB blocks prefetched to the client cache. The Read LRU list holds all the CBB blocks that are read hit and timed out in the Prefetch list. The Write LRU list holds the CBB blocks associated with all write requests.

Figure 1. Organization of Client cache

Figure 1 depicts the spatial organization and management of CBB when the client cache processes read and write requests. In order to describe the state of the CBB in the client cache more detailly, we add a free table, the free table holds the CBB in the client cache that is available for allocation in idle state. Each CBB is only stored in one of the lists at the same time, and all used CBBs are recorded in the hash table.

For the read/write request, first query the hash table to see whether the relevant CBB caches the requested data. If read/write hits, the corresponding CBB is added to the read/write LRU header; if read/write miss, corresponding CBB blocks is allocated from the free list to receive the request data, and then chained to the read/write LRU header, and record relevant CBB in the hash table. At the same time, the pre-read related parameters are determined according to the continuity of current read request and the previous read request.

4.2 Client-side caching I/O optimization mechanism

After adding the client cache, the page cache is discarded by mounting the DIRECT_IO option to reduce copies of data when read or write. High-performance applications tend to do large sequential I/O, thus the original read/write multi-channel strategy of masFS was eliminated, the client only reads/writes at the granularity of CBBs. RDMA communication is used to transmit data, and socket communication is used to transmit I/O requests. Figure 2 and 3 shows the client I/O flow after applying the client cache.

4.2.1 Read optimization
For read requests, the application bypasses the page cache and sends the request directly to the FUSE kernel. After receiving the request, the client first queries the client cache whether the requested data has been cached. If read hits, the data is directly retrieved from the corresponding CBB block in the client cache and returned to the application; if read miss, the corresponding free CBB block is allocated from the client cache for the read request, and the request is sent to the DS through the socket. After that, the DS will send the request data to the corresponding CBB block through RDMA, and finally the data returned to the client through the FUSE kernel. The prefetch parameters can be set more reasonable and efficiently by deprecating the page cache and transferring the pre-reading mechanism to the client-side cache. Pre-reading on the client can hide the processing delay between the FUSE sub-requests, thereby effectively improving the application’s read performance.

4.2.2 Write optimization
For write requests, the application bypasses the page cache and sends the write request data to the FUSE kernel buffer. The FUSE kernel splits the data and sends it directly to the client cache. After the client receives the write request data, it first queries the client cache. If it hits, the written data is modified into the corresponding CBB block directly; if write miss, the free CBB block is allocated to receive the write data. When the client cache needs to flush the dirty CBB block data back to the remote DS, the data to be written is transferred to the DS by RDMA communication. The client cache acts as a Write Cache to improve write performance.
4.3 Parameter optimization settings
In order to further improve the performance optimization effect of the client cache, it is necessary to optimize some important parameters of the client. The masFS client provides a POSIX interface to the user program based on FUSE. The FUSE kernel splits the request sent from the VFS into a 128KB size (controlled by the parameter FUSE_MAX_PAGES_PER_REQ) and forwards it to the client, which seriously affects the performance. According to the optimal value of the transfer size between DS and client [[1]], we adjust the FUSE kernel request split granularity to 1 MB, and also set the CBB block size to 1 MB. The request interval between the VFS and FUSE kernels to read and write 1MB of data is about 246.9μs [4], so the prefetch CBB timeout delay is set to a multiple of the request interval and the prefetch window. The dirty data capacity is set to 32MB according to Lustre [5].

5. Performance Evaluation

5.1 Experiment environment
Based on TH-1A supercomputer, following experiments are conducted. The masFS original version based on the VFS page cache and the version of the client cache with RDMA buffer are labeled as Pcache (Page cache) and Ccache (Client cache) respectively, both version are deployed on TH-1A to making comparison test. Use the compute node with 48GB of memory in the TH-1A as the DS and the compute node mounted with FUSE module as the client.

5.2 Results and analysis
We Use IOR benchmark to doing test, the file block size varied from 16KB to 8MB, and use 1, 2, 4, and 8 processes to read and write in parallel respectively. When testing Pcache performance, the client cache mechanism is turned off. The Ccache prefetch window is 4 CBB when testing read performance.

The test results are shown in Figure 4. Under the same processes and file block size, Ccache reads and writes better than Pcache. The Ccache read bandwidth is up to 4.8 times that of Pcache, and the write bandwidth is up to 2.3 times of Pcache. It can be seen that the pre-reading effect of Pcache is very limited. This is mainly because the FUSE kernel split the read request sent by the VFS and process the sub-requests synchronously, the read request received by the client is intermittent. FUSE increases the read delay. Moreover, the Pcache prefetch window is only 128KB. However, Ccache reads data from DS, then returns the data directly to the FUSE kernel, and then transfers it to the application, avoiding copying to the VFS page cache, reduced the latency, and prefetching on the client can effectively alleviate the performance impact caused by FUSE kernel’s split mechanism.

![Figure 4. Results of Client Read/Write Comparison Test](image-url)
When Pcache is turned on, the data to be written is first cached to Pcache, then sent to the RDMA buffer of the client through the FUSE kernel, and finally written back to DS through RDMA communication. The write data is copied twice, which seriously reduce the write bandwidth. When Pcache is turned off, the data to be written bypasses Pcache, then sent to the client’s Ccache directly via FUSE, and then written to DS, which reduces the write operation delay and improves the write bandwidth. However, when the FUSE kernel forwards VFS read/write sub-requests to the client, there is a certain interval between sub-requests, which increases the read/write operation delay additionally, and that will reduce the read/write bandwidth to some extent.

6. Conclusion and Future Work
This paper proposes a client-side caching mechanism based on RDMA buffer. In the distributed file system client developed based on FUSE, VFS page cache and FUSE kernel become important bottlenecks restricting the performance of high-performance applications. To this end, this paper focuses on the typical high-performance application I/O characteristics, taking advantages that high-performance computers support RDMA communication, we propose a client-side caching mechanism managed with RDMA buffer to improve the masFS client. The test result shows that the improved client’s I/O performance is significantly improved. The future work is to use techniques such as relaxed consistency, write combing to further optimize client-side performance.

References
[1] Liu X, Lu Y, Lu Y, et al. masFS: File System Based on Memory and SSD in Compute Nodes for High Performance Computers[C]// IEEE, International Conference on Parallel and Distributed Systems. IEEE, 2017:569-576.
[2] Dong W, Liu G, Yu J, et al. SFDC: File Access Pattern Aware Cache Framework for High-performance Computer[C]// IEEE, International Conference on High PERFORMANCE Computing and Communications. IEEE, 2015:342-350.
[3] Congiu G, Narasimhamurthy S, Sub T, et al. Improving Collective I/O Performance Using Non-volatile Memory Devices[C]// IEEE International Conference on CLUSTER Computing. IEEE, 2016:120-129.
[4] Xin Liu, Yu-tong Lu, Jie Yu, Peng-fei Wang, Jie-ting Wu, & Ying Lu. (2017).ONFS: Hierarchical Hybrid File System based on Memory, SSD, and HDD for High Performance Computers[J]. Frontiers of Information Technology & Electronic Engineering, 2017 18(12):1940-1971.
[5] Wang F, Oral S, Shipman G, et al. Understanding lustre filesystem internals [J].Oak Ridge National Laboratory, National Center for Computational Sciences, Tech. Rep. 2009..