A Cost-intelligent Application-specific Data Layout Scheme for Parallel File Systems

Huaiming Song [†], Yanlong Yin [†], Yong Chen [‡], Xian-He Sun [†]
[†] Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA
[‡] Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA
huaiming.song@iit.edu, yyin2@iit.edu, yong.chen@ttu.edu, sun@iit.edu

ABSTRACT

I/O data access is a recognized performance bottleneck of high-end computing. Several commercial and research parallel file systems have been developed in recent years to ease the performance bottleneck. These advanced file systems perform well on some applications but may not perform well on others. They have not reached their full potential in mitigating the I/O-wall problem. Data access is application dependent. Based on the application-specific optimization principle, in this study we propose a cost-intelligent data access strategy to improve the performance of parallel file systems. We first present a novel model to estimate data access cost of different data layout policies. Next, we extend the cost model to calculate the overall I/O cost of any given application and choose an appropriate layout policy for the application. A complex application may consist of different data access patterns. Averaging the data access patterns may not be the best solution for those complex applications that do not have a dominant pattern. We then further propose a hybrid data replication strategy for those applications, so that a file can have replications with different layout policies for the best performance. Theoretical analysis and experimental testing have been conducted to verify the newly proposed cost-intelligent layout approach. Analytical and experimental results show that the proposed cost model is effective and the application-specific data layout approach achieved up to 74% performance improvement for data-intensive applications.

Categories and Subject Descriptors

D.4.3 [**File Systems Management**]: Access methods; E.5 [**Files**]: Optimization**

General Terms

Performance

Keywords

data layout, data-access performance modeling, data-intensive, parallel file systems

HPDC'11, June 8–11, 2011, San Jose, California, USA.

1. INTRODUCTION

High-Performance Computing (HPC) applications like scientific computation and engineering simulations often involve large scale of data sets. They rely on parallel file systems for high-performance I/O service. The I/O behaviors of these applications are dependent on application characteristics and vary considerably from application to application in terms of request frequency, access concurrency, and data requested. Even for a given application, data access patterns may change from time to time. A complex application may not have a single dominant data access pattern, rather consists of different patterns at different phases. Data access pattern influences the performance of data access significantly. Unfortunately, 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 workload. Therefore, finding an appropriate data layout policy for a given application has great practical importance for HPC applications, especially data-intensive applications. Parallel file systems, such as Lustre [1], GPFS [2], PanFS [3], and PVFS2 [4] provide several different data layout policies for various workloads. However, the use of these data layout policies 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 the user is an expert on both sides, the application may have different patterns and may not find a perfect match with the options provided by the underlying file system. More research efforts are needed to explore an intelligent layout selection strategy and to explore the full potential of parallel file systems. In this paper, 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.

The bandwidth growth of storage systems has not kept pace with the rapid progress of the computing power due to the well-known Moore's law effect. The widening performance gap between CPU and I/O causes the processors to waste a large number of cycles waiting for data arrival and leads to the so called I/O-wall problem. Moreover, massive and out-of-order I/O requests further harm the efficiency of disks and I/O systems. For instance, several I/O requests that are issued to a disk simultaneously can cause the disk head to move back and forth frequently, which significantly degrades the overall data access throughput [5][6][7][8]. Disk performance reduces rapidly when serving requests in an interleaving fashion, and the performance degradation is especially acute for those read intensive applications, as the clients must wait (in a non-asynchronous manner) for data arrival from disks. The write performance may be less critical, because of the widely-used write behind technology [9][10]. Data write generally does not need to wait for the disk to finish.

In data-intensive and high-performance computing systems, the I/O performance optimization commonly consists of interface optimizations and storage optimizations. Interface optimizations, which are usually implemented at the I/O middleware layer, mainly focus on request re-arrangement. Request re-arrangement optimizations reduce I/O processing overhead by combining several small requests into a large or contiguous request, such as data sieving [11], List I/O [12], DataType I/O [13], and collective I/O [14][15][11][16]. Storage optimizations, which are usually implemented at the file server layer, mainly focus on data re-organization, such as data partition [17][18][19][20][21] or replication [22][23] to increase the degree of I/O parallelism. In other words, while data layout manners and user request patterns seriously affect the I/O performance in data-intensive applications, current I/O performance optimization strategies are not designed to catch 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.

Both request and data re-arrangement technologies have made their own success, however, to the best of our knowledge, little has been done to investigate a better integration of these two technologies to improve the overall I/O performance. As I/O interface and data storage are designed separately in the past, most of the interface optimizations focus on logical data address, without considering the physical data layout in the storage systems, which may limit the potential of the performance improvement. While data reorganization in parallel file system is often restricted by data access patterns occurred at the file system level, it needs sufficient application I/O workload information, through either profiling, tracing, or other mechanisms, to better optimize the data layout design of the application. Since data layout is static in nature, and I/O requests are dynamic and may not be determined until runtime, optimizing data layout is not an easy task.

In this paper, we propose a cost-intelligent data access strategy to link the interface optimization with the data layout 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 estimate the response time of data accesses for different data layout policies. (2) We present a static data layout optimization based on the overall cost estimation, which can choose an appropriate data layout for applications with specific I/O patterns. (3) We propose a cost-intelligent dynamic data access strategy with hybrid replication for those with mixed I/O workloads in the case that one layout policy cannot benefit all data accesses. (4) We implement the dynamic replication selection in MPI-IO library, which can automatically dispatch data access to one replica with the lowest access cost. Our analytical and experimental results show that the newly proposed cost-intelligent application-specific data layout approach is very promising and has a real potential in exploring 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 in data-intensive and high-performance computing systems. In Section 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 Section 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

As data access performance is widely recognized as one of the major bottlenecks in data-intensive parallel applications, a lot of efforts have been devoted to the improvement of I/O performance. The time consumption of an I/O request contains the time spent on I/O device (including disk head seek time and data read/write time) and the time spent on network (including an establish time and data transmission time) in a parallel file system. Most I/O optimizations focus on reducing the time of these two parts. Generally, the I/O performance suffers considerably if applications access data by making many small and non-contiguous I/O requests, as most of the overheads are on disk seeking and network establishing. Research in software optimization techniques can be roughly classified into several categories, including request arrangement optimizations, data organization optimizations, caching and prefetching techniques.

Request arrangement techniques mainly focus on merging a large number of small and non-contiguous data accesses into a small number of large and contiguous ones, to reduce the overall I/O time. These optimizations are usually implemented in runtime I/O libraries either in the parallel I/O middleware layer or parallel file system layer. Many research efforts have been devoted in this area for high I/O performance, such as data sieving [11], DataType I/O [13], List I/O [12], collective I/O [11][15][14][16], and two-phase I/O [25], etc. Data sieving technique [11] 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. DataType I/O [13] and List I/O [12] techniques are user-specified optimizations, that allow users to use one single I/O function representing rich I/O patterns. Collective I/O [11][15][14][16] techniques integrate many small concurrent I/O requests into one large and contiguous I/O request, and are widely used for parallel I/O optimization.

Research efforts on data organization mainly focus on physical data layout optimization [18][19][20][22][23][26] 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 manner to reduce disk head movements [5][6][7][8], thus to improve the overall I/O performance. Data partition [18][19] and replication [22][23][24] techniques are also commonly used to reduce disk head movements or to increase I/O parallelism. For example, Zhang et al [24] proposed a data replication scheme to amortize I/O workloads with 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, such as trace or profile analysis [18][19], to guide data partitioning across multiple disks or storage nodes.

Caching and prefetching can optimize the I/O performance largely because of locality and regularity of data accesses. In parallel I/O systems, caching techniques usually aim at storing data at client side buffer in a collective way [27][28][29][30], so that all I/O client processes can share data in their memories among multiple nodes through network. Data prefetching techniques aim at fetching data in advance, and can be roughly classified into two categories: informed way and speculative way. Informed data prefetching [31][32] [33][34] obtains data access patterns before data accessing, usually based on I/O trace analysis, profiling or hints. While speculative way usually prefetches data aggressively based on run-

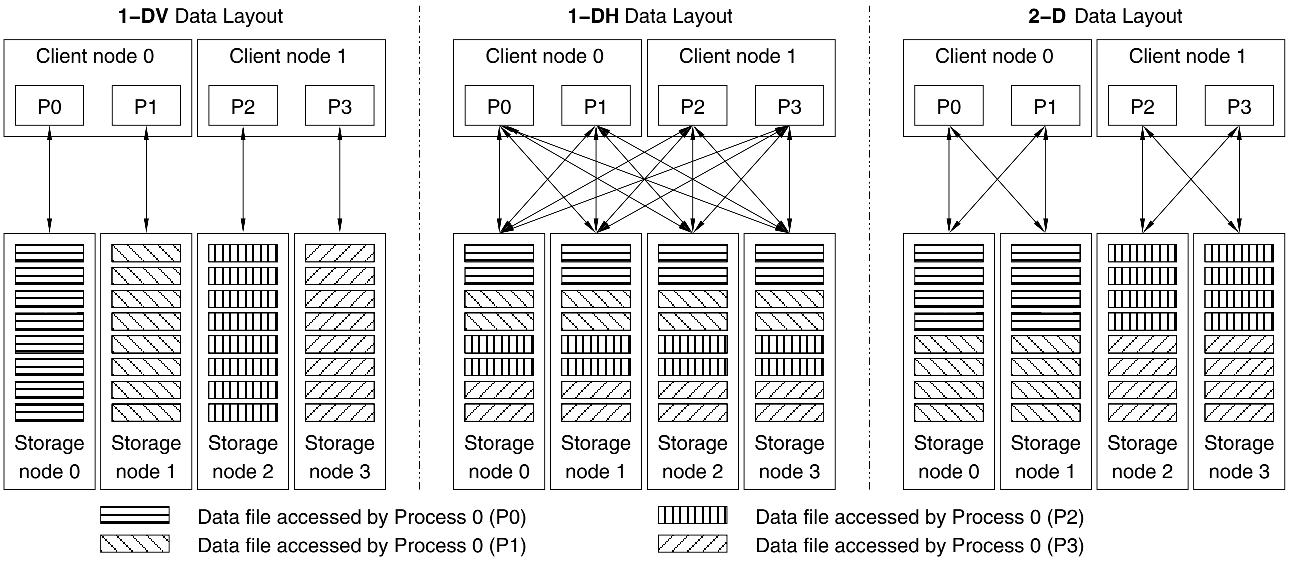

1−DV Data Layout
1−DH Data Layout
2−D Data Layout
Client node 0
Client node 1
P0
P1
P2
P3
Storage node 0
Storage node 1
Storage node 2
Storage node 3
Data file accessed by Process 0 (P0)
Data file accessed by Process 0 (P1)
Data file accessed by Process 0 (P2)
Data file accessed by Process 0 (P3)

Figure 1: Three commonly used data layout strategies.

time analysis, it is more suitable for data accesses without explicit patterns [35][36][37].

In parallel file systems, data is distributed among multiple storage servers to achieve high degree of I/O parallelism. There has been numerous research efforts 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 [17][38][39] analyzed how file system performance can be affected by many factors of workload such as the distribution manners of files, I/O request sizes, and I/O access characteristics.

Our approach is different from existing work. We propose a new cost model to guide data layout optimization for parallel I/O systems. We also propose a hybrid data replication strategy for applications with mixed I/O workloads, which can automatically perform I/O on one replica with the lowest cost for each data access.

3. COST ANALYSIS MODEL

Parallel file systems such as Lustre[1], GPFS[2], PanFS[3], and PVFS2[4], provide more than one data layout policy for different I/O workloads. We consider three most commonly used data layout policies: one dimensional vertical (1-DV for short), one dimensional horizontal (1-DH for short), and two dimensional (2-D for short) layout. These policies are demonstrated in Figure 1. The 1-DV data layout refers to the policy that data to be accessed by each I/O client process is stored on one given storage node. The 1-DH data layout is the simple striping method which distributes data across all storage nodes in a round-robin fashion. The 2-D data layout refers to the policy in which data to be accessed by each process is stored on a subset (called storage group) of storage nodes. What needs to be emphasized is that the data layout is process-oriented, rather than file-oriented.

Different layout policies in parallel I/O systems lead to different interactive behaviors between I/O client processes and storage nodes, and thus introduce different data access costs. Here the cost is defined as the time consumption for 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 $T_{storage}$).

Table 1: Parameters and descriptions

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 1 lists the most critical factors and the parameters in our model. Before getting into the details of the cost model, we make the following reasonable assumptions and considerations.

1. There is no overlap between client nodes and storage nodes, which means every data access involves network transmissions.

2. Each single I/O access is a contiguous request. Ideally, in 1-DH layout, each data access involves all n storage nodes; while in 2-D layout, all $\lceil \frac{n}{g} \rceil$ storage nodes in the corresponding storage group are involved.

3. Although each computing or storage node is featured with multi-process support, the application-layer parallel operations have to be handled one by one sequentially at the hardware layer, such as establishing multiple network connections and the startup operations for multiple data accesses to local disks.

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	$p \leq n$		me	msv	$\lceil \frac{p}{n} \rceil \alpha$	$\lceil \frac{p}{n} \rceil s\beta$
	$p > n$	$m \leq \lceil \frac{p}{n} \rceil$	$\lceil \frac{p}{n} \rceil e$	$\lceil \frac{p}{n} \rceil sv$		
		$m > \lceil \frac{p}{n} \rceil$	me	msv		
1-DH	$m \leq \lceil \frac{p}{n} \rceil$		pe	$\frac{psv}{n}$	$p\alpha$	$\frac{ps\beta}{n}$
	$m > \lceil \frac{p}{n} \rceil$		mne	msv		
2-D	$p \leq g$		$m\lceil \frac{n}{g} \rceil e$	msv	$\lceil \frac{p}{g} \rceil \alpha$	$\frac{\lceil \frac{p}{g} \rceil s\beta}{\lfloor \frac{n}{g} \rfloor}$
	$p > g$	$m \leq \frac{\lceil \frac{p}{g} \rceil}{\lfloor \frac{n}{g} \rfloor}$	$\lceil \frac{p}{g} \rceil e$	$\frac{\lceil \frac{p}{g} \rceil sv}{\lfloor \frac{n}{g} \rfloor}$		
		$m > \frac{\lceil \frac{p}{g} \rceil}{\lfloor \frac{n}{g} \rfloor}$	$m\lceil \frac{p}{g} \rceil e$	msv		

Taking the network cost as an example, the basic idea of constructing the cost model is described as follows. For each data access, the time spent on network, $T_{network}$, consists of T_e, the time spent on establishing the connection, and T_x, the time spent on transferring the data. In general cases of 1-DV data layout, each storage node is accessed by $\lceil \frac{p}{n} \rceil$ processes and needs to establish network connections with all of them sequentially. Here the number of sequential operations is $\lceil \frac{p}{n} \rceil$, thus $T_e = \lceil \frac{p}{n} \rceil e$ and $T_x = \lceil \frac{p}{n} \rceil sv$. The number of sequential operations should be chosen prudentially as network connections are affected by both computing nodes and storage nodes. In our model, we always choose the larger one when the two numbers of serial operations are different between storage nodes and computing nodes. Take 1-DV as an example, if the number of network connections on client nodes is larger than that of storage nodes ($m > \lceil \frac{p}{n} \rceil$), the number of client connections m is used, and thus $T_e = me$ and $T_x = msv$.

The storage cost $T_{storage}$ consists of T_s, the startup time, and T_{rw}, the time spent on actual data read/write. Compared with the network cost, the storage cost is more straightforward. This is because the number of sequential I/O operations is only determined by the number of assigned I/O client processes on the storage node.

The complete model covering all situations is constructed based on all the above mentioned considerations, and the cost formulas are listed in Table 2. By examining the formulas, we can obtain the following implications for data layout optimization.

1) While $p \ll n$, and s is large, the cost of 1-DH layout policy is the lowest among all three policies. The reasons are that each process is served by all n storage nodes and thus can obtain the maximum degree of parallelism. Also on each storage node, the small p (number of processes) and large s make the startup time T_s insignificant compared with data read/write time T_{rw}.

2) While $p \approx n$, 2-D layout policy produces higher bandwidth than the other two. It provides each process more than one server for parallel access and also lets each storage node serve a limited number of processes to avoid excessive disk seeks.

3) While $p \gg n$, 1-DV layout policy would be the best choice. There are too many processes that all storage nodes work at full capacity. In 1-DV layout each storage node serves the minimum number of concurrent I/O client processes, hence the network establishing time and storage startup time are both less than other data layouts.

4) While the performance of network or client nodes is the bottleneck rather than storage nodes, 2-D or 1-DV layout policy is better, because 1-DH might overwhelm the load of clients or network.

4. STATIC LAYOUT BASED ON OVERALL I/O COST

The proposed cost model is used for estimating data access time for each single I/O request in parallel I/O systems. Therefore, if we have a prior knowledge of data accesses for a data-intensive application, it can be further used to calculate the overall I/O cost. Fortunately, most data-intensive applications have regular data access patterns, thus I/O behavior can be learned from previous runs. For example, numerous tools can trace I/O requests for applications [34][43], thus it is feasible to calculate the overall cost for all data accesses in different layout policies. By comparing these overall costs, if a performance difference is found, the natural choice is to take one layout manner which generates the lowest cost for that application.

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,

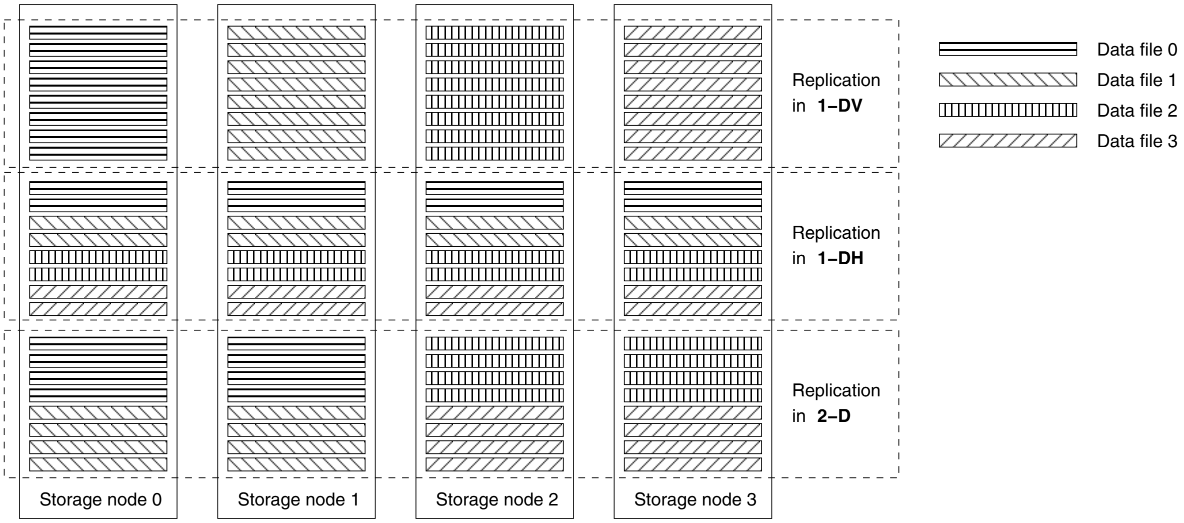


Figure 2: The hybrid data replication scheme

while I/O startup time with sequential data accesses is smaller than that of random data read/write.

Based on the overall cost calculated via I/O trace files, we are able to choose an optimal data layout policy. However, to fully utilize a data layout policy, we also need to consider some implementation issues. Application features and specific access patterns should be brought into consideration. Some revelation and optimization guidelines used in our implementation are discussed as follows.

1) If the chosen layout policy is 1-DV, there are two cases. In the first case when all I/O client processes access one shared file, it is recommended to stripe the file across all storage nodes with a stripe size equalling to the most common request size of the application. The stripe size is neither too large (otherwise a large number of I/O requests from multiple processes unevenly fall into one storage node), nor too small (otherwise data requested by each process is striped across many storage nodes like in 1-DH or 2-D layout). In the second case when each process reads/writes an independent file, it is recommended to place different files on different storage nodes.

2) If the chosen layout policy is 1-DH, it is suggested to distribute files with a relatively small stripe size to make full use of all available storage nodes, so that every I/O request can benefit from data access with highest degree of parallelism.

3) If the chosen layout is 2-D, there are also two cases. With the case of all processes accessing one shared file, it is recommended to stripe the file across all storage nodes with a stripe size that is $\frac{g}{n}$ times larger of the most common request size, thus most processes can access data from $\frac{n}{g}$ storage nodes. With the case of each process accessing one independent file, it is recommended to set file striped across $\frac{n}{g}$ storage nodes (this is natively supported by many parallel file systems, such as "*pvfs2-xattr -s -k user.pvfs2.num_dfiles...*" in PVFS2 and "*lfs setstripe ... -c OST_NUM*" in Lustre).

5. DYNAMIC ACCESS WITH HYBRID DATA REPLICATION

5.1 Hybrid Data Replication Strategy

The trace-based analysis of overall data access cost is practical for data layout optimization if the I/O workload has specific patterns, which makes it easy to choose one layout policy with the lowest cost. For example, as mentioned before, 1-DV layout policy is more suitable for a large number of concurrent I/O processes, while 1-DH 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. The difference of overall data access costs with different layout policies might be insignificant, thus it is not easy to determine which policy is the best. For example, during one run of an application with mixed I/O workloads, 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 makes it impossible for a static data layout policy to serve all data accesses in the most efficient way.

In order to solve this problem, we propose a dynamic access strategy with 'hybrid' data replications. Here, 'hybrid' means each file has several copies with different data layout policies in the parallel file system. For each data access, the strategy dynamically chooses one replica with the lowest access cost. Since each data access is assigned to the best fit replica, the hybrid data replication strategy can serve various kinds of I/O workloads with high performance. Figure 2 shows an example of the hybrid replication strategy. In this example, there are 4 files stored in 4 storage nodes, and each file has three replications in 1-DV, 1-DH, and 2-D data layout policies respectively. Each data access will be directed to one replica with the lowest data access cost, to improve the overall I/O performance. Generally, data read is simple, but data write is more complicated. There are a couple of possible solutions to handle write operations. In our design and implementation, we first write data to one replica with the lowest cost, and then apply lazy strategies [40][41] to synchronize data to the other replicas. Hence, for write operations, we only count data access cost on the chosen replica, and the cost of lazy data synchronization is considered as a background operation.

As mentioned in previous sections, three layout policies (1-DV, 1-DH, and 2-D) are defined based on the distribution manners of data accessed by each process. For the cases that each I/O client process reads/writes an independent file, it is natural to place several replicas with different layout policies. However, for the cases that all processes access a shared file, replicas with different layout policies can be defined in terms of stripe size: a big stripe size lets each process read/write data from a single file server, which is similar to 1-DV layout manner; a small stripe size lets each I/O process be served by all file servers in parallel, which is similar to 1-DH; and a medium stripe size can be regarded as 2-D layout manner, where each data access only involves a part of all storage nodes.

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

5.2 Implementation

The selection of replica for each data access is determined based on the cost estimation by the proposed model. We implement a prototype of cost estimation and dynamic data replica selection in MPI-IO libraries.

(1) File open and close. With the hybrid data replication strategy, in each `MPI_File_open()` or `MPI_File_close()` function, the library opens or closes all corresponding replicas with different layout policies. When opening the first file, it initiates the layout parameters for all the replicas. When closing a file, it synchronizes data for all replications.

(2) Data read/write. For each `MPI_File_read_xxx()` or `MPI_File_write_xxx()` function, the dynamic strategy first calculates I/O costs for all replicas, and then chooses one replica with the lowest cost to conduct read/write. When data access finished, it must synchronize data and offsets for all replicas. The data access procedure is shown as follows.

```
double cost_tmp;
int rep_idx;

// calculate data access cost for all replicas,
// and then choose the one with the lowest cost
cost_tmp = cost_cal(rep[0]);
for (i=1;i<rep_num;i++){
    if(cost_tmp > cost_cal(rep[i])){
        cost_tmp = cost_cal(rep[i]);
        rep_idx = i;
    }
}

//handle data read on the chosen replica
error = PMPI_File_xxx(rep[rep_idx], ...);

//synchronous offsets for all replicas.
off_sync();
...
```

For data reads, file offsets synchronization is to set the same offset for all file descriptors. For data writes, however, it is more complicated. As data is only written to one replica for each request, it has to synchronize data to other replicas to keep data consistency for different copies. In the prototype, after each data write, it will issue write requests of other replicas into a request queue. We design a dedicated data synchronization thread in client side to perform data write requests in the queue. Because data synchronization is a background operation, each data write can return right after putting the write requests into the queue, to be simple, we do not take the write costs of other data replicas into account.

The 'hybrid replication scheme' and 'dynamic data access strategy' are both transparent to users, and require no modification to existing programs. When I/O requests are issued, the embedded cost estimation function calculates data access cost automatically, and then decides which replica to access. In our experiments presented in Section 6, we use explicit data replications to show the effect of this strategy, where replications are fixed with certain layout policies and filenames. Compared with data access overhead, the overhead of the cost calculation for each data access is trivial and negligible.

6. EXPERIMENTAL RESULTS AND ANALYSIS

6.1 Experiments Environment

The experiments were conducted on a 65-node Sun Fire Linux-based cluster, in which there were one head node and 64 computing nodes. The head node was Sun Fire X4240, equipped with dual 2.7 GHz Opteron quad-core processors, 8GB memory, and 12 500GB 7200RPM SATA-II drives configured as RAID5 disk array. The computing nodes were Sun Fire X2200 servers, each node with dual 2.3GHz Opteron quad-core processors, 8GB memory, and a 250GB 7200RPM SATA hard drive. All 65 nodes were connected with Gigabit Ethernet. In addition, there were 17 nodes (including the head node) connected with 4X InfiniBand network. All these nodes ran Ubuntu 9.04 (Linux kernel 2.6.28.10) operating system. The experiments were tested on MPICH2-1.1.1p1 release and PVFS2 2.8.1 file system. In Ethernet environment, we employed all 64 computing nodes, of which 16 nodes work as file servers and the other 48 work as I/O client nodes. In InfiniBand testing, PVFS2 was configured with 8 file servers, and the rest 8 nodes were 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.

Our experiments consisted of two parts. The first part was to verify the data access cost model, including parameter estimation and accuracy verification. The second part was to use the overall I/O cost formulas to optimize the data layout, and also to verify the efficiency of the hybrid data replication strategy. We adopted the widely-used parallel file system benchmark IOR and mpi-tile-io to test the I/O performance. IOR is a software 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 our 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 approximate values of parameters are listed in Table 3.

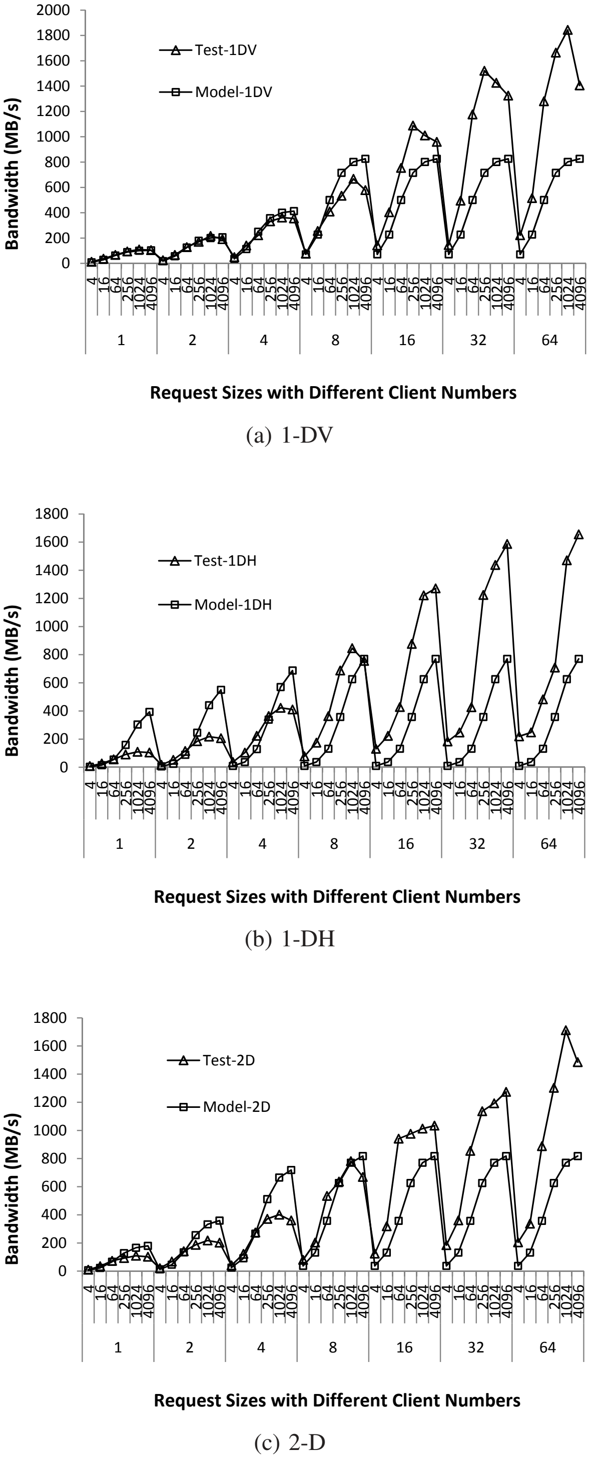
Table 3: Parameter values of the experimental platform

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

We then adopted these parameters to verify the accuracy of the cost model. In the verification experiments, IOR benchmark was tested over both Ethernet and InfiniBand interconnections. We varied the request size and the number of I/O client processes, with random and sequential data accesses. Figure 3 shows the I/O performance of model estimation and testing results with InfiniBand interconnection. In this figure, we compared performance under 1-DV, 1-DH, and 2-D data layout policies. The x axis represents different request sizes with different numbers of I/O client processes. The request sizes were 4KB, 16KB, 64KB, 256KB, 1024KB, and 4096KB, respectively. The numbers of client processes were 1, 2, 4, 8, 16, 32, and 64, respectively. As can be observed from the results, although there are differences in I/O bandwidth between testing value and model estimation, they show the same trends as request size and the number of I/O clients increase for all the three layout strategies. We also notice that when the number of I/O clients increases, the testing value is higher than the model estimation. One reason is that, the underlying Linux systems adopt buffer and cache technologies to optimize the I/O performance.

Next, we conducted experiments to verify the ability of the model to choose one layout policy with the lowest data access cost. Table 4 shows the statistical results on accuracy of the model estimation. 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. We used the cost model to estimate the I/O performance, and 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 4, the cost model performs well: the correct rate is around 80% to 85% in InfiniBand environment, and 88% to 91% with Ethernet interconnection. The results indicate that, the cost model can choose the best 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. From the results, we can also find out that, the 'Correct Rate' is a little better for random data access, and also it is a little better in Ethernet environment. One possible reason is that some optimizations carried out for sequential data access or in InfiniBand network are not included in our cost model.

Figure 4 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 IndfiniBand 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



(a) 1-DV

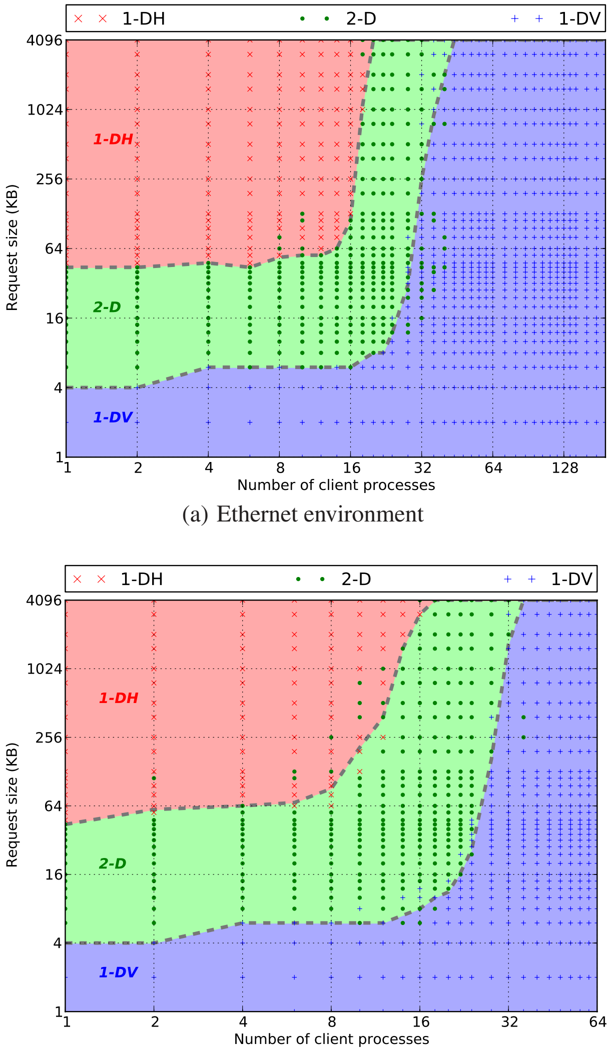
(b) 1-DH

(c) 2-D

Figure 3: Comparison of I/O bandwidth between model estimation and actual testing value for three data layout policies. These results were collected in InfiniBand environment. We varied the request size and the number of I/O client processes in this set of experiments. In each subfigure, the upper numbers on 'x' axis represent request sizes (KB), and the lower ones represent the numbers of I/O processes.

Table 4: 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%



(a) Ethernet environment

(b) InfiniBand environment

Figure 4: 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.

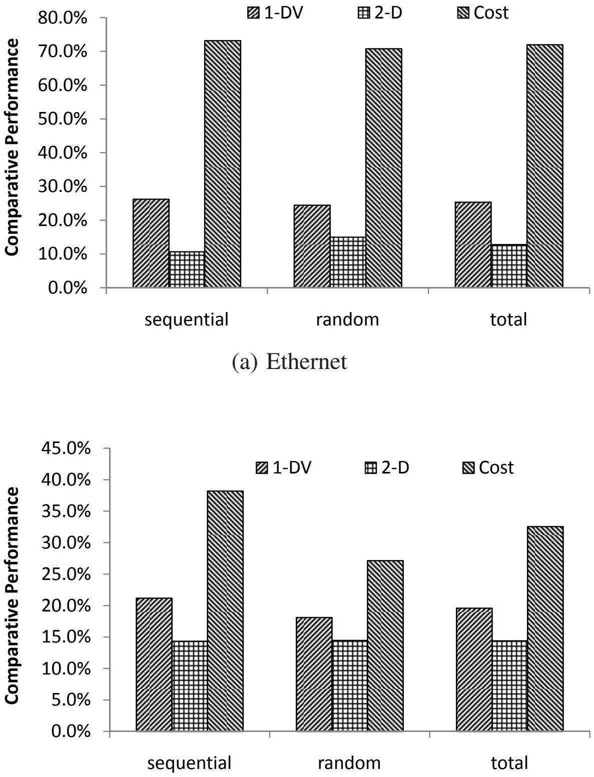
I/O bandwidth under three data layout policies, and then compared their performances. We only plotted the layout policy with the best I/O bandwidth for each access pattern. We also estimated the best performance by the proposed cost model and marked with different colors. From the figure, we observe that, in most cases the best testing performance is the same as the estimation of the cost model. We can also observe that, when the request size is very small, the 1-DV layout policy can get the best 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 is not too small, the 2-D layout can get the best I/O performance. The results are consistent with previous analysis of our cost model.

From the experimental results shown above, we can conclude that, the proposed model can 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%∼91% among the three layout policies. The proposed cost model can make accurate estimation on the performance of complex I/O workloads.

6.3 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 Ethernet tests and 1, 2, 4, 8, 16, 32, 64 in InfiniBand tests, respectively. The request sizes were 4KB, 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, 512KB, 1MB, 2MB, and 4MB, respectively. We configured one independent file with size of 64MB 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 5 shows the results of the IOR workloads, where y axis represents the comparative performance improvement with 1-DH layout policy. Here the performance is represented by the average I/O bandwidth, which is



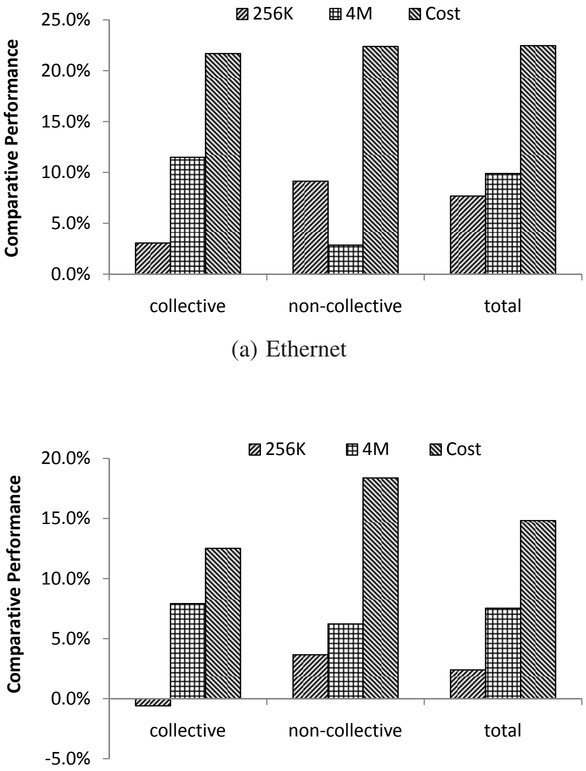
(a) Ethernet

(b) InfiniBand

Figure 5: 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.

calculated by the total data size divided by the total running time. As shown in Figure 5, the proposed cost-intelligent layout policy can get the best performance with both Ethernet and InfiniBand interconnections. The performance improvement is around 20∼74% compared with the other three data layout policies. The improvement of I/O performance with Ethernet interconnection is higher than that with InfiniBand connection.

In mpi-tile-io tests, we varied the tile size (corresponding to the request size) and the process numbers. 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, 256, respectively. Since mpi-tile-io benchmark reads only one big 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 16KB, 256KB, and 4MB, respectively. We measured the performance of collective I/O, 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 estimation. According to the analysis in Section 5, if the chosen layout was 1-DH, it would access the replica with stripe size of 16KB; if the chosen layout policy was 1-DV, then it would access 4MB replica; and the replica with stripe size of 256KB could be regarded as 2-D layout manner. Figure 6 shows the results, in which y axis represents the comparative performance improvement



(a) Ethernet

(b) InfiniBand

Figure 6: Performance improvement compared to 16KB stripe size (mpi-tile-io benchmark). In this set of experiments, stripe sizes were 16KB, 256KB, and 4MB. Based on the request size of data access, the different stripe sizes can be supposed as 1-DH, 2-D, and 1-DV layout manners, respectively. For each case, we compare the performance of the other layout manners with 16KB stripe size layout manner. The performance improvements are evaluated on the basis of 16KB stripe layout manner. Label 'Cost' refers to the cost-intelligent dynamic replication selection strategy.

with the layout policy with 4KB stripe size. The performance improvement of dynamic data access with hybrid replication strategy is around 13∼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 also show a great potential of trading unutilized storage for higher I/O performance.

7. DISCUSSION

As described in previous paragraphs, the effectiveness of the proposed cost-intelligent hybrid data layout optimization relies on the effectiveness of the cost model and the support of the hybrid data layout. 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, 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 the purpose we do not consider software overhead, such as the time spent on I/O client and file server software. We also do not consider the impact of cache, buffer and lock contention in data access, or the potential TCP Incast[42][44][45]problem on the performance. We focus on key parameters, such as latency, transmission time, number of storage group, etc. Analytical and experimental results show that we have made a good design choice. The cost model is confirmed to be feasible and able to serve our purpose.

With the fast developing of storage technology, the capacity of hard disk drives keeps increasing rapidly, while the price reduces steadily. The proposed hybrid data replication optimization trades the readily available storage capacity for better I/O performance, which makes sense for a lot of performance-critical applications. While how to handle write in the hybrid data 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 to utilize existing parallel file systems. It helps users to get better performance, but it will not deliver a better performance than the peak performance that the underlying system can deliver. We need to understand its limitations.

8. CONCLUSION

Poor I/O performance is a critical hurdle of high-performance computing systems, especially for data-intensive applications. Intensive research has been conducted in recent years in developing efficient parallel file systems and in data access optimization at different system I/O layers. However, little has been done in linking application-specific data access characteristics and file system data layout. In this paper, we introduce a cost-intelligent data access optimization scheme for data-intensive applications, and have achieved an application-specific data layout optimization through a three-fold approach. We first derive a data access cost model for parallel I/O systems. We then use the model to estimate the overall I/O cost of different data layout policies and choose an appropriate layout policy based on application I/O traces. Finally, we propose a hybrid data replication strategy for mixed workloads, in which each file has several replications with different layout policies and each I/O request can automatically choose one replica with the least access cost.

Experimental testing is conducted under the MPI program environment and PVFS2 file system. Experimental results show that the accuracy of the cost model in identifying an efficient data layout is in the range of 80∼91%, and the hybrid replication strategy can achieve 13∼74% performance improvement compared to single fixed layout policy. In summary, the proposed cost-intelligent application-specific data layout optimization is feasible and effective, and should be further investigated to explore the full potential of parallel file systems. The proposed hybrid replication strategy trades the available storage capacity for the precious data access I/O performance. As we illustrate through our study, increasing the parallelism alone is not sufficient to improve the performance of parallel file systems. The trade-off between disk capacity and I/O performance, as proposed by the hybrid replication strategy, is probably a 'must have' mechanism for future high-performance file systems.

The cost model proposed in this study is designed for one application only. In principle, it should be extensible to multiple applications. The difficulty of the extension is in the separation and identification of the data access patterns of different applications, and the interference between them. We plan to conduct a study on cost-intelligent data layout scheme for multiple applications in the future.

Acknowledgment

The authors are thankful to Dr. Rajeev Thakur, Dr. Robert Ross and Samuel Lang 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 NSF grant CCF-0621435 and CCF-0937877.

9. 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] F. Schmuck and R. Haskin, "GPFS: A Shared-disk File System for Large Computing Clusters," in *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2002, p. 19.

[3] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable Performance of the Panasas Parallel File System," in *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–17.

[4] 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*. USENIX Association, 2000, pp. 317–327.

[5] M. Seltzer, P. Chen, and J. Ousterhout, "Disk Scheduling Revisited," in *Proceedings of the USENIX Winter Technical Conference (USENIX Winter ąŕ90*, 1990, pp. 313–324.

[6] B. L. Worthington, G. R. Ganger, and Y. N. Patt, "Scheduling Algorithms for Modern Disk Drives," 1994, pp. 241–251.

[7] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *IEEE Computer*, vol. 27, pp. 17–28, 1994.

[8] C. R. Lumb, J. Schindler, G. R. Ganger, and D. F. Nagle, "Towards Higher Disk Head Utilization: Extracting Free Bandwidth from Busy Disk Drives," in *Symposium on Operating Systems Design and Implementation*. USENIX Association, 2000, pp. 87–102.

[9] J. A. Solworth and C. U. Orji, "Write-only Disk Caches," in *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1990, pp. 123–132.

[10] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-structured File System," *ACM Trans. Comput. Syst.*, vol. 10, no. 1, pp. 26–52, 1992.

[11] R. Thakur, W. Gropp, and E. Lusk, "Data Sieving and Collective I/O in ROMIO," in *FRONTIERS '99: Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation*. Washington, DC, USA: IEEE Computer Society, 1999, p. 182.

[12] A. Ching, A. Choudhary, K. Coloma, W.-k. Liao, R. Ross, and W. Gropp, "Noncontiguous I/O Accesses Through MPI-IO," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, p. 104, 2003.

[13] A. Ching, A. Choudhary, W.-k. Liao, R. Ross, and W. Gropp, “Efficient Structured Data Access in Parallel File Systems,” in *Proceedings of the IEEE International Conference on Cluster Computing*, 2003.

[14] K. E. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett, “Server-directed Collective I/O in Panda,” in *Supercomputing ’95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*. New York, NY, USA: ACM, 1995, p. 57.

[15] D. Kotz, “Disk-directed I/O for MIMD Multiprocessors,” *ACM Trans. Comput. Syst.*, vol. 15, no. 1, pp. 41–74, 1997.

[16] X. Zhang, S. Jiang, and K. Davis, “Making Resonance a Common Case: A High-performance Implementation of Collective I/O on Parallel File Systems,” in *IPDPS ’09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–12.

[17] F. Isaila and W. F. Tichy, “Clusterfile: A Flexible Physical Layout Parallel File System,” *IEEE International Conference on Cluster Computing* , vol. 0, p. 37, 2001.

[18] S. Rubin, R. Bodík, and T. Chilimbi, “An Efficient Profile-analysis Framework for Data-layout Optimizations,” *SIGPLAN Not.*, vol. 37, no. 1, pp. 140–153, 2002.

[19] Y. Wang and D. Kaeli, “Profile-guided I/O Partitioning,” in *ICS ’03: Proceedings of the 17th annual international conference on Supercomputing*. New York, NY, USA: ACM, 2003, pp. 252–260.

[20] W. W. Hsu, A. J. Smith, and H. C. Young, “The Automatic Improvement of Locality in Storage Systems,” *ACM Trans. Comput. Syst.*, vol. 23, no. 4, pp. 424–473, 2005.

[21] X.-H. Sun, Y. Chen, and Y. Yin, “Data Layout Optimization for Petascale File Systems,” in *PDSW ’09: Proceedings of the 4th Annual Workshop on Petascale Data Storage*. New York, NY, USA: ACM, 2009, pp. 11–15.

[22] H. Huang, W. Hung, and K. G. Shin, “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*. New York, NY, USA: ACM, 2005, pp. 263–276.

[23] M. Bhadkamkar, J. Guerra, L. Useche, S. Burnett, J. Liptak, R. Rangaswami, and V. Hristidis, “BORG: Block-reorganization for Self-optimizing Storage Systems,” in *Proccedings of the 7th conference on File and storage technologies*. Berkeley, CA, USA: USENIX Association, 2009, pp. 183–196. [Online]. Available: http://portal.acm.org/citation.cfm?id=1525908.1525922

[24] X. Zhang and S. Jiang, “IternterferenceRemoval: Removing Interference of Disk Access for MPI Programs through Data Replication,” in *Proceedings of the 24th International Conference on Supercomputing*, 2010, pp. 223–232.

[25] R. Thakur and A. Choudhary, “An Extended Two-phase Method for Accessing Sections of Out-of-core Arrays,” in *Scientific Programming, 5(4):301ÍC317, Winter*, 1996.

[26] C. Wang, Z. Zhang, X. Ma, S. S. Vazhkudai, and F. Mueller, “Improving the Availability of Supercomputer Job Input Data Using Temporal Replication,” *Computer Science - Research and Development*, vol. 23.

[27] B. Nitzberg and V. Lo, “Collective Buffering: Improving Parallel I/O Performance,” in *HPDC ’97: Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 1997, p. 148.

[28] X. Ma, M. Winslett, J. Lee, and S. Yu, “Faster Collective Output Through Active Buffering,” in *IPDPS ’02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*. Washington, DC, USA: IEEE Computer Society, 2002, p. 151.

[29] F. Isaila, G. Malpohl, V. Olaru, G. Szeder, and W. Tichy, “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*. New York, NY, USA: ACM, 2004, pp. 58–67.

[30] W.-k. Liao, K. Coloma, A. Choudhary, L. Ward, E. Russell, and S. Tideman, “Collective Caching: Application-aware Client-side File Caching,” in *HPDC ’05: Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 81–90.

[31] J. W. C. Fu and J. H. Patel, “Data Prefetching in Multiprocessor Vector Cache Memories,” in *ISCA ’91: Proceedings of the 18th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 1991, pp. 54–63.

[32] F. Dahlgren, M. Dubois, and P. Stenstrom, “Fixed and Adaptive Sequential Prefetching in Shared Memory Multiprocessors,” in *ICPP ’93: Proceedings of the 1993 International Conference on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 1993, pp. 56–63.

[33] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, “Informed Prefetching and Caching,” in *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*. ACM Press, 1995, pp. 79–95.

[34] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp, “Parallel I/O Prefetching Using MPI File Caching and I/O Signatures,” in *SC ’08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[35] H. Lei and D. Duchamp, “An Analytical Approach to File Prefetching,” in *Proceedings of the USENIX 1997 Annual Technical Conference*, 1997, pp. 275–288.

[36] N. Tran, D. A. Reed, and S. Member, “Automatic Arima Time Series Modeling for Adaptive I/O Prefetching,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, pp. 362–377, 2004.

[37] Y. Chen, S. Byna, X.-H. Sun, R. Thakur, and W. Gropp, “Hiding I/O Latency with Pre-execution Prefetching for Parallel Applications,” in *SC ’08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–10.

[38] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. Mclarty, “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*, Apr. 2004, p. 139–152.

[39] W. B. Ligon III and R. B. Ross, “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. Washington, DC, USA: IEEE Computer Society, 1996, p. 471.

[40] H.-C. Yun, S.-K. Lee, J. Lee, and S. Maeng, "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*. Washington, DC, USA: IEEE Computer Society, 2001, p. 527.

[41] Y. Sun and Z. Xu, "Grid Replication Coherence Protocol," in *IPDPS'04: Proceedings of 18th International Parallel and Distributed Processing Symposium*, vol. 14, p. 232, 2004.

[42] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "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*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14.

[43] K. Vijayakumar, F. Mueller, X. Ma, and P. C. Roth, "Scalable I/O Tracing and Analysis," in *PDSW '09: Proceedings of the 4th Annual Workshop on Petascale Data Storage*. New York, NY, USA: ACM, 2009, pp. 26–31.

[44] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592604

[45] V. Vasudevan, H. Shah, A. Phanishayee, E. Krevat, D. Andersen, G. Ganger, and G. Gibson, "Solving TCP Incast in Cluster Storage Systems (poster presentation)," in *FAST'09: Proceedings of the 7th USENIX Conference on File and Storage Technologies*. 2009.