Cluster Comput
DOI 10.1007/s10586-012-0200-4

Cost-intelligent application-specific data layout optimization for parallel file systems

Huaiming Song · Yanlong Yin · Yong Chen · Xian-He Sun

Received: 12 September 2011 / Accepted: 23 January 2012

Abstract Parallel file systems have been developed in recent years to ease the I/O bottleneck of high-end computing system. These advanced file systems offer several data layout strategies in order to meet the performance goals of specific I/O workloads. However, while a layout policy may perform well on some I/O workload, it may not perform as well for another. Peak I/O performance is rarely achieved due to the complex data access patterns. Data access is application dependent. In this study, a cost-intelligent data access strategy based on the application-specific optimization principle is proposed. This strategy improves the I/O performance of parallel file systems. We first present examples to illustrate the difference of performance under different data layouts. By developing a cost model which estimates the completion time of data accesses in various data layouts, the layout can better match the application. Static layout optimization can be used for applications with dominant data access patterns, and dynamic layout selection with hybrid replications can be used for applications with complex I/O patterns. Theoretical analysis and experimental testing have been conducted to verify the proposed cost-intelligent layout approach. Analytical and experimental results show that the proposed cost model is effective and the application-specific data layout approach can provide up to a 74% performance improvement for data-intensive applications.

Keywords Data layout · I/O performance modeling · Parallel file systems · Parallel I/O · Data-intensive computing

H. Song (✉)
R&D Center, Dawning Information Industry Co., Ltd., Beijing 100084, China
e-mail: songhm@sugon.com

Y. Yin · X.-H. Sun
Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA

Y. Yin
e-mail: yyin2@iit.edu

X.-H. Sun
e-mail: sun@iit.edu

Y. Chen
Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA
e-mail: yong.chen@ttu.edu

1 Introduction

Scientific and commercial applications, such as nanotechnology, astrophysics, climate, and high energy physics, are increasingly reliant on large datasets. In most data-intensive applications, the storage devices are the critical bottleneck and the situation worsens with increased data volumes. While the computation capabilities of processors have been increasing rapidly during the past decades, disk capacities have not seen the same growth. Therefore, there exists an enormous performance gap between processors and disks. Parallel file systems, such as Lustre [1], GPFS [2], PanFS [3], and PVFS2 [4] are designed to mask the ever-increasing gap between computing and I/O performance, by combining large numbers of storage devices and providing high degree of I/O parallelism. There are three commonly used data layouts in existing parallel file systems: *1-DH*, *1-DV*, and *2-D*. 1-DH is short for one dimensional horizontal layout, which refers to the layout strategy that data accessed by each I/O client is distributed across all file servers. 1-DV is short for one dimensional vertical, which means each I/O client accesses data from one file server. 2-D layout refers to the policy that each I/O client accesses data from a

Published online: 15 February 2012

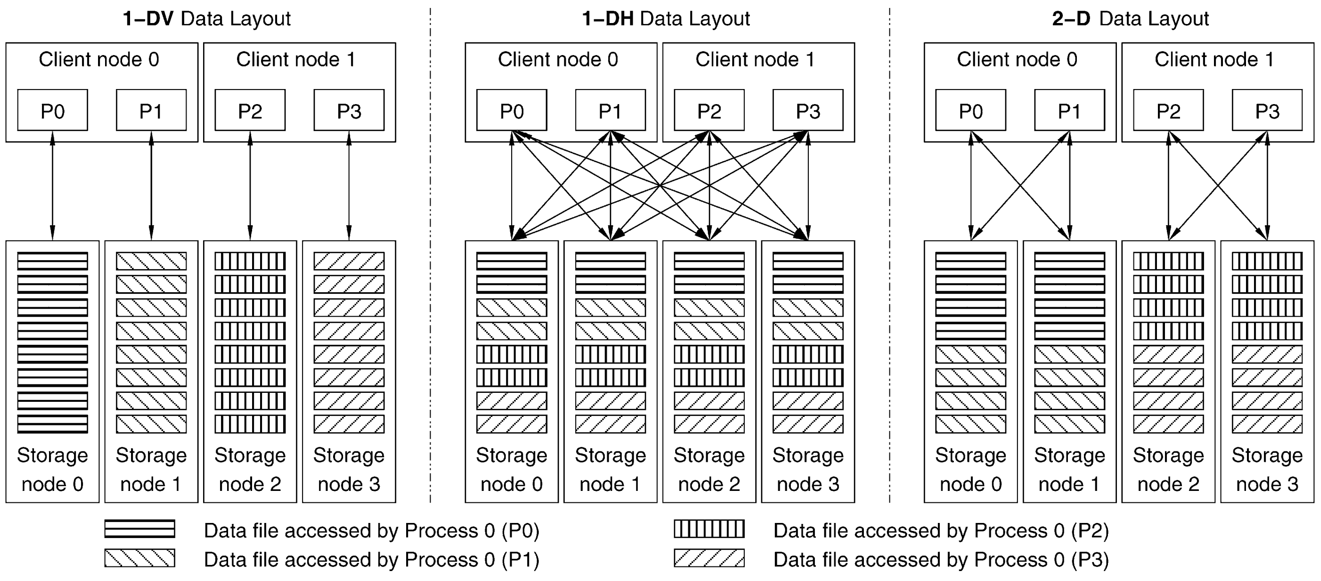

1−DV Data Layout
1−DH Data Layout
2−D Data Layout
Client node 0
Client node 1
P0
P1
P2
P3
Storage node 0
Storage node 1
Storage node 2
Storage node 3
Data file accessed by Process 0 (P0)
Data file accessed by Process 0 (P1)
Data file accessed by Process 0 (P2)
Data file accessed by Process 0 (P3)

Fig. 1 Three commonly used data layout strategies

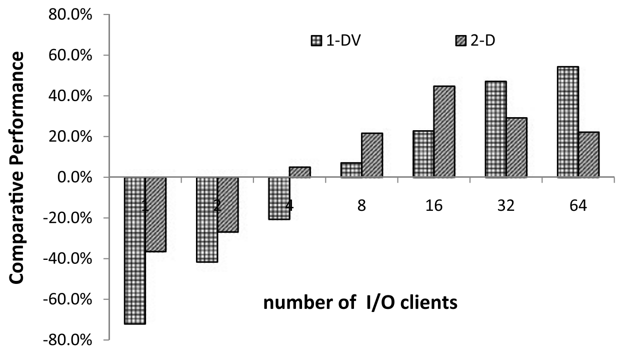
subset of all file servers. These policies are demonstrated in Fig. 1, and they are designed for different I/O workloads.

For a workload, I/O behaviors are dependent on application characteristics and vary considerably from application to application in terms of request frequency, data size, access concurrency, and data continuity. Even in one application, data access pattern may vary from time to time. Therefore, a complex application may not have a single dominant data access pattern, but rather consist of many patterns during different phases. For example, the request size may be very large at one time while small at another time, and the number of concurrent I/O requests might change often.

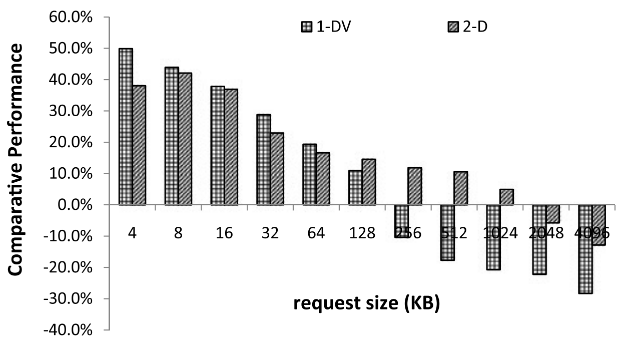
Generally, a large I/O request should invoke parallel access to many file servers to increase throughput, while a small request should be processed on a single file server to reduce latency. Therefore, data access patterns and data layout manners influence the performance of parallel I/O significantly. Nevertheless, there is no single data layout policy working well for all workloads. A data layout policy that works well for one type of I/O workload may be a bad choice for another. Moreover, in existing parallel file systems, the use of these data layouts is hectic and rare due to several limitations. To identify an appropriate data layout option, the user needs to understand the I/O workload of his/her application, and needs to understand the underlying parallel file systems. Even if the user is an expert on both sides, the application may have data access patterns that do not have a perfect match with the options provided by the underlying file system. More research efforts are needed to explore an intelligent layout selection strategy which provides the full potential of parallel file systems. Finding an appropriate data layout for a given application has significant and practical importance for HPC applications, especially for applications that are data-intensive.

Understanding this interaction between access pattern and data layout in parallel file systems is critical to optimizing I/O performance. Figure 2 shows an example of I/O performance under three typical data layouts. As the I/O performance varied a lot for different access patterns, to make the results clear, we regarded the performance of 1-DH as the baseline, and compared the performance of 1-DV and 2-D with the baseline for each case. In subfigure (a), a fixed request size with an increasing number of concurrent I/O clients is described, while in subfigure (b) a fixed number of I/O clients with an increasing request size is described. From the results of subfigure (a) it can be observed that when the concurrency is small, 1-DH has the highest I/O performance; however, when the number of concurrent processes increases, 2-D or 1-DV have the highest bandwidth. Subfigure (b) shows that the request size is also influential on I/O performance under different data layouts. The results indicate that different access patterns require different layouts in order to optimize I/O performance. While data layout and user request patterns seriously affect the I/O performance of data-intensive applications, current I/O performance optimization strategies are not designed to capture data access pattern as an application-specific feature. This is an inherited limitation of existing approaches, and we address this limitation well in this study. We propose an innovative cost-intelligent application-specific data layout approach, in which a cost model is developed to guide the data layout selection. An optimal data layout is determined automatically for a given application.

In this paper, we propose a cost-intelligent data access strategy to integrate the data access optimization with layout



(a) Fixed request size (1024 KB)



(b) Fixed number of I/O clients

Fig. 2 I/O Performance comparison for different access patterns of three layout policies: 1-DH, 1-DV, and 2-D. As the I/O performance varied a lot for different access patterns, to make the results clear, we regarded the performance of 1-DH as the baseline, and compared the performance of 1-DV and 2-D with the baseline for each case. The results were collected from the IOR benchmark on a PVFS2 system with 4 file servers and 8 computing nodes with sequential I/O workloads

optimization techniques, which is beneficial to various types of I/O workloads. This study makes the following contributions. (1) We propose a cost model of data access for parallel file systems, which can be used for estimating the completion time of data accesses with different data layout policies. (2) We present a static layout optimization approach, to identify the optimal data layout for applications with dominant access patterns. (3) We also propose a dynamic data access strategy with hybrid replication for applications with mixed I/O workloads where one layout policy does not benefit all data accesses. The analytical and experimental results show that the newly proposed cost-intelligent application-specific data layout approach is very promising and has real potential in unleashing the full potential of parallel file systems.

The rest of this paper is organized as follows. Section 2 briefly reviews the related work in I/O optimization technologies for data-intensive and high-performance computing systems. In Sect. 3, we present a cost model of data access under different layout policies in parallel file systems. Section 4 proposes a layout selection strategy for a given application based on the analysis of overall data access cost. Section 5 describes a dynamic access strategy with hybrid data replications. Experimental and analytical results are presented in Sect. 6. Section 7 discusses the applicable spheres and potential further improvements of the cost-intelligent data access scheme. Section 8 concludes the paper.

2 Related work

Parallel file systems are widely used to exploit high degree of I/O parallelism in data-intensive and high-performance computing applications. However, poor data access performance has been recognized as the system bottleneck. Numerous research efforts have been devoted to improving I/O performance through data access optimization and data organization techniques.

2.1 Data access optimization

In order to optimize data access for parallel I/O, a collection of advanced techniques, such as request arrangement, caching and prefetching, have been proposed and developed. These techniques are successful in reducing the overhead on network or I/O servers during data accesses, and they are usually implemented in parallel I/O libraries layer or file server layer.

Request arrangement techniques, including data sieving [5], two-phase I/O [6], collective I/O [5, 7, 8], Datatype I/O [9] and List I/O [10], mainly focus on merging small and non-contiguous data accesses into large and contiguous requests, to reduce the I/O completion time. Data sieving [5] arranges requests to access a single large contiguous chunk containing small pieces of data from the first request to the last request instead of accessing each portion separately. Two-phase I/O [6] and collective I/O [5, 7, 8] in MPI-IO libraries ameliorate the performance penalties incurred when directly mapping the distribution of data on file servers to the distribution in processor memories, as a moderate number of concurrent accesses to an I/O server often get better performance. Both Datatype I/O [9] and List I/O [10] provide ways for richer semantics of various non-contiguous or structured data access patterns, to reduce the amount of I/O requests and the overhead in network transmission.

Caching and prefetching can optimize the I/O performance significantly because of locality and regularity of data accesses. In parallel I/O systems, caching techniques usually aim at storing data at a client side buffer in a collective way [11–14], so that all I/O client processes can share data in their memories among multiple nodes through the network. Data prefetching techniques aim at fetching data in advance, and can be roughly classified into two categories: informed and speculative. Informed data prefetching [15–18] obtains data access patterns before data accessing, usually based on I/O trace analysis, profiling or hints. While speculative methods usually prefetch data aggressively based on runtime analysis, and are more suitable for data accesses without explicit patterns [19–22].

2.2 Data organization

Research efforts on data organization mainly focus on physical data layout optimization [23–29] among multiple file server nodes according to the I/O workloads of applications. Since I/O requests usually fall into several patterns in parallel applications, it is possible to re-organize the data layout in the storage nodes to reduce disk head movement [30–32, 37, 38], thus improving the overall I/O performance. Data partition [23, 24] and replication [26, 27, 33] techniques are also commonly used to reduce disk head movements or to increase I/O parallelism. For example, Zhang et al. [33] proposed a data replication scheme to distribute I/O workloads on multiple replicas to improve the performance, so that each I/O node only serves requests from one or a limited number of processes. Most parallel file systems, such as Lustre [1], GPFS [2], PanFS [3], and PVFS2 [4], provide several data layout policies. A large number of data layout optimization techniques are based on transcendental I/O workload information, such as trace or profile analysis [23, 24], to guide data partitioning across multiple disks or storage nodes.

In parallel file systems, data is distributed among multiple storage servers to achieve a high degree of I/O parallelism. Numerous research efforts have been devoted to data access performance and cost analysis in parallel file systems. Welch et al. [3] provided an analysis of the performance for parallel file systems from several aspects, including I/O performance, system recovery performance, and metadata operation performance. Several other research work [34–36] analyzed how file system performance can be affected by many factors of workload such as the distribution of files, I/O request sizes, and I/O access characteristics.

The proposed approach is different from existing work. A new cost model to guide data layout optimization for parallel I/O systems is developed. In a previous work [29], analysis of the completion time of data access under different data layouts was determined. The cost model in this paper is an extension of the previous cost model. When an application has dominant I/O patterns, it chooses an optimal data layout for that application based on the overall access cost analysis. When an application has mixed I/O workloads without dominant patterns, a dynamic data access strategy with hybrid data replications is proposed, which can automatically perform I/O on one replica with the lowest cost for each data access.

3 Cost analysis model

In this paper, the three typical data layout policies are considered: 1-DH, 1-DV, and 2-D. These layout policies are supported by most parallel file systems, e.g. Lustre [1] and PVFS2 [4]. What needs to be emphasized is that the data layout here is process-oriented, rather than file-oriented. That is to say, 1-DH layout means one client process accesses data from all file servers, 1-DV means one client process accesses data from one file server, and 2-D means one client process accesses data from a subset of all file servers.

Different layout policies in parallel I/O systems lead to different interactive behaviors, and thus introduce different data access costs. In order to analyze data access cost in parallel file systems, the interactions between I/O clients and file servers must be understood. The cost is defined as the completion time of each data access, and it mainly consists of two parts: the time spent on the network transmission (denoted as $T_{network}$) and the time spent on the local I/O operations of storage nodes (denoted as $T_{storage}$). Generally, the time spent on network, $T_{network}$, consists of T_e and T_x. The former is the time spent on establishing the network connection and preparing for data transmission, and the latter is the time spent on transferring the data. The storage cost $T_{storage}$ consists of T_s, the startup time, and T_{rw}, the time spent on actual data read/write. Thus the data access cost can be represented as follows.

$$\begin{aligned} T &= T_{network} + T_{storage} \\ &= (T_e + T_x) + (T_s + T_{rw}) \end{aligned} \tag{1}$$

Network establish time and storage node startup time are independent from data size, while the transmission time and read/write time are proportional to the data size. To make the model simple and effective, it is assumed that there are no overlaps between I/O clients and file servers. This implies that every data access involves network transmission. Also only contiguous I/O requests are considered, and non-contiguous requests, e.g. list I/O, could be regarded as a set of contiguous data accesses. Table 1 lists all the parameters

Table 1 Parameters in cost analysis model

Parameters	Description
p	Number of I/O client processes.
n	Number of storage nodes (file servers).
m	Number of processes on one I/O client node.
s	Data size of one access.
e	Cost of single network connection establishing.
v	Network transmission cost of one unit of data.
α	Start up time of one disk I/O operation.
β	Cost of reading/writing one unit of data.
g	Number of storage groups in 2-D layout.

Table 2 Cost formulas for three layout policies

Layout Type	Condition		Network Cost $T_{network}$		Storage Cost $T_{storage}$	
			Establish T_e	Transmission T_x	Startup T_s	I/O T_{rw}
1-DV	$p \le n$		me	msv	$\lceil \frac{p}{n} \rceil \alpha$	$\lceil \frac{p}{n} \rceil s\beta$
	$p > n$	$m \le \lceil \frac{p}{n} \rceil$	$\lceil \frac{p}{n} \rceil e$	$\lceil \frac{p}{n} \rceil sv$		
		$m > \lceil \frac{p}{n} \rceil$	me	msv		
1-DH	$m \le \lceil \frac{p}{n} \rceil$		pe	$\frac{psv}{n}$	$p\alpha$	$\frac{ps\beta}{n}$
	$m > \lceil \frac{p}{n} \rceil$		mne	msv		
2-D	$p \le g$		$m\lceil \frac{n}{g} \rceil e$	msv	$\lceil \frac{p}{g} \rceil \alpha$	$\frac{\lceil \frac{p}{g} \rceil s\beta}{\lfloor \frac{n}{g} \rfloor}$
	$p > g$	$m \le \frac{\lceil \frac{p}{g} \rceil}{\lfloor \frac{n}{g} \rfloor}$	$\lceil \frac{p}{g} \rceil e$	$\frac{\lceil \frac{p}{g} \rceil sv}{\lfloor \frac{n}{g} \rfloor}$		
		$m > \frac{\lceil \frac{p}{g} \rceil}{\lfloor \frac{n}{g} \rfloor}$	$m\lceil \frac{p}{g} \rceil e$	msv		

considered in this model. The complete cost formulas covering all situations are listed in Table 2. The detailed analysis of data access cost can be found in literature [29].

The cost formulas shown in Table 2 provide a detailed analysis of completion time for data accesses under different data layouts. From the cost model, it is easy to calculate which layout gets the lowest completion time for a given data access pattern. Although there are several variables among the model parameters, for most applications, the runtime variables such as m, p and n are fixed for each run. In general, for a given system, e, v, α and β can be regarded as constants. The network transmission time and read/write time on storage nodes are proportional to data size, while I/O startup time with sequential data accesses is smaller than that of random data read/write.

4 Layout optimization based on overall I/O cost

The proposed cost model could be used to estimate data access time for each I/O request under different data layouts in parallel file systems. However, as mentioned in Sect. 1, one application may have different kinds of data accesses. Because different I/O requests require different layouts for optimal I/O performance, and it is not easy to identify which layout is best for the application. A layout optimization strategy based on the overall data access cost is proposed. Since calculating I/O cost for each request by the model is possible, given that there is a prior knowledge of data accesses for the application, it is not difficult to calculate the overall cost, by summing the cost of all I/O requests. A lot of data-intensive applications have regular data access patterns, and the I/O behavior can be learned from previous runs. For example, numerous tools were developed to trace I/O requests for applications [18, 39]. By calculating the I/O cost for each data access and summing them together, the overall cost for all data accesses in different layout policies is found. The following formula shows how to calculate the overall data access cost for an application.

$$\begin{aligned} TC &= \sum_{i=1}^{n} T_i \\ &= \sum_{i=1}^{n} ((T_{ei} + T_{xi}) + (T_{si} + T_{rwi})) \end{aligned}$$

By comparing these overall costs under three data layouts, it is possible to determine the optimal data layout which leads to the lowest cost for that application.

$$TC_{optimal} = MIN\{TC_{1-DH}, TC_{1-DV}, TC_{\text{2-D}}\}$$

Figure 3 shows the procedure for a layout optimization strategy based on overall data access analysis. The procedure consists of two steps. The first step is to estimate system parameters of a given parallel file system. In this implementation, a pre-estimation phase was used to calculate the value of these parameters. We use one file server in the parallel file system to test α and β, and use a pair of nodes (one I/O client and one file server) to estimate network parameters, e and v. By measuring the parameters with different request sizes and repeating each case with multiple runs(the number is configurable), the averages can be obtained and substituted for the parameter values. The next step is to compute the overall access cost with the parameters obtained from the first step. In this step, I/O traces are used as input to sum the access cost for every I/O request one by one for the three data layouts. As mentioned above, the data layout that has

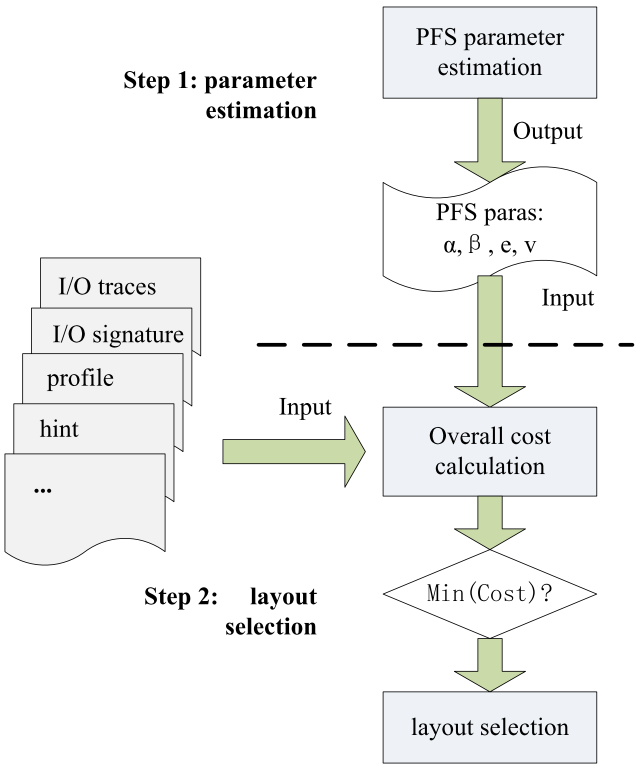


Fig. 3 Layout optimization based on overall access cost

the minimum access cost is the optimal data organization for that application.

Data-intensive applications usually involve a large number of I/O requests with different request size and concurrency, and there is no single layout that benefits all I/O requests. The aforementioned overall cost analysis simplifies layout selection for these applications. It is important to point out that three layout policies (1-DV, 1-DH, and 2-D) are defined based on the distribution of data accesses by each process. In cases where each I/O client process reads/writes an independent file, it is natural to place different files with different layout policies. However, in cases where all processes access a shared file, different data layout policies can be defined in terms of different stripe sizes: a large stripe size that lets each process read/write data from a single file server, which is similar to 1-DV layout manner; a small stripe size that lets each I/O process be served by all file servers in parallel, which is similar to 1-DH data layout; and a medium stripe size can be regarded as a 2-D layout manner, where each data access only involves a subset of all storage nodes. Therefore, application features and specific I/O scenarios should be considered when applying this layout optimization approach.

5 Dynamic access with hybrid data replication

5.1 Hybrid data replication strategy

The overall access cost analysis is practical for data layout optimization in data-intensive applications, especially those with dominant I/O patterns. For example, if an application has a large number of concurrent I/O processes, 1-DV layout is the best; where 1-DH layout is more applicable for the cases with less number of concurrent I/O processes and large requests. However, in some applications, there are mixed I/O workloads with various data access patterns, and there is no dominant patterns. Moreover, the difference of the overall access costs with different layouts might be insignificant. It is not easy to determine which policy is the best as the overall access cost analysis is not applicable. For example, during one run of an application, there might be a small number of concurrent I/O requests at one moment, or a burst of I/O requests at another moment. In addition, the request size could be large or small at different moments. This variation makes it impossible for a static data layout policy to serve all data accesses in the most efficient way. Therefore, a dynamic access strategy with 'hybrid' data replication is proposed to optimize layout for the above application scenarios.

Hybrid data replication means that each file has multiple copies with different data layout policies in the parallel file system, i.e. one replica in 1-DH layout, another in 1-DV layout, and the third replica in 2-D layout. For each data access, we first calculate the costs under these replicas, and then select one replica with the lowest access cost. Therefore, the dynamic data access strategy is a fine-grained optimization, which improves I/O performance with layout selection for every data access: different accesses could be mapped to different data copies independently, and all accesses could be optimally improved. Since each data access is assigned to the best-fit replica, the dynamic data access strategy with hybrid data replication can serve all kinds of I/O workloads with high performance. It significantly improves I/O performance for data-intensive applications.

Figure 4 shows an example of the hybrid replication strategy. In this example, there are four files stored in four storage nodes. Each file has three replications in 1-DV, 1-DH, and 2-D data layout respectively. Each data access will be directed to one replica with the lowest access cost, to improve the performance for every access, and thus improve overall performance. Generally, data read is simple, but data write is more complicated because the data must be consistent among the multiple copies. There are a couple of possible solutions to handle write operations. In this design and implementation, the data is first written to one replica with the lowest cost, and lazy strategies [40] are applied to synchronize data to the other replicas. Hence, for write operations, only data access cost on the optimal replica is counted,

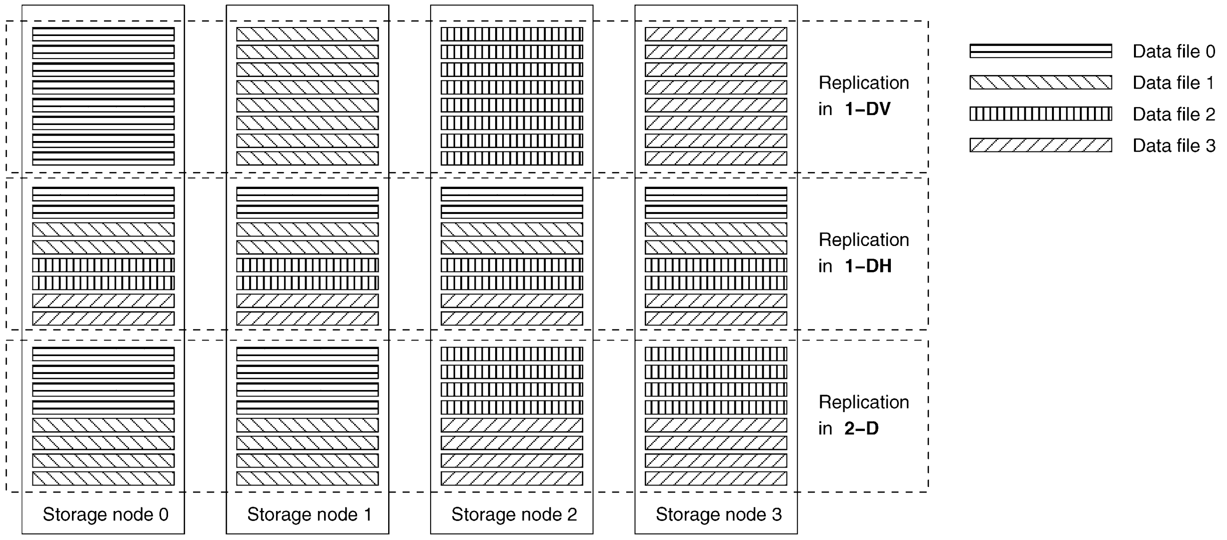


Fig. 4 The hybrid data replication scheme

and the cost of lazy data synchronization is considered as a background operation.

Similarly, application features must be considered for the hybrid data replication approach. When each I/O process accesses an independent file, it is easy to adopt different layout policies for different data replicas. However, when all I/O clients access a shared file, different layout can be defined in terms of different stripe size. As mentioned in the previous section, a large stripe size that lets each I/O process be served by one I/O server is similar to the 1-DV layout. A small stripe size that lets each I/O process be served by all I/O servers in parallel is similar to the 1-DH layout. A medium stripe size is similar to the 2-D layout.

A prototype of these cost calculations and layout selection procedures is implemented in the MPI-IO libraries. The hybrid data replication is transparent to the users. The users can run their programs without any modifications. This prototype can perform cost calculation and layout selection automatically. Below are some modifications to the MPI-IO libraries.

File open: for each file open function, all the copies are opened.

File read: for each data read function, calculate access costs for all data copies, and then select one copy with the lowest cost to read data.

File write: for each data write, synchronize related data blocks, and then perform I/O on one copy with the lowest cost to perform write. Then add the write requests of other copies to a lazy synchronization queue.

File close: synchronize data for all copies and close all of them.

For data writes, all lazy write requests are put to a request queue right after writing to the optimal replica. A dedicated data synchronization thread is implemented in the client side library to perform these lazy write requests in the queue. Because data synchronization is a background operation, each data write function can return right after putting the lazy write request into the queue.

Admittedly, the hybrid data replication strategy needs more storage space. This is a trade-off between data access performance and storage capacity. With the ever-increasing disk capacities and ever-increasing performance gap between CPU and disk, the trade-off should be increasingly important for some performance-crucial and data-intensive applications. This hybrid strategy provides a good alternative to existing strategies.

6 Experimental evaluation

6.1 Experimental platform

The experiments were conducted on a 65-node Sun Fire Linux-based cluster, including one head node and 64 computing nodes. The head node Sun Fire X4240 is equipped with dual 2.7 GHz Opteron quad-core processors, 8 GB memory, and 12 500 GB 7200 RPM SATA-II drives configured as RAID5 disk array. The hardware and software configuration of the computing nodes are shown in Table 3. In the Ethernet environment, all 64 computing nodes were employed, of which 16 nodes work as file servers and the other 48 nodes work as I/O client nodes. In the InfiniBand testing, PVFS2 was configured with 8 file servers, and the rest 8 nodes served as I/O clients. The head node is used for management, and there was no overlap between file servers and I/O client nodes in either environment.

Table 3 Node information of experiment platform

CPU	2.3 GH Quad-Core AMD Opteron ∗ 2
Memory	8 GB
Storage	SATA II 250 GB, 7200 RPM
Network	Gigabit Ethernet (additional 4X IB for 16 nodes)
OS	Ubuntu 4.3.3-5, Linux kernel 2.6.28.10
PVFS2	PVFS2 version 2.8.1

Table 4 Parameter values of the experimental platform

Parameters	Ethernet	InfiniBand
e	0.0003 sec	0.0002 sec
v	$\frac{1}{120\text{ MB}}$	$\frac{1}{1000\text{ MB}}$
α	0.0003 sec (rand), 0.0001 sec (sequent)	
β	$\frac{1}{120\text{ MB}}$	

Table 5 Statistics of model estimation with different interconnections

Interconnection	Access Type	Total Case Count	Correct Count	Correct Rate
Ethernet	Random	4200	3821	90.98%
	Sequential	4200	3702	88.14%
InfiniBand	Random	2200	1867	84.86%
	Sequential	2200	1763	80.14%

The experiments consisted of three parts. The first part was to verify the data access cost model, including parameter estimation and model accuracy verification. The second part was to use the I/O cost formulas to optimize the data layout based on overall access cost. The third part was to verify the efficiency of the hybrid data replication strategy. The widely-used parallel file system benchmark IOR and MPI-TILE-IO to test the I/O performance were used. IOR is a benchmark program used to test random and sequential I/O performance of parallel file systems. MPI-TILE-IO is a benchmark that tests the performance of MPI-IO for non-contiguous access workload. PVFS2 can work with both MPI-IO and POSIX interfaces, and the former was tested in the experiments. Unless otherwise specified, in all our testing, 1-DV means that each file was placed in one storage node; 1-DH means each file was striped across all storage nodes; and 2-D means each file was striped on 2 storage nodes.

6.2 Model verification

First we conducted experiments to get the approximations of e, v, α, and β of the cost model in our experimental platform. In order to get disk startup time and read/write rate, we employed one storage node to test α and β. We also employed a pair of nodes to estimate network parameters, e and v, in both Ethernet and InfiniBand environments. We performed experiments with different request sizes and repeat these tests thousands of times for both random and sequential I/O patterns. We got the parameter values by calculating the average values. For disk startup time, we measured different startup times on storage nodes for sequential and random data accesses, respectively. The values of the parameters are listed in Table 4.

Next, we conducted experiments to verify the model's ability to select the layout policy with the lowest data access cost. In this set of experiments, we ran a large number of cases, and all data access patterns were tested in three data layout policies with different request sizes and process numbers. Table 5 shows the statistical results of the model accuracy. We used the cost model to estimate the I/O performance, to choose the layout policy with the lowest data access cost. We compared this chosen layout policy with actual test results. If the chosen data layout actually produced the best performance, the estimation was marked as 'Correct'. As can be seen from Table 5, the cost model performs well: the correct rate is around 80% to 85% in the InfiniBand environment, and even higher with the Ethernet interconnection. The results indicate that the cost model can select the optimal layout policy with the highest I/O performance for most data access patterns. The cost model is effective to identify the best layout policy for data-intensive applications.

Figure 5 shows the distribution of which layout policies can get the best I/O performance for different data access patterns. The results were collected in both Ethernet and InfiniBand environments. In this set of experiments, we tested sequential data access patterns, and varied the request size and the number of concurrent I/O processes. For each data access pattern, we measured the I/O bandwidth under three data layout policies, and then compared their performances. We only plotted the layout policy with the highest I/O bandwidth for each access pattern. We also estimated the highest performance via the proposed cost model and marked with different colors. From the figure we can observe that in most cases the highest performance through measurement is the same as the estimation of the cost model. The results show that the accuracy of the cost model is very high. We can also observe that when the request size is very small, the 1-DV layout policy can get the highest I/O bandwidth, which is because the network communication overhead dominated the performance. When the number of concurrent I/O processes is very high, the 1-DV can get the best performance, which is because the 1-DV layout can reduce the contention in storage nodes. When the number of concurrent processes is small and the request size is large, the 1-DH can get the best bandwidth, as data access can benefit from parallel processing and large data block read/write. When the number of concurrent I/O processes is medium and the request size

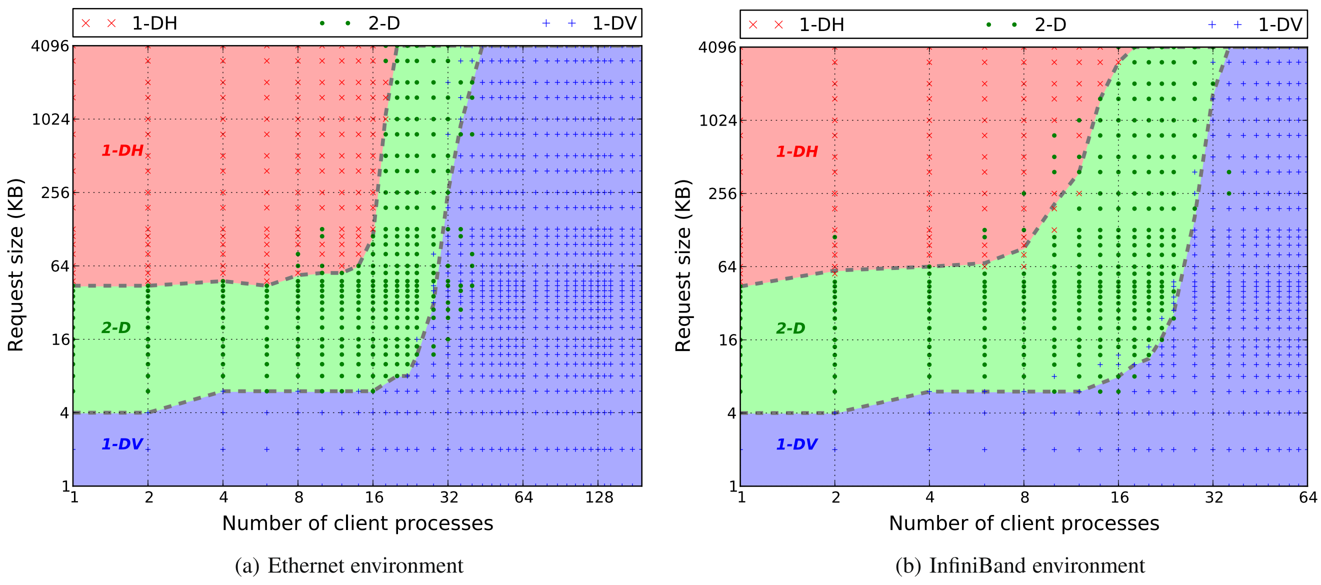


Fig. 5 I/O performance comparison among the three data layout policies. In this set of experiments, we adopted sequential data access patterns. We varied the number of MPI processes and the request size. For each data access pattern, we tested the I/O bandwidth for all three layout policies and then compared their performances. We plotted the layout policy with the actual highest I/O bandwidth for each access pattern. We also use different colors to represent the best performance estimated by the proposed cost model

is not too small, the 2-D layout can get the best I/O performance. The results confirmed the results of previous analysis of our cost model.

From all the experimental results shown above, we can conclude that, the proposed model can effectively predict the access cost for all data access patterns. Although there is a small deviation in performance between the actual testing and the model estimation, the accuracy of selecting the layout policy with the best performance is as high as $80\% \sim 91\%$ among the three layout policies. Therefore, the proposed cost model can make accurate estimation on the performance of complex I/O workloads.

6.3 Layout optimization according to the overall access cost

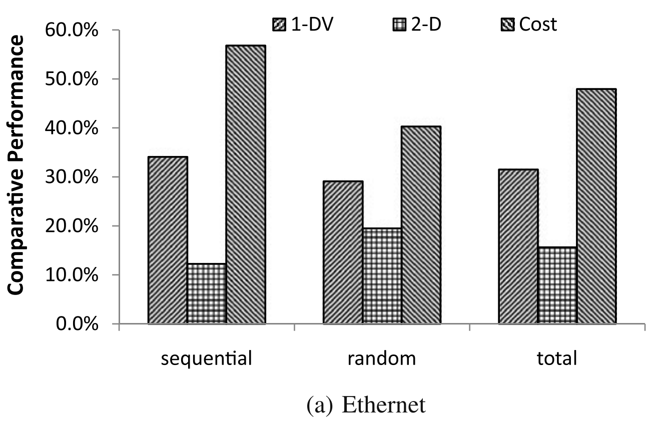
For complex applications with mixed I/O workloads, we propose an overall access cost analysis approach to select the best data layout. We conducted experiments to evaluate layout optimization based on overall cost analysis. We designed 10 different application scenarios with mixed I/O workloads of IOR and MPI-TILE-IO benchmarks. Each scenario consisted of 6 to 10 IOR or MPI-TILE-IO instances with different access patterns by modifying the runtime parameters, e.g. request size and the number of concurrent I/O processes. We ran these instances one by one to simulate different access patterns at different times for each scenario. We measured the average I/O bandwidth for both sequential and random I/O workloads in three different layouts respectively. We also measured the I/O bandwidth in optimized layout by overall cost analysis for all scenarios.

Figure 6 shows the comparison results of I/O performance for different data layouts. We used the I/O bandwidth of 1-DH as the baseline, and compared the bandwidth achieved in other layouts to it. From the results we can observe that the proposed layout optimization based on overall cost analysis can achieve up to 57% performance improvement in the Ethernet environment and up to 35% improvement in the InfiniBand Environment. Therefore, the proposed overall cost analysis approach can significantly improve I/O performance for data-intensive applications. The improvement of I/O performance with an Ethernet interconnection is higher than that with an InfiniBand connection. The reason may be that the accuracy of the proposed cost model is higher in an Ethernet environment, as shown in Table 5.

6.4 Dynamic data access with hybrid replications

We designed experiments to measure the efficiency of dynamic data access with hybrid replications. We ran two sets of mixed workloads. For each of them, we ran a set of IOR or MPI-TILE-IO instances one by one with different runtime parameters, to simulate different data access patterns at different moments.

In IOR tests, we varied the process number and request size in different data layout policies. The process numbers were 1, 2, 4, 8, 16, 32, 48, 96 in the Ethernet tests and 1, 2, 4, 8, 16, 32, 64 in the InfiniBand tests, respectively. The request sizes were 4 KB, 8 KB, 16 KB, 32 KB, 64 KB,



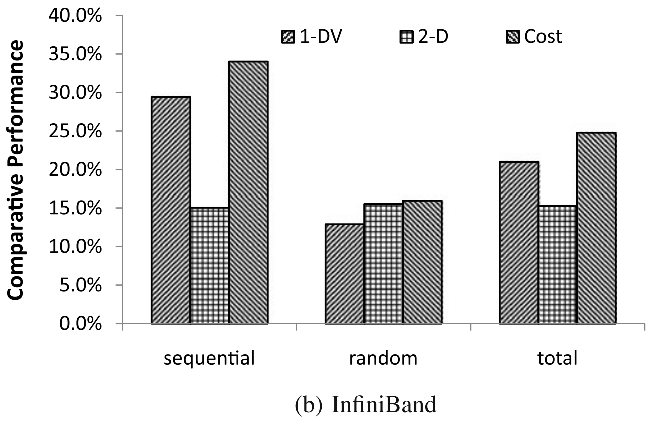
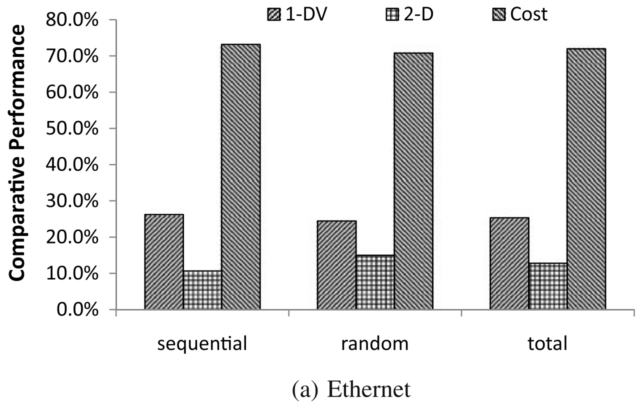


Fig. 6 Performance improvement of overall cost analysis approach (mixed IOR and MPI-TILE-IO workloads). We use the performance with 1-DH layout as baseline, and compare the performance of other layout manners with it. Label 'Cost' refers to the optimized layout according to overall access cost

128 KB, 256 KB, 512 KB, 1 MB, 2 MB, and 4 MB, respectively. We configured one independent file with size of 64 MB for each process. We measured the performance of random and sequential access patterns in 1-DV, 1-DH, and 2-D layout policies. We also compared them with the cost-intelligent dynamic replication selection strategy. We set these layout policies to different directories in PVFS2 system. Since in each test the processes first wrote the files and then read them back, the cost-intelligent policy first chose which directory to access based on the cost estimation, and then wrote/read files in that directory. Figure 7 shows the results of the IOR workloads, where the vertical axis represents the comparative performance improvement with 1-DH layout policy. Here the performance is represented by the average I/O bandwidth, which is calculated by the total data size divided by the total running time. As shown in Fig. 7, the proposed cost-intelligent layout policy can get the best performance with both Ethernet and InfiniBand interconnections. The performance improvement is around $20 \sim 74\%$ compared with the other three data layout policies. Similar to the results in Fig. 6, the improvement with the Ethernet interconnection is higher than that with the InfiniBand connection.

In MPI-TILE-IO tests, we varied the tile size (corresponding to the request size) and the number of I/O processes. The tile sizes were $1024 * 1024$ Bytes, $1024 * 2048$ Bytes, $2048 * 2048$ Bytes, and the numbers of I/O process were 4, 16, 64, 128, and 256, respectively. Since MPI-TILE-IO benchmark reads only one shared file, we simply striped the file across all file servers, and made 3 copies with different stripe sizes for different replicas. The stripe sizes were 16 KB, 256 KB, and 4 MB, respectively. We measured the performance of collective I/O and non-collective I/O on each replica respectively, and then compared them with cost-intelligent policy. Here the cost-intelligent policy chooses one replica to read based on the data access cost



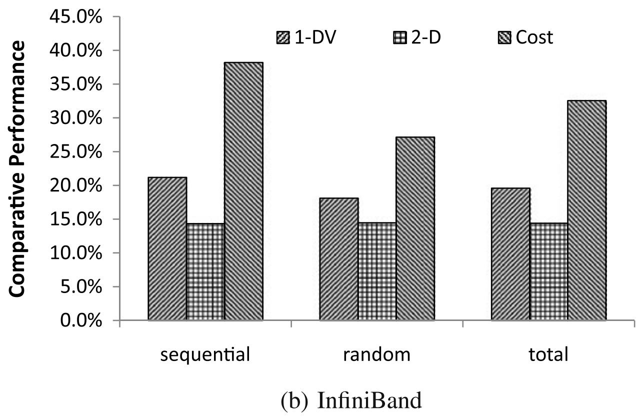
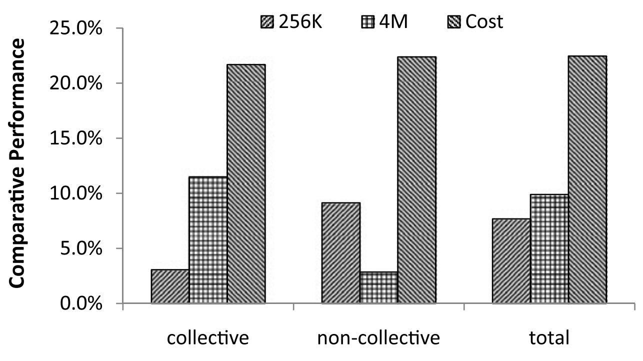
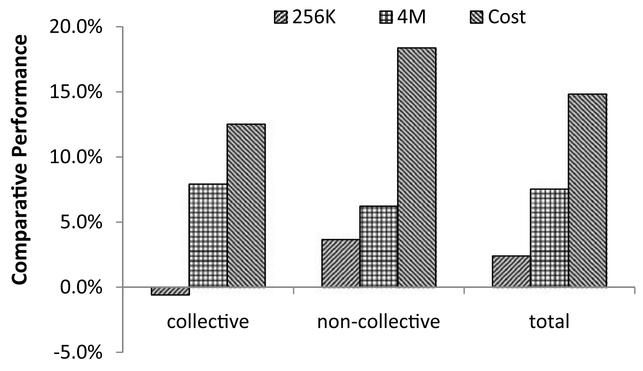


Fig. 7 Performance improvement compared to 1-DH (IOR benchmark). For each case, we compare the performance of other layout manners with 1-DH layout manner. The performance improvements are evaluated on the basis of 1-DH layout manner. Label 'Cost' refers to the cost-intelligent dynamic replication selection strategy

estimation. According to the analysis in Sect. 5, if the chosen layout was 1-DH, it would access the replica with stripe size of 16 KB; if the chosen layout policy was 1-DV, then it



(a) Ethernet



(b) InfiniBand

Fig. 8 Performance improvement compared to 16 KB stripe size (MPI-TILE-IO benchmark). In this set of experiments, stripe sizes were 16 KB, 256 KB, and 4 MB. Based on the request size of data access, the different stripe sizes can be thought of as 1-DH, 2-D, and 1-DV layout manners, respectively. For each case, we compare the performance of the other layout manners with 16 KB stripe size layout manner. The performance improvements are evaluated on the basis of 16 KB stripe layout manner. Label 'Cost' refers to the cost-intelligent dynamic replication selection strategy

would access 4 MB replica; and the replica with stripe size of 256 KB could be regarded as a 2-D layout manner. Figure 8 shows the results, in which the vertical axis represents the comparative performance improvement with the layout policy with 4 KB stripe size. The performance improvement of dynamic data access with hybrid replication strategy is around 13 $\sim$ 23% compared with other static layout policies.

For both configurations of mixed IOR workloads and MPI-TILE-IO workloads, the hybrid data replication scheme can achieve significant performance improvement compared with any single static data layout policy. Therefore, the cost-intelligent dynamic replication selection strategy is effective for mixed I/O workloads. The results show a great potential of trading underutilized storage for higher I/O performance.

7 Discussion

As described in previous paragraphs, the cost-intelligent data layout includes two approaches: layout optimization based on overall access cost analysis and dynamic data access with hybrid data replication. The former approach provides a quantitative method to select an optimal layout by summarizing all data accesses, while the hybrid data replication strategy improves the performance of all data accesses by sacrificing more storage capacity. The effectiveness of the proposed cost-intelligent data layout optimization relies on the accuracy of the cost model. When an application has a dominant data access pattern, the cost model alone should work well. When an application has more than one performance sensitive data access pattern, layout optimization by overall cost analysis or the hybrid data layout mechanism becomes a good companion to support the cost model.

To be effective, the cost-model needs to be simple and accurate, whereas the accuracy is in the sense of relative performance comparison, not the absolute bandwidth. To serve this purpose, only the overhead incurred on network and storage device are considered, and software overhead, such as the time spent on I/O client and file server software, is ignored. Also, this does not consider the impact of cache, buffer and lock contention during data access, or the potential TCP Incast [41–43] problem on the performance. Focus was placed on key parameters, such as latency, transmission time, and number of storage nodes, etc. Analytical and experimental results show that this was a good design choice. The cost model is confirmed to be feasible and able to serve its purpose.

The proposed hybrid replication strategy optimizes every data access for a data-intensive application. Therefore it generates even higher performance improvement than the overall access cost analysis approach. With the rapid development of storage technology, the capacity of hard disk drives keeps increasing rapidly, and the price reduces steadily. The proposed hybrid data replication optimization trades the available storage capacity for better I/O performance, and makes sense for performance-critical applications. While how to handle data consistency for writing in the hybrid layout approach is discussed, the hybrid approach is designed for read intensive applications. It is a good alternative for certain applications, not a solution designed for all. In a nutshell, the cost-intelligent hybrid data layout scheme proposed in this study is designed to tune data layout automatically in order to utilize existing parallel file systems.

8 Conclusion

Parallel file systems have been developed in recent years to ease the I/O bottleneck for data-intensive applications.

Extensive research efforts have been devoted to improving I/O performance, either by better arrangement of data accesses in parallel I/O library, or by better organization of data blocks on file servers. Nevertheless, little has been done for a better integration of application-specific data access characteristics and file system data layout. In this paper, a novel cost-intelligent data layout optimization scheme is proposed for data-intensive applications. The application-specific layout approach has made a three-fold achievement. It derives a data access model to estimate I/O cost for parallel file systems. We then use the model to analyze the overall I/O cost of different data layouts and to choose an optimal layout for a given application. Finally, we propose a hybrid data replication strategy for mixed I/O workloads, in which each file has multiple replications with different data layouts and each I/O request can be automatically dispatched to one replica with the least access cost to exploit the full potential of parallel I/O systems.

We have conducted experimental testing under the MPI program environment and PVFS2 file system to verify the proposed layout scheme. The results demonstrate that the cost model is accurate and effective: the improvement in I/O performance is up to 57% by overall access cost analysis, and the hybrid replication strategy can achieve 13∼74% improvement of I/O bandwidth compared to a single fixed layout policy. In summary, the proposed cost-intelligent application-specific data layout optimization is proved to be able to provide satisfactory improvement of I/O performance, especially for applications with various data access patterns. The overall I/O cost analysis approach provides a quantitative method to achieve I/O performance improvement by summarizing all data access, while the hybrid replication strategy trades the available storage capacity for the critical data access I/O performance. As is illustrated through this study, increasing the parallelism alone is not sufficient to improve the performance of parallel file systems. The trade-off between storage capacity and I/O performance, as proposed by the hybrid replication strategy, is a ‘must have’ feature for future high-performance file systems.

While this research has proposed and verified this design, it has also revealed more research issues. The cost model proposed in this study is designed for one application only. In principle, it should be extensible to multiple applications. However, the difficulty of the extension is in the separation and identification of the data access patterns of different applications, and the interference between them. Further study on cost-intelligent data layout scheme for multiple applications is required in the future.

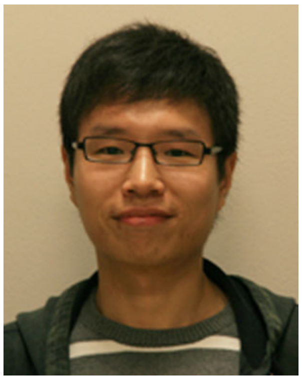
Acknowledgement The authors are thankful to Dr. Rajeev Thakur and Dr. Robert Ross of Argonne National Laboratory for their constructive and thoughtful suggestions toward this study. The authors are also grateful to anonymous reviewers for their valuable comments and suggestions. This research was supported in part by National Science Foundation under US NSF grant CCF-0621435 and CCF-0937877.

References

1. Lustre: A scalable, robust, highly-available cluster file system. White Paper, Cluster File Systems, Inc. (2006) [Online]. Available: http://www.lustre.org/
2. Schmuck, F., Haskin, R.: GPFS: A shared-disk file system for large computing clusters. In: FAST’02: Proceedings of the 1st USENIX Conference on File and Storage Technologies, p. 19. USENIX Association, Berkeley (2002)
3. Welch, B., Unangst, M., Abbasi, Z., Gibson, G., Mueller, B., Small, J., Zelenka, J., Zhou, B.: Scalable performance of the Panasas parallel file system. In: FAST’08: Proceedings of the 6th USENIX Conference on File and Storage Technologies, pp. 1–17. USENIX Association, Berkeley (2008)
4. Carns, P.H., Ligon, W.B. III, Ross, R.B., Thakur, R.: PVFS: A parallel file system for Linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference, pp. 317–327. USENIX Association, Berkeley (2000)
5. Thakur, R., Gropp, W., Lusk, E.: Data sieving and collective I/O in ROMIO. In: FRONTIERS’99: Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation, p. 182. IEEE Computer Society, Washington (1999)
6. Thakur, R., Choudhary, A.: An extended two-phase method for accessing sections of out-of-core arrays. Sci. Program. **5**(4), 301–317 (1996)
7. Seamons, K.E., Chen, Y., Jones, P., Jozwiak, J., Winslett, M.: Server-directed collective I/O in Panda. In: SC’95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (CDROM), p. 57. ACM, New York (1995)
8. Chen, Y., Sun, X.-H., Thakur, R., Song, H., Jin, H.: Improving parallel I/O performance with data layout awareness. In: Cluster’10: Proceedings of the IEEE International Conference on Cluster Computing 2010. IEEE Computer Society, Washington (2010)
9. Ching, A., Choudhary, A., Liao, W.-K., Ross, R., Gropp, W.: Efficient structured data access in parallel file systems. In: Cluster’03: Proceedings of the IEEE International Conference on Cluster Computing (2003)
10. Ching, A., Choudhary, A., Coloma, K., Liao, W.-K., Ross, R., Gropp, W.: Noncontiguous I/O accesses through MPI-IO. In: CCGRID’03: Proceedings of the 3rd IEEE International Symposium on Cluster Computing and the Grid, p. 104 (2003)
11. Nitzberg, B., Lo, V.: Collective buffering: improving parallel I/O performance. In: HPDC’97: Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, p. 148. IEEE Computer Society, Washington (1997)
12. Ma, X., Winslett, M., Lee, J., Yu, S.: Faster collective output through active buffering. In: IPDPS’02: Proceedings of the 16th International Parallel and Distributed Processing Symposium, p. 151. IEEE Computer Society, Washington (2002)
13. Isaila, F., Malpohl, G., Olaru, V., Szeder, G., Tichy, W.: Integrating collective I/O and cooperative caching into the “ClusterFile” parallel file system. In: ICS’04: Proceedings of the 18th Annual International Conference on Supercomputing, pp. 58–67. ACM, New York (2004)
14. Liao, W.-K., Coloma, K., Choudhary, A., Ward, L., Russell, E., Tideman, S.: Collective caching: Application-aware client-side file caching. In: HPDC’05: Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing, 2005. HPDC-14, pp. 81–90. IEEE Computer Society, Washington (2005)

15. Fu, J.W.C., Patel, J.H.: Data prefetching in multiprocessor vector cache memories. In: ISCA'91: Proceedings of the 18th Annual International Symposium on Computer Architecture, pp. 54–63. ACM, New York (1991)
16. Dahlgren, F., Dubois, M., Stenstrom, P.: Fixed and adaptive sequential prefetching in shared memory multiprocessors. In: ICPP'93: Proceedings of the 1993 International Conference on Parallel Processing, pp. 56–63. IEEE Computer Society, Washington (1993)
17. Patterson, R.H., Gibson, G.A., Ginting, E., Stodolsky, D., Zelenka, J.: Informed prefetching and caching. In: Proceedings of the 15th ACM Symposium on Operating Systems Principles, pp. 79–95. ACM Press, New York (1995)
18. Byna, S., Chen, Y., Sun, X.-H., Thakur, R., Gropp, W.: Parallel I/O prefetching using MPI file caching and I/O signatures. In: SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, pp. 1–12. IEEE Press, Piscataway (2008)
19. Lei, H., Duchamp, D.: An analytical approach to file prefetching. In: Proceedings of the USENIX 1997 Annual Technical Conference, pp. 275–288 (1997)
20. Tran, N., Reed, D.A., Member, S.: Automatic ARIMA time series modeling for adaptive I/O prefetching. IEEE Trans. Parallel Distrib. Syst. **15**, 362–377 (2004)
21. Chen, Y., Byna, S., Sun, X.-H., Thakur, R., Gropp, W.: Hiding I/O latency with pre-execution prefetching for parallel applications. In: SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, pp. 1–10. IEEE Press, Piscataway (2008)
22. Rhodes, P.J., Tang, X., Bergeron, R.D., Sparr, T.M.: Iteration aware prefetching for large multidimensional scientific datasets. In: SSDBM'05: Proc. of the 17th International Conference on Scientific and Statistical Database Management, Berkeley, CA, US, pp. 45–54 (2005)
23. Rubin, S., Bodík, R., Chilimbi, T.: An efficient profile-analysis framework for data-layout optimizations. SIGPLAN Not. **37**(1), 140–153 (2002)
24. Wang, Y., Kaeli, D.: Profile-guided I/O partitioning In: ICS'03: Proceedings of the 17th Annual International Conference on Supercomputing, pp. 252–260. ACM, New York (2003)
25. Hsu, W.W., Smith, A.J., Young, H.C.: The automatic improvement of locality in storage systems. ACM Trans. Comput. Syst. **23**(4), 424–473 (2005)
26. Huang, H., Hung, W., Shin, K.G.: FS2: Dynamic data replication in free disk space for improving disk performance and energy consumption. In: SOSP'05: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles, pp. 263–276. ACM, New York (2005)
27. Bhadkamkar, M., Guerra, J., Useche, L., Burnett, S., Liptak, J., Rangaswami, R., Hristidis, V.: BORG: Block-reORGanization for self-optimizing storage systems In: Proceedings of the 7th Conference on File and Storage Technologies, pp. 183–196. USENIX Association, Berkeley (2009). [Online]. Available: http://portal.acm.org/citation.cfm?id=1525908.1525922
28. Wang, C., Zhang, Z., Ma, X., Vazhkudai, S.S., Mueller, F.: Improving the availability of supercomputer job input data using temporal replication. Comput. Sci. Res. Dev. **23** (2009)
29. Song, H., Sun, X.-H., Yin, Y., Chen, Y.: A cost-intelligent application-specific data layout scheme for parallel file systems. In: HPDC'11: Proceedings of the 20th International ACM Symposium on High Performance Distributed Computing, pp. 37–48 (2011)
30. Seltzer, M., Chen, P., Ousterhout, J.: Disk scheduling revisited. In: Proceedings of the USENIX Winter Technical Conference, USENIX Winter '90, pp. 313–324 (1990)
31. Worthington, B.L., Ganger, G.R., Patt, Y.N.: Scheduling algorithms for modern disk drives. In: SIGMETRICS'94: Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 241–251 (1994)
32. Lumb, C.R., Schindler, J., Ganger, G.R., Nagle, D.F.: Towards higher disk head utilization: extracting free bandwidth from busy disk drives. In: OSDI'00: Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation, pp. 87–102. USENIX Association, Berkeley (2000)
33. Zhang, X., Jiang, S.: InterferenceRemoval: Removing interference of disk access for MPI programs through data replication. In: ICS'10: Proceedings of the 24th International Conference on Supercomputing, pp. 223–232 (2010)
34. Isaila, F., Tichy, W.F.: Clusterfile: a flexible physical layout parallel file system. In: Cluster'01: Proceedings of the 3rd IEEE International Conference on Cluster Computing, p. 37 (2001)
35. Wang, F., Xin, Q., Hong, B., Brandt, S.A., Miller, E.L., Long, D.D.E., Mclarty, T.T.: File system workload analysis for large scientific computing applications. In: Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies, pp. 139–152, Apr. 2004
36. Ligon, W.B., Ross, R.B.: Implementation and performance of a parallel file system for high performance distributed applications. In: HPDC'96: Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing, p. 471. IEEE Computer Society, Washington (1996)
37. Ruemmler, C., Wilkes, J.: An introduction to disk drive modeling. IEEE Comput. **27**, 17–28 (1994)
38. Tian, Y., Klasky, S., Abbasi, H., Lofstead, J., Grout, R., Podhorszki, N., Liu, Q., Wang, Y., Yu, W.: EDO: Improving read performance for scientific applications through elastic data organization. In: Cluster'11: Proceedings of the IEEE International Conference on Cluster Computing. Cluster, vol. 11 (2011)
39. Vijayakumar, K., Mueller, F., Ma, X., Roth, P.C.: Scalable I/O tracing and analysis. In: PDSW'09: Proceedings of the 4th Annual Workshop on Petascale Data Storage, pp. 26–31. ACM, New York (2009)
40. Yun, H.-C., Lee, S.-K., Lee, J., Maeng, S.: An efficient lock protocol for home-based lazy release consistency. In: CCGRID'01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid, p. 527. IEEE Computer Society, Washington (2001)
41. Phanishayee, A., Krevat, E., Vasudevan, V., Andersen, D.G., Ganger, G.R., Gibson, G.A., Seshan, S.: Measurement and analysis of TCP throughput collapse in cluster-based storage systems. In: FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies, pp. 1–14. USENIX Association, Berkeley (2008)
42. Vasudevan, V., Phanishayee, A., Shah, H., Krevat, E., Andersen, D.G., Ganger, G.R., Gibson, G.A., Mueller, B.: Safe and effective fine-grained TCP retransmissions for datacenter communication. In: SIGCOMM'09: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, pp. 303–314. ACM, New York (2009). [Online]. Available: http://doi.acm.org/10.1145/1592568.1592604
43. Vasudevan, V., Shah, H., Phanishayee, A., Krevat, E., Andersen, D., Ganger, G., Gibson, G.: Solving TCP incast in cluster storage systems (poster presentation). In: FAST'09: Proceedings of the 7th USENIX Conference on File and Storage Technologies (2009)

Huaiming Song is a research fellow in Dawning Information Industry Co., Ltd. Before joined Dawning, Dr. Song worked as a postdoctoral research associate in the Department of Computer Science at Illinois Institute of Technology (IIT). He received his B.S. degree in Computer Science from Xi'an Jiaotong University, and his Ph.D. degree in Computer Science from Institute of Computing Technology, Chinese Academy of Sciences. His research interests are Large-Scale and Data-Intensive systems, including parallel file systems, parallel databases and parallel computing. More information about Dr. Song can be found at http://mypages.iit.edu/~hsong20/.



Yanlong Yin is a Ph.D. candidate in the Department of Computer Science at the Illinois Institute of Technology (IIT). He received his B.E. degree in Computer Engineering and M.S. degree in Computer Science, both from Huazhong University of Science and Technology, China. Mr. Yin's research interests include parallel I/O systems and storage systems supporting for high-performance computing and data-intensive computing. More information about Mr. Yin can be found at http://www.iit.edu/~yyin2/.

Yong Chen is an Assistant Professor in the Computer Science Department of the Texas Tech University (TTU). He is the Director the Data Intensive Scalable Computing Laboratory at TTU. He received his B.E. degree in Computer Engineering and M.S. degree in Computer Science, both from University of Science and Technology of China, and his Ph.D. degree in Computer Science from Illinois Institute of Technology. Prior to joining TTU, Dr. Chen worked in the Future Technologies Group of the Computer Science and Mathmetics Division at the DOE Oak Ridge National Laboratory. Dr. Chen's research interests include parallel and distributed computing, high-performance computing, computer architectures and systems software. More information about Dr. Chen can be found at http://www.myweb.ttu.edu/yonchen/.

Xian-He Sun is the chairman and a professor of the Department of Computer Science, the director of the Scalable Computing Software laboratory at the Illinois Institute of Technology (IIT), and a guest faculty in the Mathematics and Computer Science Division at the Argonne National Laboratory. Before joining IIT, he worked at DoE Ames National Laboratory, at ICASE, NASA Langley Research Center, at Louisiana State University, Baton Rouge, and was an ASEE fellow at Navy Research Laboratories. Dr. Sun is an IEEE fellow and is serving and has been served as an associate editor or on the editorial board for many scholarly journals in the field of parallel and distributing processing and high performance computing, including the flagship journals of IEEE Transaction on Parallel and Distributed Systems and Journal of Parallel and Distributed Computing. His research interests include parallel and distributed processing, high-end computing, memory and I/O systems, and performance evaluation. He has close to 200 publications and 4 patents in these areas. More information about Dr. Sun can be found at his web site http://www.cs.iit.edu/~sun/.