# A Layout-aware Optimization Strategy for Collective I/O

Yong Chen [1] Huaiming Song [1] Rajeev Thakur [2] Xian-He Sun [1]
[1] Department of Computer Science, Illinois Institute of Technology
[2] Mathematics and Computer Science, Argonne National Laboratory
{yong.chen@iit.edu, huaiming.song@iit.edu, thakur@mcs.anl.gov, sun@iit.edu}

## ABSTRACT
In this study, we propose an optimization strategy to promote a better integration of the parallel I/O middleware and parallel file systems. We illustrate that a layout-aware optimization strategy can improve the performance of current collective I/O in parallel I/O system. We present the motivation, prototype design and initial verification of the proposed layout-aware optimization strategy. The analytical and initial experimental testing results demonstrate that the proposed strategy has a potential in improving the parallel I/O system performance.

## Categories and Subject Descriptors
C.4 [**Performance of Systems**]: Design studies. D.4.3 [**File Systems Management**]: Access methods.

## General Terms
Performance

## Keywords
Parallel I/O, MPI I/O, parallel file system, collective I/O, data layout, petascale file system

## 1. INTRODUCTION
High-performance computing (HPC) has achieved substantial computational performance improvement over the past decades [16]. However, while computing resources are making rapid progress, there is a significant gap between computational capability and data-access capability. Due to this gap, although computational resources are available, they have to stay idle waiting for data to arrive, which leads to a severe overall performance degradation [3][8]. Figure 1 shows the normalized performance improvement of the processor, memory (DRAM) and disk storage over the past decades. It can be seen that the data-access speed has been improving with a much slower pace than the computational performance. This slow performance improvement of data-access speed is predicted to continue in the near future. In the meantime, many HPC applications are becoming more and more data intensive [10]. Due to the growing performance disparity and emerging data intensive applications, I/O has become a critical performance bottleneck in HPC systems.

Parallel I/O middleware and parallel file systems are two critical components to provide high data-access bandwidth for HPC applications. However, historically, parallel I/O middleware and parallel file systems are developed separately with a separated modular design. This separation enhances the transparency between different parallel I/O components at distinct layers and eases the software implementation. Nevertheless, this separation can lead to a potential information gap between these two layers meantime. For instance, the collective I/O, one of the most important optimization strategies in parallel I/O middleware, often relies on the logical layout of file accesses from multiple processes, other than the physical data layout on file servers, because of lacking this information. On the other hand, it is the parallel file system that decides the physical layout of data among multiple file servers and thus decides the access latency and concurrency. There is a disparity between the information available in parallel I/O middleware and parallel file systems. In this study, we argue that it would be beneficial if we bridge this information gap and have a "well-matched" I/O. We propose to pass some of the data layout information from parallel file systems to parallel I/O middleware and rearrange accesses in a way that reduces access latency (exploiting both physical locality and parallelism). This strategy optimizes the existing widely used two-phase collective I/O design and fosters a better integration of parallel I/O middleware and file systems. The recent works in log-like reordering of accesses and intermediate library of rearranging accesses [1][7] have demonstrated the importance and significant potential of arranging data accesses in a proper manner. This optimization strategy requires understanding file system abstractions, gaining knowledge of disk storage, knowing the designs of high-level libraries, and making intelligent decisions. It is a challenging and tedious task for end-users. There is a research need to conduct such an optimization automatically and transparently to users.

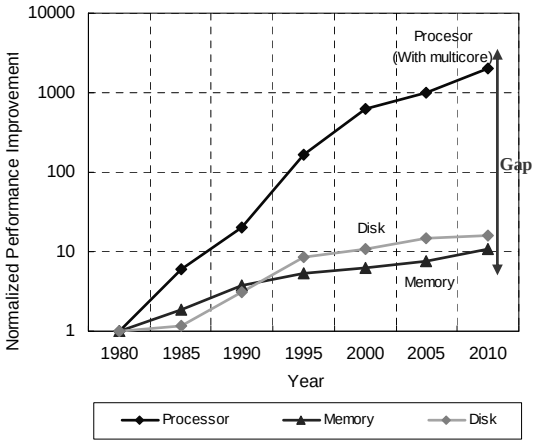



**Figure 1. Performance improvement trend of processor, memory and disk.**

The rest of this paper is organized as follows. We first briefly review collective I/O, one of the most important optimization


*HPDC'10*, June 20–25, 2010, Chicago, Illinois, USA.

strategies in parallel I/O systems, and its common two-phase implementation in Section 2. Section 3 introduces the basic idea of the layout-aware optimization strategy for collective I/O and the initial design. Section 4 presents the preliminary experimental and analytical results. Section 5 concludes this study and discusses ongoing and future work.

## 2. BACKGROUND

Many parallel applications require accessing large arrays from file servers and distribute the data among multiple processes in a certain manner. Even though each process may access several non-contiguous portions of a file, the requests of multiple processes are often interleaved and may constitute a large contiguous portion of a file together [14]. In order to achieve better I/O performance, a group of processes may cooperate with each other in reading or writing data in a collective and efficient way, which is known as *collective I/O*. In addition, the collective I/O can improve the I/O access efficiency by filtering the overlapping and redundant requests from multiple processes. Furthermore, it can reduce the number of file system calls (and thus the overhead involved) by combining and merging small requests.

The collective I/O is a general idea that exploits the correlations among accesses from multiple processes of a parallel application and optimizes its I/O accesses. It can be applied at many levels, such as disk level [6], server level [12] or client level [14]. In this study, we focus on parallel I/O middleware and the integration with parallel file systems. The collective I/O has been well implemented in the most popular MPI-IO middleware implementation, ROMIO [14]. If the user chooses collective I/O semantics and provides the entire access information of a group of processes to the underlying MPI-IO middleware, the MPI-IO implementation can improve I/O performance considerably by combining the requests of different processes and servicing the combined aggregate requests.

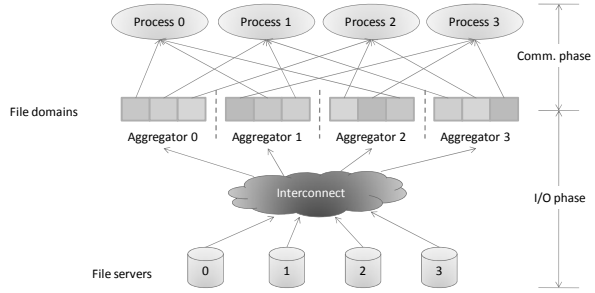



**Figure 2. Collective I/O and two-phase implementation**

The most popular method of implementing collective I/O is a *two-phase strategy* [11] (and its extension - a generalized two-phase I/O [15]). This strategy carries out collective I/O with separated I/O phase and data exchange phase (or communication phase). Figure 2 shows an example of two-phase collective I/O read. In this example, we assume all processes participate in the I/O phase (the processes participating I/O phase are called *aggregators* and the number of aggregators can be specified by users) and each process (also aggregator) has sufficient memory for temporary buffer. The two-phase I/O implementation has a first-round communication to let each aggregator knows the aggregated span of the I/O requests of all processes. The implementation then partitions the aggregated span of requests into multiple file domains with each aggregator responsible for carrying out I/O requests for its own file domain by using its temporary buffer. This phase is called the I/O phase. In the data exchange phase, each aggregator sends data to the requesting processes, and each process receives its required data from corresponding aggregators that fetch the data on behalf of it. Note that if the temporary buffer is limited, we may need multiple times of two-phase I/O to accomplish one collective I/O operation [14].

## 3. LAYOUT-AWARE OPTIMIZAITON STRATEGY FOR COLLECTIVE I/O

As shown in Figure 2 and explained in the previous section, a separated design of parallel I/O middleware and parallel file systems makes the current collective I/O strategy and implementation unaware of the physical layout of data on file servers and disks. The calculation of the span of the combined I/O requests and the creation of the file domains are based on logical partitions. Even though the file domain that each aggregator is responsible for carrying out I/O requests is logically contiguous, it does not translate to physically contiguous, as which is decided by the layout strategy of the underlying parallel file systems. The separated design of parallel I/O middleware and file systems and the information gap between them may limit the performance improvement potential of collective I/O.

We argue that it would be beneficial if we bridge the information gap between parallel I/O middleware and parallel file systems. Specifically, in this study, we propose to incorporate the physical layouts of data distribution among servers, the information from parallel file systems, with parallel I/O middleware, and rearrange file domain's partition and the requests from aggregators in a fashion that matches with the physical layout on servers. In addition, with the support of noncontiguous file system calls in advanced parallel file system such as in PVFS2 [9], we are able to keep the number of file system calls and overhead in constant. We call this strategy as a layout-aware optimization strategy for collective I/O. In essence, this proposed strategy fosters a better integration of the parallel I/O middleware and parallel file systems by revealing certain information with each other. As demonstrated by initial testing and analysis, this approach could be beneficial in improving the overall parallel I/O system performance.

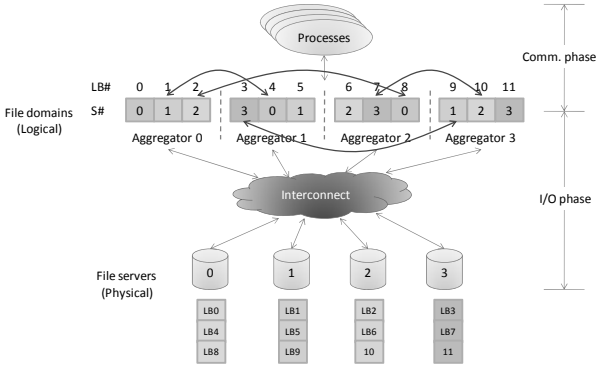



**Figure 3. Layout-aware optimization for collective I/O**

Figure 3 illustrates the basic idea of such a layout-aware optimization strategy for collective I/O. In this figure, we illustrate the details of the previous collective I/O operation example and the physical layout of requested data (distinguished by the logical position, i.e. LB#) on file servers. The file server number, i.e. S#, represents which file server the requested data reside on. Since the proposed layout-aware optimization strategy focuses on optimizing the I/O phase of the two-phase I/O strategy, we omit the details of communication phase here. The data layout strategy of file servers is assumed to be the most common round-robin mechanism. The proposed layout-aware optimization strategy rearranges the partitions of file domains and the requests of aggregators in a way that the requests are physically contiguous as much as possible, as shown in Figure 3. This example demonstrates that we rearrange requests of aggregators to have each aggregator accesses data on file servers contiguously, and multiple aggregators can access file servers concurrently. We assume that the data layout information can be obtained from the API provided by the underlying parallel file systems. It is not unusual that parallel file systems provide the interface to inquire the data layout on file servers, such as in PVFS2 [2][9]. Note that the rearrangement here is to change the requests that each aggregator carries out on behalf of the processes, or the way the aggregators access data. The rearrangement does not exchange data themselves among aggregators.

# 4. PRELIMINARY EXPERIMENTAL TESTING AND ANALYSIS

In this section, we present the results of several initial tests by manually bridging the information gap and providing the layout information and rearranging aggregators' accesses between parallel I/O middleware and parallel file systems. This method can be effective for us to measure the performance improvement of the proposed layout-aware optimization and analyze its potential. We first briefly describe the experimental environment and then present the preliminary testing and analytical results with two benchmarks.

## 4.1 Experimental Setup

Our experiments were conducted on a 65-node Sun Fire Linux-based cluster. This cluster is composed of one Sun Fire X4240 head node, with dual 2.7 GHz Opteron quad-core processors and 8GB memory, and 64 Sun Fire X2200 compute nodes with dual 2.3GHz Opteron quad-core processors and 8GB memory. The head node has 12 500GB 7.2K-RPM SATA-II drives configured as RAID-5 system. Each compute node has a 250GB 7.2K-RPM SATA hard drive. The experiments were tested on MPICH2-1.0.5p3 release and PVFS 2.8.1 file system.

## 4.2 Preliminary Results and Analysis

### *4.2.1 Synthetic Benchmark*

We have coded a synthetic benchmark in which each process does strided reads but the aggregated requests of all processes are sequential reads over the file. We have performed a series of tests on the Sun Fire cluster to compare the performance of rearranged layout-aware accesses and the original one. The total size of the data accessed by all processes are 128MB, 320MB, 640MB, 800MB and 4000MB respectively. The results are shown in Figure 4. It can be observed that the optimization strategy with layout awareness could have a considerable impact on the performance of parallel I/O system. The performance variation and the potential performance improvement could be up to 48%.

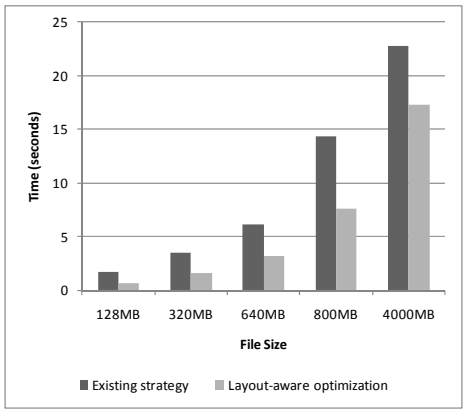
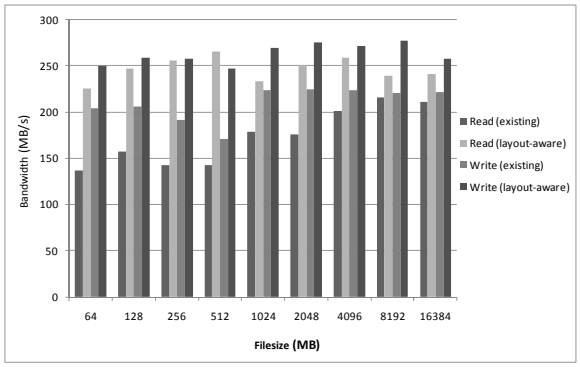
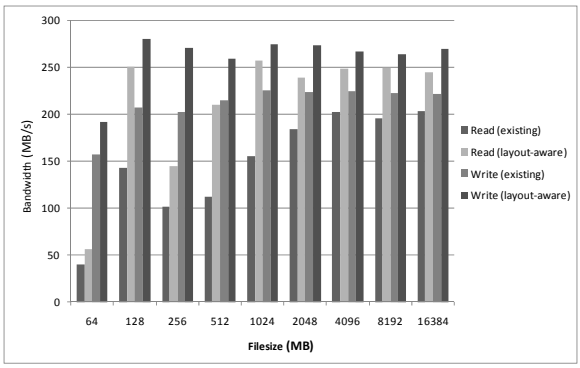



**Figure 4. Potential of layout-aware optimization with synthetic benchmark testing**

### *4.2.2 IOR Benchmark*




(a) Random accesses




(b) Interleaved accesses

**Figure 5. Potential of layout-aware optimization with IOR benchmark testing**

Figure 5 reports the testing results with IOR-2.10.2 benchmark from Lawrence Livermore National Laboratory [4]. We performed both interleaved reads/writes and random reads/writes tests, and varied the file size. As can be seen from these results, the layout-aware optimization strategy can affect the IOR benchmark testing performance considerably. The layout-aware strategy can potentially improve the I/O bandwidth up to 86%, 45%, 113% and 35% for random reads, random writes, sequential reads and sequential writes respectively. The average potential improvement of layout-aware strategy with different file sizes is 46%, 26%, 49% and 24% for random reads, random writes, sequential reads and sequential writes respectively.

## 5. CONCLUSION AND FUTURE WORK

Parallel I/O middleware and parallel file systems are fundamental and critical components for data-intensive and high-performance computing applications. While both of the technologies have made their success, little has been done to investigate a better integration of these two parallel I/O subsystems and to improve the overall performance. Collective I/O is one of the most important optimization strategies for parallel I/O middleware; however, with the separation of parallel I/O middleware and parallel file systems, the current collective I/O strategy may not produce the optimal performance improvement. In this study, we propose to reveal certain information between parallel I/O middleware and file systems to improve the overall I/O performance. Specially, we propose to incorporate layout information with collective I/O to improve the performance gain of collective I/O optimization. Although the current study and testing results are preliminary, they have demonstrated that layout-aware optimization can improve the collective I/O performance considerably. We will continue this research direction and carry out the prototype system implementation and evaluation. We are also actively working on modeling and dynamically choosing optimal data layout for HPC applications depending on the specific application features [13]. Recent works in a replicated storage system such as [5] demonstrate a great potential in improving I/O throughput and data durability and availability by utilizing idle storage. We plan to explore data layout optimizations in such replicated storage systems too.

## 6. REFERENCES


[1] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, M. Wingate, “PLFS: A Checkpoint Filesystem for Parallel Applications,” in Proc. of ACM/IEEE Supercomputing'09, 2009.

[2] P. H. Carns, W. B. Ligon 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, 2000.

[3] Y. Chen, S. Byna, X.-H. Sun, R. Thakur, and W. Gropp, "Hiding I/O Latency with Pre-execution Prefetching for Parallel Applications," in Proc. of the ACM/IEEE SuperComputing Conference (SC'08), 2008.

[4] Interleaved or Random (IOR) Benchmark, *http://sourceforge.net/projects/ior-sio/.*

[5] A. Gharaibeh and M. Ripeanu. Exploring Data Reliability Tradeoffs in Replicated Storage Systems. in Proc. of High Performance Distributed Computing (HPDC), 2009

[6] D. Kotz. Disk-directed I/O for MIMD Multiprocessors. ACM Transactions on Computer Systems, 15(1):41–74, 1997.

[7] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki and C. Jin, “Flexible IO and Integration for Scientific Codes Through the Adaptable IO System (ADIOS),” in Proc. of the 6th International Workshop on Challenges of Large Applications in Distributed Environments, 2008.

[8] J. May, “Parallel I/O for High Performance Computing,” Morgan Kaufmann Publishing, 2001.

[9] PVFS2 Development Team, “PVFS Developer's Guide,” *http://www.pvfs.org/cvs/pvfs-2-8-branch-docs/doc//pvfs2-guide.pdf.*

[10] I. Raicu, I. Foster, Y. Zhao, P. Little, C. Moretti, A. Chaudhary, D. Thain. "The Quest for Scalable Support of Data Intensive Workloads in Distributed Systems", in Proc. of ACM High Performance Distributed Computing (HPDC), 2009.

[11] J. del Rosario, R. Bordawekar, and A. Choudhary, "Improved Parallel I/O via a Two-Phase Run-time Access Strategy", in Proc. of the Workshop on I/O in Parallel Computer Systems at IPPS ’93, 1993.

[12] K. Seamons, Y. Chen, P. Jones, J. Jozwiak and M. Winslett, "Server-Directed Collective I/O in Panda", in Proc. of Supercomputing’95. ACM Press, 1995.

[13] X.-H. Sun, Y. Chen and Y. Yin, "Data Layout Optimization for Petascale File Systems," in Proc. of The 4th Petascale Data Storage Workshop, 2009.

[14] R. Thakur, W. Gropp and E. Lusk, “Data Sieving and Collective I/O in ROMIO,” in Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation, 1999.

[15] R. Thakur and A. Choudhary, "An Extended Two-Phase Method for Accessing Sections of Out-of-Core Arrays," Scientific Programming, (5)4:301-317, Winter 1996.

[16] Top 500 Supercomputing Website. *http://www.top500.org.*