A Server-Level Adaptive Data Layout Strategy for Parallel File Systems

Huaiming Song [†§], Hui Jin [‡], Jun He [‡], Xian-He Sun [‡], Rajeev Thakur [♯]

[†]*Research and Development Center, Dawning Information Industry, Beijing, 100193, China*
[‡]*Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA*
[♯] *Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA*
songhm@sugon.com, {hjin6, jhe24, sun}@iit.edu, thakur@mcs.anl.gov

***Abstract*—Parallel file systems are widely used for providing a high degree of I/O parallelism to mask the gap between I/O and memory speed. However, peak I/O performance is rarely attained due to complex data access patterns of applications. Based on the observation that the I/O performance of small requests is often limited by the request service rate, and the performance of large requests is limited by I/O bandwidth, we take into consideration both factors and propose a server-level adaptive data layout strategy. The proposed strategy adopts different stripe sizes for different file servers according to the data access characteristics on each individual server. We let the file servers that can fully utilize bandwidth hold more data, and the file servers that are limited with request service rate hold less data. As a result, heavy-load servers can offload some data accesses to light-load servers for potential improvement of I/O performance. We present a method to measure access cost for each data block and then utilize an equal-depth histogram approach to distributed data blocks across multiple servers adaptively, so as to balance data accesses on all file servers. Analytical and experimental results demonstrate that the proposed server-level adaptive layout strategy can improve I/O performance by as much as 80.3% and is more appropriate for applications with complex data access patterns.**

***Keywords*-Server-level adaptive data layout, variable stripe size, equal-depth histogram, data layout optimization, parallel file system**

I. Introduction

High-performance computing applications, such as scientific computation and engineering simulations, often involve large data sets. However, performance improvements in computing technology have vastly out-paced the improvements in storage technology. Parallel file systems, such as Lustre [1], GPFS [2], pNFS [3], PVFS2 [4], and PanFS [5], 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.

In parallel file systems, one data file is usually striped across multiple file servers with a fixed stripe size. This design benefits simple, well-formed, and generic data access patterns for two reasons: parallel data accessing and balanced data size on multiple servers. Nevertheless, performance varies from application to application due to their complex I/O workloads, meaning that data access pattern may vary from time to time. The request size can be large or small, and some data blocks may be accessed many times, while others seldom accessed. Due to the non-uniform access characteristics, equal data size on multiple file servers cannot guarantee unified data access on them. For example, some file servers may involve lots of small data accesses, which leads to low I/O bandwidth; while others may have large data accesses, achieving a full utilization of bandwidth. In addition, the number of involved I/O requests is different for different file servers. As a result, peak I/O performance is rarely attained in parallel file systems due to application's non-uniform data access.

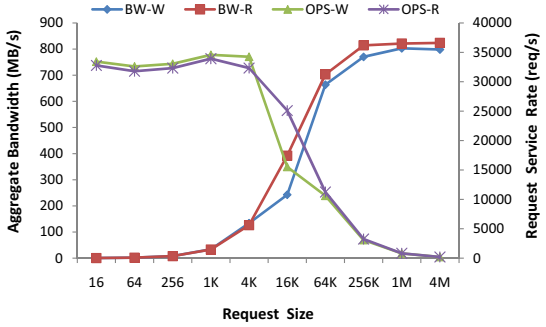


Figure 1. I/O performance for different request sizes. Labels 'BW-W' and 'BW-R' refer to bandwidth for data writing and reading. Labels 'OPS-W' and 'OPS-R' refer to request service rate.

Generally, I/O performance in each file server has two components, request service rate and I/O bandwidth, which are usually measured by the metrics 'IOPS' and 'MB/s'. If one file server receives a burst of small I/O requests, although the bandwidth is far from saturated, the request service rate will be the bottleneck. On the other hand, if one file server only serves large requests, the bandwidth will get the peak value although the request service rate is not high. Figure 1 shows the I/O performance of the IOR benchmark under different request sizes; these experiments were conducted in an eight-node PVFS2 system. From the results it can be seen that, when the request size is very small (less than 1 KB), the I/O bandwidth is small, but the request service rate reaches the peak value; when the request size increases (larger than 256 KB), the request service rate goes down, but the bandwidth gets saturated. The results indicate that both request service rate and bandwidth can

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.246

limit I/O performance. Existing parallel file systems treat all file servers equally, without the awareness of different access characteristics on them. This overlooking of access characteristics on each individual file server may be the loss of a potential optimization opportunity that could benefit the overall performance of parallel file systems.

In this paper, we take into consideration both service rate and bandwidth on all file servers and propose a server-level adaptive data layout strategy for parallel file systems. The basic idea is that we let those file servers that can fully utilize bandwidth hold more data, and those file servers limited with service rate hold less data. Thus heavy-load servers can offload some data accesses to light-load servers by an optimized data block allocation strategy in order to balance their I/O workloads. The proposed data layout strategy better utilizes the potential of all file servers and is thus able to improve the overall I/O performance. We propose a methodology for distributing data blocks across multiple file servers adaptively.

Specifically, this paper makes the following contributions.

(1) We argue that considering both request service rate and I/O bandwidth restrictions would be greatly beneficial for performance improvement of data-intensive applications with non-uniform data access patterns.
(2) A server-level adaptive layout strategy is proposed, which adopts different stripe sizes on different file servers, to improve the overall I/O performance of parallel file systems.
(3) An equal-depth histogram approach is presented to redistribute data across file servers adaptively, to balance I/O workload on them.
(4) Extensive analytical and experimental testing were conducted to verify the proposed server-level adaptive layout strategy. Experimental results demonstrate that the proposed adaptive data layout is promising in performance improvement and has a real potential for applications with complex I/O workloads.

The rest of the paper is organized as follows. Section II reviews the related work on data access cost analysis and data layout optimization. Section III makes an analysis of access costs of data blocks, and presents how to distribute data blocks to balance I/O workloads across multiple file servers. Section IV proposes the server-level adaptive data layout scheme. Experimental and analytical results are presented in Section V. Finally, Section VI concludes this study and discusses future work.

II. Related Work

I/O performance is a recognized system bottleneck for many large-scale and data-intensive applications. Existing parallel file systems improved I/O performance by providing high degree of I/O parallelism. However, due to the non-uniform data access patterns, peak I/O performance is rarely achieved for most data-intensive applications. Numerous research efforts have been devoted to improving I/O performance by data access analysis and data layout optimization.

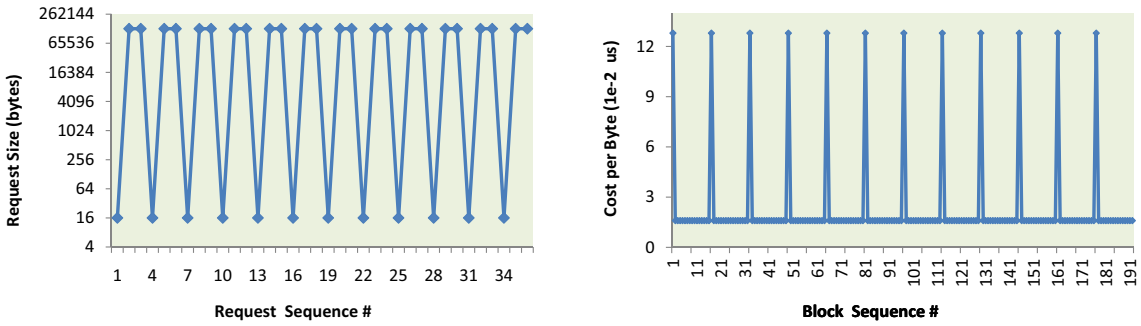
A. Data Access Analysis

It is well known that I/O performance highly depends on application data access patterns in large-scale and data-intensive systems. In order to explore potential performance improvement, a collection of data access analysis techniques [6][7][8][9][10][11] have been developed for parallel I/O systems. Research on data access analysis has mostly focused on access pattern discovery [8][12][13] and data access cost analysis [10][11][14]. The analytical results can be used for data access optimization (e.g., caching and prefetching [8][15]), or data layout optimization on file servers [11][16][17].

Smirni et al. [12] and Madhyastha et al. [13] studied I/O characteristics of scalable parallel applications and provided a classification of I/O patterns based on three access features: I/O operation type, sequentiality, and request size. I/O workloads studies conducted by Kotz et al. [18] show that in many data-intensive applications a large number of I/O requests are small and have irregular patterns. Several I/O trace tools [8][9] were developed to collect and analyze data access behaviors of data-intensive and parallel applications. Also there are some studies [6][7] focusing on the methodologies of parallel I/O trace analyzing. Our previous work [11] proposed a cost model to estimate data access time for various I/O patterns in different data layouts in parallel file systems. Chen et al. [14] and Song et al. [17] proposed to use a combination of request frequency (or number of requests) and request size to measure the access cost of data blocks, and then make a better allocation of these blocks. In this paper, we measure the access cost of all data blocks based on the same considerations as in other work [14][17]. We propose to employ an equal-depth histogram approach to redistribute data blocks based on their costs, to make a full potential of I/O performance improvement.

B. Layout Optimization

Research efforts on data layout optimization mainly focus on data reorganization across multiple file servers according to the I/O workloads of applications. Since I/O requests usually fall into several patterns in parallel file systems, it is possible to reorganize the data layout to reduce the number of disk seeks [19][20][21]. These data reorganization techniques improve the overall I/O performance due to a prior knowledge of application I/O behaviors. Besides, data partition [22][23] and replication [11][16] techniques are also widely used either to reduce disk head movements or to increase the degree of I/O parallelism. For example, Zhang et al [16] proposed a data replication scheme to amortize I/O workloads with multiple replicas to improve the performance, so that each file server only serves requests from one or a limited number of processes. Weil et al



(a) Data access patterns of LANL App1

(b) Cost per byte for data blocks of LANL App1

Figure 2. Data access cost analysis of LANL Anonymous App1 [26]. In subfigure (a), App1 consists of a large loop of sequential data writes, and in each loop there are 3 I/O requests: 1 small request and 2 large requests. In subfigure (b), data blocks involving small requests have higher access cost-per-byte(calculated as Cost(B_i)/BLOCK_SIZE for each data block) than others.

[24] employed a pseudo-random data distribution strategy, which can adaptively balance I/O workloads when adding or removing storage devices. PLFS[25] rearranged application data so that N-1 write pattern is better suited in the underlying file system, which is especially suitable for checkpointing applications. However, all of above approaches do not consider the differences of access cost of data blocks. Our previous work [11] proposed a hybrid replication scheme for applications with complex data access patterns, and dynamically chose one replica with the lowest access cost for each data access. For those applications that have different access patterns in different parts of files, we also proposed to adopt a segment-level adaptive layout[17], to achieve high I/O performance for all I/O requests.

All these data reorganization techniques have succeeded in their own right. However, little has been done to make an insight of the differences of data access patterns between different file servers in a clustered parallel file system. Our work takes into consideration both I/O bandwidth and request service rate limitations for each individual file server and uses an adaptive data layout to improve the overall I/O performance.

III. Data Access Cost Analysis

Due to the non-uniform data access of data-intensive applications, the different data blocks cannot be regarded as the same. Some data blocks may be accessed many times, while others may be seldom accessed. Besides, some blocks may be accessed by large requests, while others may be involved in small data accesses. In order to explore the potential performance improvement on all file servers, we first present an approach to analyze data access cost of all data blocks, and then introduce an adaptive method to distribute these data blocks across all file servers based on the cost analysis. In this paper, the access cost of a data block refers to cost per byte, i.e. the time spent for access each byte in the block.

A. Data Access Cost Analysis

Request size and access frequency are two key factors that affect I/O performance for each file server in parallel file systems. Similarly, we consider request size and access frequency to analyze the access cost for each data block, in order to allocate these data blocks on multiple file servers. Generally, each data access on a storage node includes the startup time and data read/write time [17]. The startup time, including disk seek time and software response time, is independent of the request size. The data read/write time is proportional to the data size. Assume that startup time for each data access is α on one node, and the transmission time of a single unit of data is β. We use K to represent the prorated number of data accesses and S to represent the total requested size on a data block. Thus the access cost of one data block can be roughly calculated as

$$Cost(B_i) = K_i \cdot \alpha + S_i \cdot \beta \quad (1)$$

Here i refers the serial number of a data block, which is between 0 and FILE_SIZE/BLOCK_SIZE. S_i refers to the total requested data size on that data block. K_i is a prorated cost determined by the number of requests that involve this block. The data block used for analysis is a logical unit, and different block size would introduce different number of file segmentations, resulting in a different cost distribution.

For analyzing the access cost, we use a counter for each data block, which indicates the prorated number of requests issued to it. If a request falls within one data block, the value of the counter for that block is increased by 1. If a large request involves m data blocks, the counters of all involved blocks are increased by $1/m$. That is because all these blocks share the request with only one startup time, as reflected in Formula 1.

To make the analysis of access cost more clearly, we take the I/O trace of LANL Anonymous App1 [26] as an example. In this application, there are three I/O requests in each loop, one small request with 16 bytes, and followed

by two large requests with (128K-16) bytes and 128 KB, respectively. Figure 2 shows the sequence of request size the and the cost per byte of every data block. Subfigure (a) shows the size of each request in the access sequence, and subfigure (b) shows the cost per byte of all data blocks. We set the block size as 16 KB for analysis. From the results we can see that the data blocks that involve small data requests have much higher access cost than those involving large requests. Therefore, it is beneficial to distribute data blocks across multiple file servers with the awareness of individual differences of the access costs of all data blocks.

The access cost formula makes an comprehensive consideration of both requests size and access frequency. If a data block has a lot of small data accesses, then it can be regarded as a high cost block. Likewise; if a data block is always involved in large data accesses, the cost per byte is usually not too high. For simplicity, we call a data block has high cost per byte the 'high cost block', and a file server has many high cost blocks the 'high cost server'. As a result, file servers that have too many high cost blocks will lead to low I/O bandwidth. The purpose of the proposed server-level adaptive layout is to make a full utilization of all file servers, by moving some data from high cost servers to low cost servers.

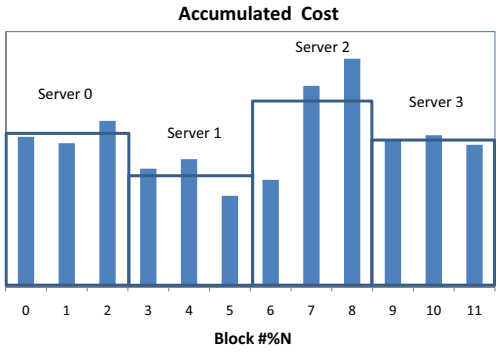
B. I/O Workload Redistribution

The server-level adaptive layout adopts different stripe sizes for different file servers. Assume the number of file servers is n, and their individual stripe sizes are: s_1, s_2, s_3, ... s_n. The proposed data layout does not change the overall stripe size of each round. If we use the block size b to calculate block access costs, and the number of data blocks for each round is N, we can calculate the round size as

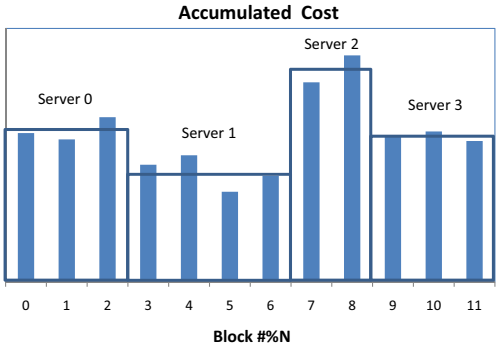
$$\sum_{i=1}^{n} s_i = N \cdot b = n \cdot s. \tag{2}$$

Here s s the fixed stripe size of existing parallel file systems. The proposed adaptive data layout employs the same round-robin way to distribute data blocks, with each file server hold different numbers of blocks, and all file servers keep a fixed number of data blocks for each round. Therefore, it is not difficult to understand that for two blocks id_i and id_j, if $id_i \equiv id_j(modN)$, they will be placed in the same file server. As a result, we can sum up data access cost of all blocks with the same remainder when dividing their block id by N. Thus we can get N values of the accumulated data access costs, namely c_0, c_1, c_2, ... c_{N-1}. Therefore, the data block redistribution can be converted to dividing the access costs series into n (the number of file servers) subsets, to balance the data access costs of all file servers.

We then adopt an equal-depth histogram [27] method to divide the costs of data blocks (indexed from 0 to $N-1$) into following n parts by $n-1$ breakpoints x_1, x_2, ... x_{n-1}



(a) Equal-width data block distribution



(b) Equal-cost data block distribution

Figure 3. Two data distribution manners. The height of each bar refers to the accumulated cost of data blocks, and the area of each rectangle refers to the access cost of one file server, which is the sum of all accumulated costs in that server.

$(0 < x_1 < x_2 < ... < x_{n-1} < N-1)$.

$$S = \{S_1[0, x_1], S_2[x_1+1, x_2], ..., S_n[x_{n-1}+1, N-1]\}$$

These n parts have the same or very close subtotal access cost. One file server holds one subset of all the data blocks. The access cost of each file server is calculated as follows (to make the equation true for the first file server and the last one, we let $x_0 = -1$ and $x_n = N-1$).

$$Cost(FS_i) = \sum_{j=x_{i-1}+1}^{x_i} c_j \qquad (i = 1 \sim n)$$

The equal-depth histogram approach is applied to guarantee all the file servers get the same or very close data access cost. Figure 3 shows an example of the equal-depth histogram approach of block redistribution. In this figure, there are 4 file servers, and each round consists of 12 data blocks. In subfigure (a), all file servers get the same number of data blocks (meaning the same stripe size), similar to the simple striping manner in existing parallel file systems. In subfigure (b), data block allocation is based on their access costs, although different file servers have different number of data blocks, the workload is more balanced.

After the data block re-distribution, file servers have large requests hold more data than those have small data accesses. As a result, all file servers could remain the same or very close busyness during runtime. The equal-depth histogram approach make all I/O workloads are more evenly distributed among the file servers, and it can exploit the potential I/O performance of all nodes.

IV. Server-level Adaptive Layout

As described in aforementioned section, the proposed server-level adaptive layout moves some data from heavy load servers to light load servers. Figure 4 is a diagram of the server-level adaptive layout. In this figure, the data file is distributed on four file servers with different stripe sizes. In this section, we present two approaches to achieve the adaptive data layout: offline approach and online approach. The offline approach optimize data layout according to a prior knowledge of applications, while the online approach dynamically adjust the stripe size on multiple file servers based on runtime information.

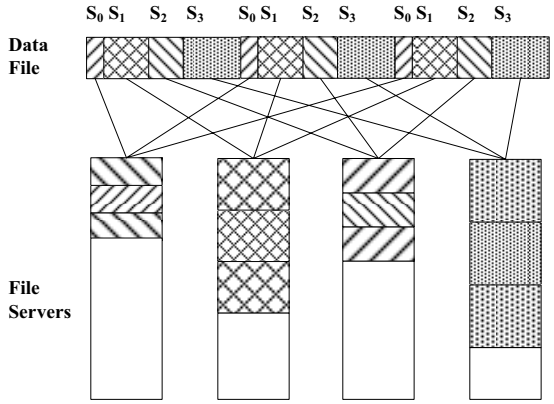


Figure 4. A diagram of server-level adaptive layout. In this figure there are 4 file servers with stripe size: S_0, S_1, S_2, and S_3. It shows how data is distributed across file servers.

A. Offline Approach

The idea of offline approach is that data cost analysis and block allocation are conducted offline, and the analyzed result is then used for data layout optimization of future runs of applications. Since the cost analysis is conducted offline, this approach needs a prior knowledge of data access patterns of applications. There are a large body of studies on data access pattern analysis using parallel I/O traces [6][7][8][9]. Usually, the data access pattern can be described in I/O trace files, I/O signature [8], profiles [28][29], and hints [30], etc. The offline approach can utilize all of them to analyze access cost and distribute data blocks across file servers.

Basically, the offline optimization includes two phases: cost analysis and block distribution. In the cost analysis phase, the analyzing module takes access pattern as input, calculates all cost of all logical data blocks as described in Section III, and then outputs the accumulated cost series: c_0, c_1, c_2, ... c_{N-1}. In the block distribution phase, the distribution module takes the cost series as input, and outputs the optimized block distribution through the equal-depth histogram method. Both phases need the parallel file system parameters for calculation. The optimized data layout then can be used for file distribution of the application, either by create new files for later runs, or adjust file layout in existing file system.

B. Online Approach

The idea of online approach is that, data block cost analysis and data block distribution are conducted dynamically at runtime. We designed a local array for each file server to keep access costs of all data blocks on that file server (integer values calculated by Formula 1). The array index is the block id. Since the data block is a logical concept and the size could be configurable, the space taken by the cost array is negligible. For example, if the block size is 4 KB, and the cost is a 4-byte integer, the array size is 0.1% of the file size on each file server. Therefore, the whole cost array can reside in memory at runtime. The array size could be even smaller if we choose a large block size.

With the online approach, when a file server receives a data access, it will update the cost array according to Formula 1 for involved blocks. Periodically, the data cost array is analyzed on each file server, and then the accumulated block costs are calculated. A global analysis module figures out the optimal strip size of each file server. After the calculation of optimal data layout, data need to be migrated between file servers to balance the I/O workload. We design an online data migration module to move data between file servers. Usually, data moving happens between adjacent file servers, so the volume of network transmission is not supposed to be too large. In order to reduce the overhead introduced by extra data read/write on file servers, we recommend moving data when the system resource usage is in low level.

C. Implementation

We have implemented a prototype of the server-level adaptive layout scheme in PVFS2[4]. We designed a module to estimate system parameters. We choose one file server in the parallel file system to test α and β, by measuring the parameters with different request sizes and repeating each case with multiple runs(the number is configurable), and then calculating their average values. We utilize the variable stripe size feature of PVFS2, by which stripe size can be configured to be different on different file servers. We use the '`setfattr`' command to set data distribution of directories using the POSIX interface (or the '`pvfs2-xattr`' command with the direct PVFS2 interface).

Table I
NODE INFORMATION OF EXPERIMENT PLATFORM

CPU	Quad-Core AMD Opteron(tm) Processor 2376 * 2
Memory	4 * 2GB, DDR2 333MHz
Storage	SATA II 250GB, 7200RPM
Network	1 gbps Ethernet, and 16 nodes with 4X InfiniBand (8 gbps)
OS	Ubuntu 4.3.3-5, Linux kernel 2.6.28.10
PVFS2	PVFS2 version 2.8.1

For the offline optimization approach, we directly use variable stripe sizes to distribute data files for future runs after the data block cost analysis. For the online optimization, it is more complicated. In our prototype, we simply recreate files by making a copy at runtime when the system resource is idle. However, this is not the optimal method, as it involves large amount of data shipping between file servers. An optimal way should keep the amount of data shipping minimal. We will further investigate the optimal data migration strategy for the online approach in our future work.

V. EXPERIMENTS

A. Experimental Setup

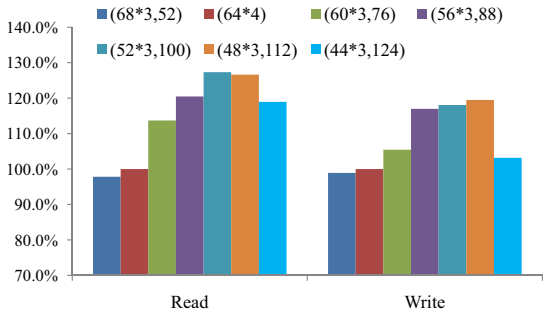
Experimental tests were conducted for the proposed server-level adaptive data layout with several benchmarks. The experiments were conducted on a 65-node SUN Fire Linux cluster, in which there are 64 computing nodes and one head node. The head node is Sun Fire X4240, with dual 2.7 GHz Opteron quad-core processors and 8 GB memory. The hardware and software configuration of computing nodes are shown in Table I.

We evaluated the proposed server-level adaptive layout with the widely-used parallel file system benchmark IOR and MPI-TILE-IO. 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. We also evaluated the layout with the I/O trace of a real application, 'Anonymous LANL App 1' [26].

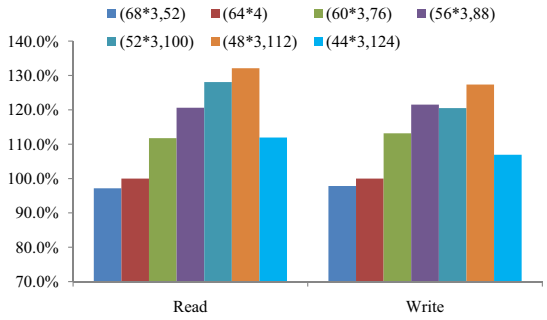
B. Results Analysis

First we conducted experiments to get the approximations of system parameters α and β. In order to get disk startup time for each request and data read/write rate, we used one of the file server to measure them. We performed experiments with different request sizes and repeated these tests thousands of times. We got the parameter values by calculating the average values. The approximate value of α was around 0.2 millisecond, and the β was around $\frac{1}{100MB/s}$. We then adopted these parameters to verify the effectiveness of the server-level adaptive layout strategy.

Next, we compared the I/O performance of the proposed data layout with existing fixed stripe size layout strategy. We conducted experiments with IOR benchmark, and we used



(a) Ethernet Environment



(b) InfiniBand Environment

Figure 5. Relative bandwidth of the IOR with different data layouts. Layout (68*3, 52) means the stripe sizes were 68 KB, 68 KB, 68 KB, and 52 KB for the four file servers respectively, and (64*4) means the stripe size was 64 KB for all file servers. Other layouts have the similar meanings. We regarded the performance of the existing layout with fixed stripe size(64KB) as baseline (100%), and compared the performance of all data layouts for both read and write.

4 nodes for PVFS2 file servers. During each run of the IOR benchmark, it first generates I/O request array before data access according to the input parameters, and then performs the requests in the array one by one. In our testing, we modified the codes of request generation. We preset the request sequence as a loop of data accesses: {192*1KB, 1*64KB} * LOOP_COUNT. This means in each loop, there are 192 requests of 1 KB size, and 1 request of 64 KB. The number of I/O clients was 16, which were deployed on 8 computing nodes. Figure 5 shows the experimental results. We used the block size 4 KB for data access analysis, and compared the I/O performance of all data layout manners with the fixed stripe size manner. From the results, we can see that the server-level adaptive layout can achieve I/O performance improvement for both data read and write. As shown in the figure, the optimal data layout is {48 KB, 48 KB, 48 KB, 112 KB}. The performance improvement is 26% for read and 19% for write in Ethernet environment, and 32% for read and 27% for write in InfiniBand environment. The performance improvement in Ethernet environment is not as high as that in InfiniBand environment, which is because

Table II
COMPARISON OF DIFFERENT ROUND SIZES AND BLOCK SIZES

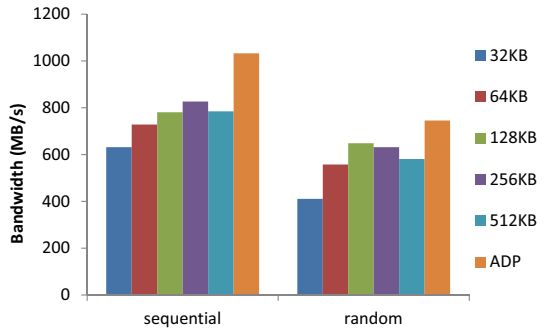
Round Size (KB)	Block Size (KB)	Optimal Layout (KB)	Performance Improvement
256	1	(49,49,49,109)	32.5%
	2	(50,50,50,106)	32.3%
	4	(48,48,48,112)	32.0%
	8	(48,48,48,112)	32.0%
512	1	(98,158,98,158)	18.2%
	2	(98,158,98,158)	18.2%
	4	(100,156,100,156)	18.1%
	8	(96,160,96,160)	17.9%

the Ethernet performance may limit the full potential of file servers in some cases.

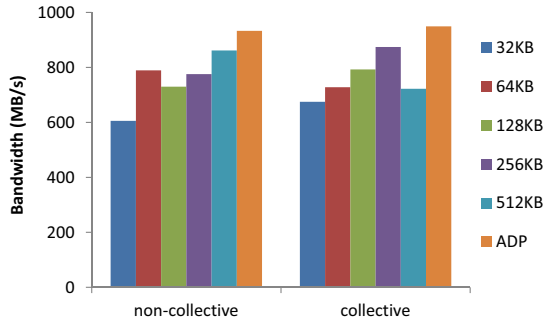
We then evaluated the effectiveness of different block sizes for the proposed server-level adaptive layout. Table II shows the experimental results in InfiniBand environment. We used the same data access pattern as in Figure 5. We changed the block sizes for cost analysis, and we measured the performance under two round sizes (round size is calculated using Equation 2), 256 KB and 512 KB. We compared the performance with fixed stripe size data layout, i.e., 64 KB * 4 for the first case and 128 KB * 4 for the second case. From the results, we can see that the block size slightly affects data block distribution in these cases. In addition, the smaller the size of data block used for cost analysis, the higher the I/O performance improvement that can be achieved. Generally, small data block size involves more calculation of cost analysis, and 4 KB block size is an ideal choice for lots of applications.

We also conducted a set of experiments to evaluate the proposed data layout with a synthetic workload with IOR and MPI-TILE-TILE instances. Figure 6 shows the results. For each benchmark, we ran multiple instances one by one with different runtime parameters to simulate non-uniform access patterns on different file servers. We compared the aggregate I/O bandwidth with different data layouts. This set of experiments were conducted in all 64 nodes, with each node configured as both file server and I/O client. For IOR tests, we measured I/O performance with sequential and random workloads respectively, while we measured aggregate bandwidth with collective I/O and non-collective I/O workloads in MPI-TILE-IO tests. We used 2 KB block size for cost analysis for the server-level adaptive layout. From the results we can see that the server-level adaptive data layout can achieve up to 81.5% performance improvement with IOR benchmark and 54.6% with MPI-TILE-IO benchmark.

Finally, we evaluated the proposed layout with a real application I/O trace, namely 'Anonymous LANL App 1' [26]. The data access patterns and cost analysis results are shown in Figure 2. We replayed the data accesses of this application according to the I/O trace, by simulating the same application scenario: 64 MPI-IO clients, and same I/O



(a) IOR benchmark



(b) MPI-TILE-IO benchmark

Figure 6. Performance of a synthetic workload with IOR and MPI-TILE-IO benchmarks. Label 'ADP' refers to the proposed adaptive data layout.

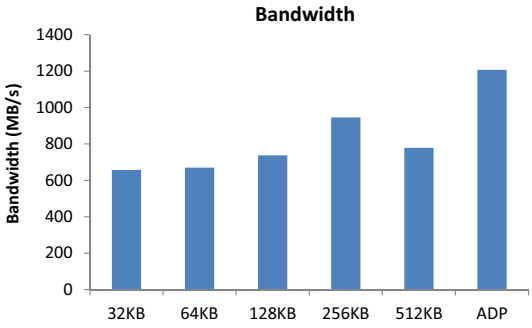


Figure 7. Performance of LANL App1 under different data layout configurations. Label 'ADP' means the proposed server-level adaptive data layout

requests for each I/O client. We conducted the experiments with 16-node InfiniBand environment, in which all 16 nodes were served as both file servers and I/O clients. We adopted different stripe sizes for different file servers, and then measured the overall I/O bandwidth and request service rate. In this set of experiments, we used 2 KB block size for cost analysis for the server-level adaptive layout. We compared the results of server-level adaptive data layout with existing uniform stripe size strategies. Figure 7 shows the experimental results. In this figure, we measured I/O

performance of 5 fix stripe size layout manners: 32 KB, 64 KB, 128 KB, 256 KB, and 512 KB. We also measured the I/O performance of the proposed data layout. From the result we can observe that the server-level adaptive data layout can achieve around 80.3% performance improvement compared to the data layouts with fix stripe size. The results indicate that it is beneficial to move some data from file servers with high costs to those with low access cost, to balance the data access workload. The proposed server-level adaptive data layout is an effective performance optimization strategy for applications with complicated data access patterns.

VI. Conclusions and Future Work

Request service rate and bandwidth are usually two I/O performance metrics in parallel I/O systems. Existing parallel file systems are designed for generic I/O access characteristics that treat all I/O requests being equally distributed in all file servers, ignoring the difference of I/O workloads on individual servers. Therefore, the peak I/O performance can rarely be achieved, especially for applications with non-uniform access patterns.

In this paper, we propose a server-level adaptive data layout, which adopts different stripe sizes for different file servers according to data access costs on them. We present how to calculate the access costs for data blocks, and how to distribute these blocks across multiple file servers according to the access costs by using an equal-depth histogram method. We also propose two approaches, offline approach and online approach, to put this server-level adaptive layout into use for data-intensive applications. Experimental results with the IOR and MPI-TILE-IO benchmarks demonstrate a significant improvement of I/O performance. The performance improvement was up to 81.5% for IOR workload and 54.6% for MPI-TILE-IO workload. We also evaluated the proposed data layout with a real application I/O trace, and the improvement of I/O performance was up to 80.3%. The server-level adaptive layout can achieve higher performance by exploiting the potential capability of all file servers. The essence of the proposed adaptive data layout is to let heavy-load servers offload some data accesses to light-load servers, in order to achieve balanced data access in all file servers. Therefore, it is suitable for large-scale and data-intensive applications with complex access patterns.

In future, we plan to extend the proposed data layout to distributed file systems and cloud environment, where the individual differences of data access patterns and file servers' capabilities may be even larger than parallel file systems.

VII. Acknowledgment

The authors are thankful to Samuel Lang and Dr. Robert Ross of Argonne National Laboratory for their constructive and thoughtful suggestions toward this work. This research was supported in part by National Science Foundation under NSF grant CCF-0621435 and CCF-0937877, and in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

[1] "High-performance Storage Architecture and Scalable Cluster File System," Lustre File System White Paper, December 2007.

[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] G. Gibson, B. Welch, G. Goodson, and P. Corbett, "Parallel NFS Requirements and Design Considerations," Internet Draft, October 2004.

[4] I. F. Haddad, "PVFS: A Parallel Virtual File System for Linux Clusters," *Linux Journal*, p. 5, 2000.

[5] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas Activescale Storage Cluster: Delivering Scalable High Bandwidth Storage," in *SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2004, p. 53.

[6] P. Lu and K. Shen, "Multi-layer Event Trace Analysis for Parallel I/O Performance Tuning," *International Conference on Parallel Processing*, vol. 0, p. 12, 2007.

[7] A. Konwinski, J. Bent, J. Nunez, and M. Quist, "Towards an I/O Tracing Framework Taxonomy," in *Proceedings of the 2nd International Workshop on Petascale Data Storage: held in conjunction with Supercomputing '07*, ser. PDSW '07. New York, NY, USA: ACM, 2007, pp. 56–62.

[8] 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.

[9] 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.

[10] 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.

[11] H. Song, Y. Yin, Y. Chen, and X.-H. Sun, "A Cost-intelligent Application-specific Data Layout Scheme for Parallel File Systems," in *Proceedings of the 20th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '11. 2011.

[12] E. Smirni and D. A. Reed, "Workload Characterization of Input/Output Intensive Parallel Applications," in *in Proceedings of the 9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer-Verlag, 1997, pp. 169–180.

[13] T. M. Madhyastha and D. A. Reed, "Learning to Classify Parallel Input/Output Access Patterns," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 802–813, 2002.

[14] F. Chen, D. A. Koufaty, and X. Zhang, "Survival of the Fittest: Making the Best Usage of SSDs in High-performance Storage Systems," in *Proceedings of the 25th ACM International Conference on Supercomputing*, ser. ICS '11. 2011.

[15] 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.

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

[17] H. Song, Y. Yin, X.-H. Sun, R. Thakur, and S. Lang, "A Segment-level Adaptive Data Layout Scheme for Improved Load Balance in Parallel File Systems," in *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, ser. CCGRID '11. 2011.

[18] D. Kotz and N. Nieuwejaar, "File-system Workload on a Scientific Multiprocessor," *IEEE Parallel and Distributed Technology*, vol. 3, pp. 51–60, March 1995.

[19] M. Seltzer, P. Chen, and J. Ousterhout, "Disk Scheduling Revisited," in *In Proceedings of the USENIX Winter Technical Conference (USENIX Winter 90*, 1990, pp. 313–324.

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

[21] 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.

[22] 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.

[23] Y. Wang and D. Kaeli, "Profile-guided I/O Partitioning," in *Proceedings of the 17th Annual International Conference on Supercomputing*, ser. ICS '03. New York, NY, USA: ACM, 2003, pp. 252–260.

[24] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A Scalable, High-performance Distributed File System," in *OSDI '06: Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, 2006, pp. 307–320.

[25] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, "PLFS: a Checkpoint Filesystem for Parallel Applications," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA: ACM, 2009, pp. 21:1–21:12.

[26] "Application I/O Traces: Anonymous LANL App1." [Online]. Available: http://institutes.lanl.gov/plfs/maps/

[27] P. B. Gibbons, Y. Matias, and V. Poosala, "Fast Incremental Maintenance of Approximate Histograms," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, ser. VLDB '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 466–475.

[28] S. W. Son, S. P. Muralidhara, O. Ozturk, M. Kandemir, I. Kolcu, and M. Karakoy, "Profiler and Compiler Assisted Adaptive I/O Prefetching for Shared Storage Caches," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 112–121.

[29] G. Yadgar, M. Factor, K. Li, and A. Schuster, "MC2: Multiple Clients on a Multilevel Cache," in *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, ser. ICDCS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 722–730.

[30] C. M. Patrick, M. Kandemir, M. Karaköy, S. W. Son, and A. Choudhary, "Cashing in on Hints for Better Prefetching and Caching in PVFS and MPI-IO," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 191–202.