HAS: Heterogeneity-Aware Selective Layout Scheme for Parallel File Systems on Hybrid Servers

Shuibing He, Xian-He Sun, Adnan Haider
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois, USA
{she11, sun}@iit.edu, ahaider3@hawk.iit.edu

Abstract—**Hybrid parallel file systems (PFS), consisting of multiple HDD and SSD I/O servers, provide a promising design for data intensive applications. The efficiency of a hybrid PFS relies on the file's data layout. However, most current layout strategies are designed and optimized for homogeneous servers. Using them directly in a hybrid PFS neither addresses the heterogeneity of servers nor the varying access patterns of applications, making hybrid PFSs disappointingly inefficient.**

In this paper, we propose HAS, a novel heterogeneity-aware selective data layout scheme for hybrid PFSs. HAS alleviates the inter-server load imbalance through skewing data distribution on heterogeneous servers based on their storage performance. To largely improve the entire system's I/O efficiency, HAS adaptively selects the optimal data layout from three typical candidates according to the application's data access patterns, based on a newly developed selection and distribution algorithm. We have implemented HAS within OrangeFS to provide efficient data distribution for data-intensive applications. Our extensive experiments validate that HAS significantly increases the I/O throughput of hybrid PFSs, compared to existing data layout optimization methods.

Keywords-**Parallel I/O System; Parallel File system; Data Layout; Solid State Drive**

I. Introduction

Parallel file systems (PFS) have been widely used in high performance computing (HPC) systems during the past few decades. A PFS, such as OrangeFS [1], Lustre [2] and GPFS [3], can achieve superior I/O bandwidth and large storage capacity by accessing multiple I/O servers simultaneously. However because of the existing performance gap between I/O servers and CPU, the so called I/O wall, current PFSs cannot fully meet the enormous data requirements of many HPC applications [4], especially for data intensive HPC applications.

Solid state disks (SSD) are attracting attention in HPC domains [5] due to their low access latency and high data bandwidth. However, building a sole SSD file system for a large computing cluster often is not a feasible choice, due to the cost of SSDs and the merits of HDDs, such as high capacity and decent peak bandwidth for large sequential requests. A hybrid parallel I/O system, which is comprised of both HDD servers (HServer) and SSD servers (SServer), is more practical for cost-constraint systems [6]. In practice, SServers can be used as a cache [7] or in a multi-tiered system [8]. These two approaches have their merits but cannot fully utilize the I/O parallelism of the servers. Detailed comparison of these architectures is out of the scope of this paper. In this work, we focus on high-capacity SServers and use them as flat storage.

To make full use of high-performance SServers, a PFS must rely on an efficient file data layout, which is an algorithm defining a file's data distribution across available servers. Traditional layout methods utilize fixed-size stripe to dispatch files across multiple servers. Generally, data layout depends on application behaviors. To further improve the storage performance, numerous works are devoted to the file data layout optimizations, such as data reorganization [9], data partition [10], [11], data replication [12], [13], and data stripe resizing [14]. However, most current schemes are designed and optimized for homogeneous servers. When applied to hybrid PFSs, the homogeneity assumptions do not hold, thus providing two limitations.

First, the heterogeneity of storage servers can significantly decrease the overall system performance if the distribution scheme is not conscious of server performance. Due to their intrinsic properties, SServers always outperform HServers. In PFSs, a large file request is commonly divided into multiple sub-requests, which will be concurrently served by multiple HServers and SServers. In this case, SServers are left idling while HServers continue to process their requests. This inter-server load imbalance leads to under-utilization of system hardware resources. Our experimental results show that load-imbalance significantly slows down a request (Section II).

Second, most previous works are only designed for a particular or limited set of access patterns. For example, the commonly used *simple stripe* policy in OrangeFS [1] is only suitable for large parallel file requests, but performs poorly for other access patterns. As applications become data-intensive and complicated, access patterns can vary considerably between applications, in terms of request size, access type (read or write), and access concurrency. A data layout optimized for only a specific access pattern is not efficient for all applications. An efficient, comprehensive data layout scheme should depend on application's access patterns.

In this paper, we propose a heterogeneity-aware selective (HAS) data layout scheme for hybrid PFSs to address the above challenges. HAS eliminates load-imbalance by assigning varied-size file stripes or distributing different number of files to heterogeneous servers based on their performance.

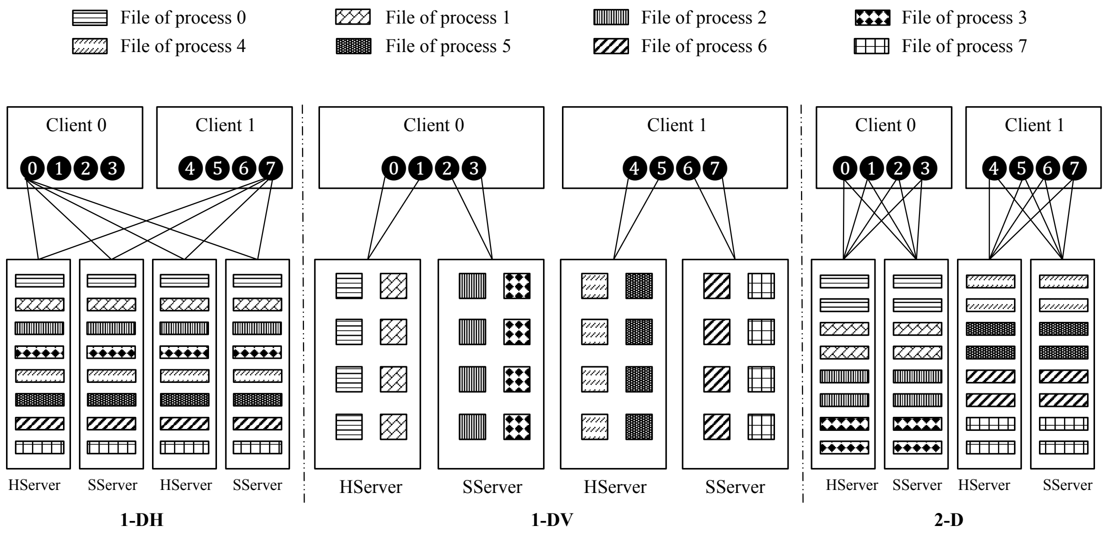


Fig. 1. The three typical data layout policies in PFSs. For 1-DH and 2-D, HServer and SServer are assigned with fixed-size file stripes. For 1-DV, HServer and SServer are distributed with identical number of files.

In addition, to obtain the global optimal performance, HAS selects, based on application characteristics, the data layout policy with the minimal access cost from three typical layout candidates existing in PFSs as the final data layout method. To be specific, HAS applies an analytical model which considers both an application's access patterns and a server's storage characteristics to achieve the optimal data layout.

One might expect to determine the proper layout mechanism for a given application pattern to be simple. In reality, it is a complex issue for several reasons. First, the performance of each server can be significantly impacted by application access patterns, such as request size, access type (read or write), access concurrency (number of processes), etc. Second, the server performance is also related with the storage device. Even under the same pattern, HServer and SServer exhibit different performance behaviors. Finally, besides the storage cost, the network cost, affected by application access patterns, is also an integral part of the overall data access cost. HAS fully considers these critical factors when determining the appropriate data layout to achieve optimal performance in hybrid PFSs.

Specifically, we make the following contributions.

- We introduce a cost model, which is a function of system configurations and application I/O patterns, to evaluate the I/O completion time of each file request for each of the three different layout policies in hybrid PFSs.
- We propose a selective data layout scheme, which distributes data using the least expensive layout policy determined by the cost analysis. The distribution is implemented either by varying the file stripe sizes or the number of files on different servers.
- We implement the prototype of the HAS scheme under OrangeFS, and have conducted extensive tests to verify the benefits of the HAS scheme. Experiment results illustrate that HAS can significantly improves I/O performance.

The rest of this paper is organized as follows. The background and motivation are given in Section II. We describe the design and implementation of HAS in Section III. Performance evaluations of HAS are presented in Section IV. We introduce the related work in Section V, and conclude the paper in Section VI.

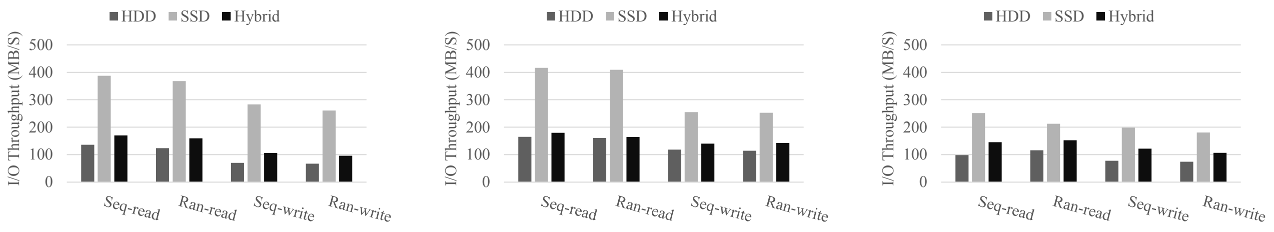
II. Background and Motivation

A. Typical Data Layout Policies in PFSs

PFSs, such as OrangeFS [1], Lustre [2] and GPFS [3], support three typical data layout policies — one-dimensional horizontal (*1-DH*), one-dimensional vertical (*1-DV*), and two-dimensional (*2-D*) layout. As shown in Figure 1, *1-DH* is the simple striping method that distributes a process's file across all available servers in a round-robin fashion. *1-DV* performs no striping at all, and instead places the file data on one server. *2-D* is a hybrid method, it distributes the file on a subset of servers. All three layout policies utilize fixed-size file stripes to distribute file data, and each of them work well for a particular kind of I/O access patterns [12]. However, these schemes are designed for homogeneous PFSs on identical storage servers, and they will perform poorly for hybrid PFSs.

B. Motivation Example

To illustrate the impact of server heterogeneity on the overall system performance, we ran IOR [15] to access parallel OrangeFS files in a hybrid environment (denoted by Hybrid) with four HServers and four SServers. IOR ran with 16 processes, each of which accessed an individual file. We limited the request size to 512KB, and the access pattern to sequential and random read and write. For the purpose of performance comparison, we also ran IOR with the same parameters on two homogeneous sub-clusters: four HServers (denoted by HDD) and four SServers (denoted by SSD). Figure 2 shows



(a) 1-DH layout (b) 1-DV layout (c) 2-D layout

Fig. 2. Throughputs of IOR with three typical data layout schemes. The system is tested with homogeneous and heterogeneous server configurations, the file stripe is 64KB, and the group size is two in 2-D layout.

the throughputs of IOR with the three typical layout policies. We can observe that for all policies, the hybrid cluster with eight servers slightly outperforms the homogeneous cluster with four low-speed HServers, and performs worse than the homogeneous system with four SServers. In other words, more heterogeneous hardware resources actually degrade I/O performance. This illustrates that traditional layout schemes are highly inefficient for hybrid PFSs.

C. Reasons for Poor Performance of Hybrid PFSs

1) Inter-Server Load Imbalance: Traditional layout policies will lead to severe load imbalance among heterogeneous servers. Generally, a large file request will be divided into multiple sub-requests, concurrently served by the underlying servers. With fixed-size file stripe, HServers and SServers possibly handle same-size sub-requests. As a result, SServers will waste much time on waiting for the low-speed HServers. To show this, we analyze the I/O time of each server in the sequential read test in Figure 2(a). We find that HServers take roughly 3.5X time to complete I/O operations compared with SServers (Other layouts have the similar results).

2) Single Server I/O Inefficiency: Existing layout polices may also incur severe I/O inefficiency on a single server. Similar to the cases in homogeneous systems, each layout scheme in hybrid PFSs leads to distinct sub-requests and access concurrency on each server. For example, *1-DH* produces much higher I/O concurrency than *1-DV* when a large number of processes exist. The server performance can be largely degraded due to sever I/O contention. In other words, although the inter-server load balance is maintained, a layout policy not considering application access patterns can offset the overall system performance.

III. Design and Implementation

In this section, we first introduce the basic idea of our proposed heterogeneity-aware selective (HAS) data layout scheme. Then we describe the cost model and algorithm used to determine the optimal data layout for hybrid PFSs. Finally, we present the implementation of HAS.

A. The Basic Idea of HAS Overview

Since traditional layout schemes lead to severe performance degradation, the proposed data layout scheme, HAS, aims to optimize the performance of hybrid PFSs through skewing data distribution on heterogeneous servers.

Figure 3 shows the idea of HAS. Compared with traditional layout policies which assign each server with fixed-size file stripes or fixed-number of files, HAS distributes file data on heterogeneous servers with varied-size file stripes or various number of files based on the server performance. This can alleviate inter-server load imbalance. Further more, HAS chooses the layout scheme with minimal access cost from the three layout candidates as the final layout for an application to improve the overall system I/O efficiency.

However, determining the proper layout policy and its corresponding stripe sizes or number of files on each server is not easy for several reasons. First, the server performance is a complex function of multiple factors, such as application access patterns and storage media. Second, in addition to the storage cost, the network cost also plays an important role in the overall data access cost. To identify the optimal data layout, we built an analytical cost model accounting for application, network, and storage characteristics to evaluate the data access time in a heterogeneous I/O environment.

B. Data Access Model in Hybrid PFSs

1) Assumptions and Definitions: The overall I/O time is a function of various parameters, which are listed in table I. The *system* and *application pattern* parameters are used as known inputs, and *data layout* parameters are optimized depending on the inputs.

Note that the storage related parameters show distinct characteristics on heterogeneous servers. First, the start up time of SServer is much smaller than HServer's. Second, data transfer time of SServer is several times smaller than that of HServer's. Finally, while HServer may have identical read and write performance, SServer usually has a faster read performance than writes because write operations lead to many background activities due to garbage collection and wear leveling [16].

We only consider *1-DH*, *1-DV*, and *2-D* layout policies due to their popularity. In each policy, the files are distributed on the underlying servers as in Figure 3. We assume each process accesses one file and only that file. For *1-DH* and *2-D*, stripe sizes are varied depending on the type of server. For *1-DV*, since all file data is distributed on one server, varying the stripe size will not affect the request cost so we vary the number of files on heterogeneous servers instead. Due to symmetry, we assume perfect load balance of data access within HServers and SServers but not between different types of servers. In

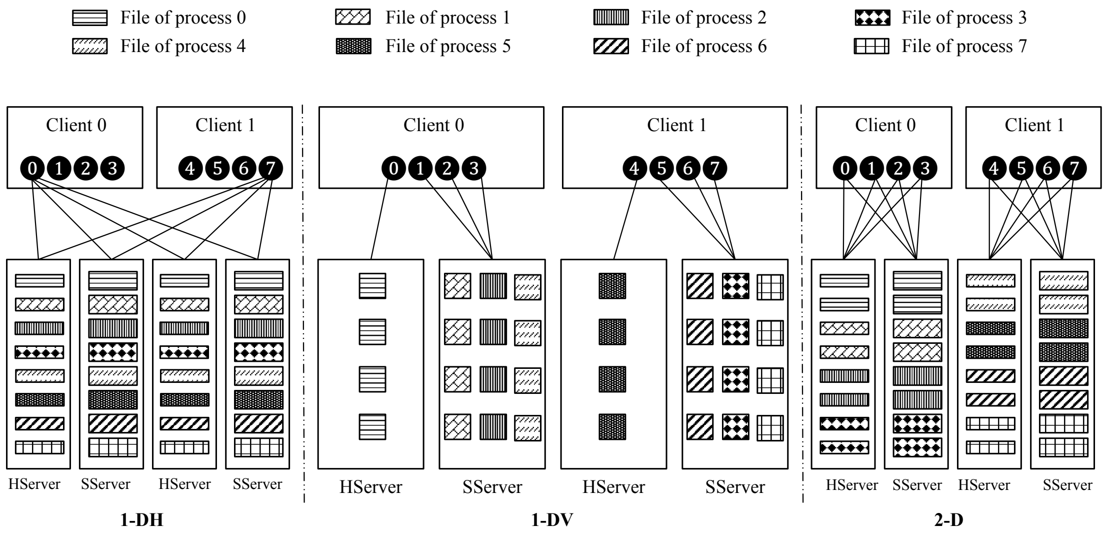


Fig. 3. The three data layout policies in hybrid PFSs after optimization. For 1-DH and 2-D, HServer and SServer are assigned with varied-size file stripes. For 1-DV, HServer and SServer are distributed different number of files.

TABLE I
PARAMETERS IN COST ANALYSIS MODEL

System Parameters	
m	Number of HServers
n	Number of SServers
c	Number of process on one client node
e	Average network establishing time per connection
t	Unit data network transmission time
α_h	Average startup time of one operation on HServer
β_h	Unit data transfer time on HServer
α_{sr}	Average startup time for read on SServer
β_{sr}	Unit data transfer time for read on SServer
α_{sw}	Average startup time for write on SServer
β_{sw}	Unit data transfer time for write on SServer
Application Pattern Parameters	
p	Number of client processes
r	Size of file request
o	Type of file request (read or write)
Data Layout Parameters	
s_h	Stripe size on HServer in 1-DH and 2-D layout
s_s	Stripe size on SServer in 1-DH and 2-D layout
p_h	Number of process on HServer in 1-DV layout
p_s	Number of process on SServer in 1-DV layout
g	Number of storage groups in 2-D layout

addition, we have following assumptions for each policy:

- For *1-DH*, we assume each file request is served by all the *m*+*n* servers, so that each storage node can contribute to the aggregate I/O performance. Each sub-request on the server has the same size with the stripe on that server. Thus we have the following constraint.

$$m \times s_h + n \times s_s = r \tag{1}$$

  Usually s_s is larger than s_h to achieve load balance. In an extreme case, s_h can be zero (which means file data are only distributed on SServers) if there is a possibility to improve the performance.
- For *1-DV*, the number of processes on each server equals the number of files. We assume all files are distributed on the $m + n$ servers, thus

$$m \times p_h + n \times p_s = p \tag{2}$$

  Similarly, p_s is larger than p_h, and p_h can be zero.
- For *2-D*, we assume each group includes m/g HServers and n/g SServers, and a file request is distributed on all the $(m + n)/g$ servers in that group as in *1-DH* policy, thus

$$m \times s_h + n \times s_s = g \times r \tag{3}$$

2) Access Cost Analysis: Our cost model divides the overall I/O time of a parallel data access into four parts. T_E is the network establishing time, T_X is the network transferring time, T_S is the storage startup time, and T_T is the storage transfer time. The former two parts are network related access costs (T_{NET}), and the latter two are storage related access costs (T_{STOR}). Taking their sum gives the overall I/O time (T):

$$T = T_E + T_X + T_S + T_T \tag{4}$$

Establishing cost: T_E depends on the number of establishing operations for the parallel data accesses. Since a network

Cost	**Condition**		**Network cost** T_{NET}		**Storage cost** T_{STOR}
			Establish T_E	Transfer T_X	$Max\{$Start T_{SH}+ I/O T_{TH}, Start T_{SS}+I/O $T_{TS}\}$
$\boldsymbol{T_{1\text{-}DH}}$	$p \le c(m+n)$		$c(m+n)e$	$\max\{crt, ps_st\}$	$p * \max\{(\alpha_h + s_h\beta_h), (\alpha_{sr} + s_s\beta_{sr})\}$
	$p > c(m+n)$		pe	$\max\{crt, ps_st\}$	$p * \max\{(\alpha_h + s_h\beta_h), (\alpha_{sr} + s_s\beta_{sr})\}$
$\boldsymbol{T_{1\text{-}DV}}$	$c \le p_s$		p_se	p_srt	$\max\{p_h(\alpha_h + r\beta_h), p_s(\alpha_{sr} + r\beta_{sr})\}$
	$c > p_s$		ce	crt	$\max\{p_h(\alpha_h + r\beta_h), p_s(\alpha_{sr} + r\beta_{sr})\}$
$\boldsymbol{T_{2\text{-}D}}$	$p \le g$		$c\left\lceil \frac{p}{g} \right\rceil e$	crt	$\left\lceil \frac{p}{g} \right\rceil * \max\{(\alpha_h + s_h\beta_h), (\alpha_{sr} + s_s\beta_{sr})\}$
	$p > g$	$\left\lceil \frac{p}{g} \right\rceil \le c\left\lceil \frac{n}{g} \right\rceil$	$c\left\lceil \frac{n}{g} \right\rceil e$	$\max\{crt, \left\lceil \frac{p}{g} \right\rceil s_st\}$	$\left\lceil \frac{p}{g} \right\rceil * \max\{(\alpha_h + s_h\beta_h), (\alpha_{sr} + s_s\beta_{sr})\}$
		$\left\lceil \frac{p}{g} \right\rceil > c\left\lceil \frac{n}{g} \right\rceil$	$\left\lceil \frac{p}{g} \right\rceil e$	$\max\{crt, \left\lceil \frac{p}{g} \right\rceil s_st\}$	$\left\lceil \frac{p}{g} \right\rceil * \max\{(\alpha_h + s_h\beta_h), (\alpha_{sr} + s_s\beta_{sr})\}$

Fig. 4. Data access cost for read requests on hybrid Hservers and SServers

establishing operation is related with both the client and the server, T_E is determined by the higher cost of the two. Take the *1-DH* layout as an example, each client needs to establish network connections with all servers serially, thus $T_E = c(m+n)e$. From a server's point of view, it is accessed by p processes, thus $T_E = pe$. Then, the final establishing time $T_E = max\{c(m+n)e, pe\}$.

Transfer cost: T_X is related with the network data transfer size and the network data transfer rate. Similarly, it is determined by the maximal cost of a client and a server. We still use the *1-DH* layout as an example. For a client, $T_X = crt$; for HServer, $T_X = ps_ht$; for SServer, $T_X = ps_st$. Since s_h will always be at most s_s, $T_X = max\{crt, ps_st\}$.

Startup cost: T_S is relatively straightforward and determined by the number of I/O operations on one server, namely the number of client processes assigned on that server. For a parallel request, T_S is determined by the maximal cost among all involved I/O servers.

Read/write cost: T_T is the time spent on actual data read/write operations. It can be calculated by the ratio of the request size over the data transfer rate of storage devices. Similarly, T_T is determined by the maximal value of all I/O servers for a parallel request.

We refer to the data access cost in the three layout polices as T_{1-DH}, T_{1-DV} and T_{2-D} respectively, which is calculated as in Figure 4. T_{1-DH} is derived from our previous work [17], which expresses the cost as a function of s_h and s_s. For the new proposed formulas, T_{1-DV} describes the cost as a function of p_h and p_s, and T_{2-D} utilizes s_h, s_s and g for the same goal. Figure 4 only displays the access cost for read requests; writes will be similar except startup and unit data transfer time for SServers will change. These three policies imply more application-aware and effective layout optimization methods for hybrid PFSs.

By examining the formulas, we can capture the following implications for data layout optimizations.

- For *1-DH* and 2-D, the storage read/write cost T_{TH} and T_{TS} on the two types of servers can be balanced by increasing the stripe size of SServer (s_s) and decreasing that of HServer (s_h), but doing so may increase the network transfer time T_X on SServer, possibly delaying the overall completion time (T).
- For *1-DV*, the storage read/write cost T_{TH} and T_{TS} can be balanced by increasing the number of files on SServer (p_s) and decreasing that on HServer (p_h). Similarly, this may increase the network transfer time T_X, offsetting the reduction of T.

C. Optimized Data Layout Determination

We note that the model consists of linear equalities and inequalities of unknown variables (max can be expressed as multiple linear inequalities). Therefore, the model can be solved exactly to minimize the total I/O cost under three layout policies, subject to the above constraints. Since we have described the stripe size determination for the *1-DH* layout policy [17], we show the methods to determine the optimal data layout for the other two policies.

1) 1-DV Layout Optimization: The system and pattern related parameters can be regarded as constants, and T_{1-DV} is a function of two unknowns—p_h and p_s. The final problem is to choose the values of p_h and p_s to minimize T_{1-DV}. For condition $c \le p_s$, according to the member values in the corresponding maximum expressions in the formulas in Figure 4, we translate this optimization problem into two linear programming (LP) problems shown below.

$$\text{Case 1: Minimize} \qquad T^1_{1-DV} = p_s(e + rt) + p_h(\alpha_h + r\beta_h) \tag{5}$$

$$\text{subject to} \begin{cases} mp_h + np_s & = p \\ p_s(\alpha_{sr} + r\beta_{sr}) & \leq p_h(\alpha_h + r\beta_h) \\ 0 & \leq p_h \leq p/m \\ c & \leq p_s \leq p/n \end{cases} \tag{6}$$

Case 2: Minimize $T^2_{1-DV} = p_s(e+rt)+p_s(\alpha_{sr}+r\beta_{sr})$ (7)

$$\text{subject to} \begin{cases} mp_h + np_s & = p \\ p_h(\alpha_h + r\beta_h) & \leq p_s(\alpha_{sr} + r\beta_{sr}) \\ 0 & \leq p_h \leq p/m \\ c & \leq p_s \leq p/n \end{cases} \tag{8}$$

Then

$$T_{1-DV} = min\{T^1_{1-DV}, T^2_{1-DV}\} \tag{9}$$

The first constraint of Equation 6 and 8 is Equation 2. The second constraint is the only difference between case 1 and case 2; they account for the two possible values of T_{STOR} in Figure 4. The third and fourth constraints for *1-DV* are directly derived from Equation 2. For example, the max of p_h is achieved by letting $p_s = 0$ in Equation 2, as goes for calculating the max of p_s. The final cost for *1-DV* determined by Equation 9 is the minimum of the two cases.

The cases for the alternate condition will be similar to the two above, except some values will be interchanged according to the formulas in Figure 4.

2) 2-D Layout Optimization: T_{2-D} is a function of three unknown parameters s_h, s_s, and g. Similarly, based on the member values in the maximum expressions in Figure 4, we translate the optimization problem into two linear programming (LP) problems for the condition $p \leq g$.

Case 1: Minimize $T^1_{2-D} = c((p/g)e+rt)+(p/g)(\alpha_{sr}+s_s\beta_{sr})$ (10)

$$\text{subject to} \begin{cases} ms_h + ns_s & = gr \\ 1 < & g < (m+n) \\ \alpha_h + s_h\beta_h & \leq \alpha_{sr} + s_s\beta_{sr} \\ 0 & \leq s_h \\ 0 & \leq s_s \end{cases} \tag{11}$$

Case 2: Minimize $T^2_{2-D} = c((p/g)e+rt)+(p/g)(\alpha_h+s_h\beta_h)$ (12)

$$\text{subject to} \begin{cases} ms_h + ns_s & = gr \\ 1 < & g < (m+n) \\ \alpha_{sr} + s_s\beta_{sr} & \leq \alpha_h + s_h\beta_h \\ 0 & \leq s_h \\ 0 & \leq s_s \end{cases} \tag{13}$$

Then

$$T_{2-D} = min\{T^1_{2-D}, T^2_{2-D}\} \tag{14}$$

The first constraint is Equation 3. The second constraint ensures the achieved layout remains *2-D*. Similar to the second constraint of Equation 6, the third constraint accounts for the possible values of T_{STOR}. The last two constraints for Equation 10 and 12 simply ensure the stripe sizes remain positive. The cases for other conditions will be similar to the two above. Note the above optimizations use the storage values for read, the optimizations for write can be done similarly.

3) Gobal Data Layout Optimization: After the linear optimizations, there will be three potential layout methods, each with their own cost T_{1-DH} , T_{1-DV}, and T_{2-D}. Then, HAS will select the minimum of three candidates as the most optimal data distribution, which accounts for application, storage, and network characteristics.

Note that the linear program is expressed with two or three unknown variables, the search space is very small and solving the program requires acceptable time cost.

D. Implementation

We implemented HAS in OrangeFS [1], which is a popular parallel file system and directly provides the varied-size striping method for each file server. The procedure of the HAS scheme includes the following three phases, as shown in Figure 5.

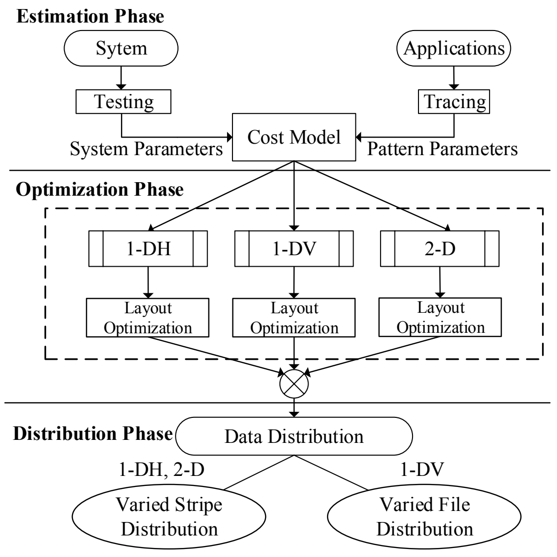


Fig. 5. HAS data layout optimization procedure.

The *estimation* phase consists of two parts, system testing and application tracing. For system testing, the network parameters, e and t, the storage parameters, α_h,β_h,$\alpha_{sr/w}$,$\beta_{sr/w}$, and the system parameters, such as m and n can be regarded as constants. We use all file servers in the parallel file system to test the storage parameters for HServers and SServers with sequential/random and read/write patterns and then we calculate the average for HServers and SServers. We use many pairs of clients and file servers to estimate the network parameters. Again the tests are conducted thousands of times for the purpose of accuracy, and we use the average value for the network parameters. For application tracing, we use a trace collector, IOSIG [18], to obtain the run-time statistics of data accesses during the application's first execution. Many HPC applications access their files with predictable access patterns and they often run multiple times [19]. This provides an opportunity to achieve the proposed data layout scheme. Based on the I/O trace, we obtain the application's I/O pattern related parameters, such as p, r, and o.

In the *optimization* phase, using the parameters obtained in the *estimation* phase, we apply the cost model and linear programming optimization methods in III-C to determine the optimal file data distribution on HServers and SServers for each of the three layout policies. Since each policy may only give sub-optimal performance because of unique characteristics of applications, HAS compares their performance and chooses the one with minimal cost as the final data layout policy.

In the *distribution* phase, we distribute the file data with the optimal layout policy and the corresponding layout parameters for later runs of the applications. For *1-DH* and *2-D*, we utilize the APIs supported by OrangeFS to implement the specific variable stripe distribution and group distribution. In OrangeFS, parallel files can either be accessed by the PVFS2 or the POSIX interface. For PVFS2 interface, we utilize the "pvfs2-xattr" command to set the data distribution policy and the related layout parameters for directories where the application files are located. For POSIX interface, we use the "setfattr" command to reach the similar data layout optimization goal. For *1-DV* policy, we create different numbers of process files on HServers and SServers.

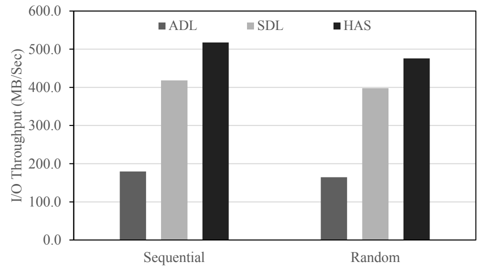
E. Discussion

HAS can potentially lead to more storage space consumption for SServers, which might be an unwanted feature by users. Fortunately, most file systems do not make full use of the storage space in the underlying devices. On the other hand, HAS focuses on high-capacity SServers thus the issue of lack space is not frequently encountered. In the worst case, with the possibility of an SServer running out of its space, we design a data migration method to balance the storage space by moving data from SServers to HServers, so that the available remaining space on SServers can be guaranteed for new incoming requests. This problem can also be addressed by using the hybrid PFS to store performance-critical data and the PFS only on HServers to store the rest of the data.

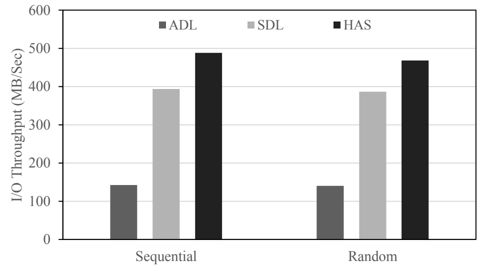
IV. Performance Evaluation

A. Experimental Setup

We conducted the experiments on a 65-node SUN Fire Linux cluster, where each node has two AMD Opteron(tm) processors, 8GB memory and a 250GB HDD. 16 nodes are equipped with additional OCZ-REVODRIVE 100GB SSD. All nodes are equipped with Gigabit Ethernet interconnection. The operating system is Ubuntu 9.04, and the parallel file system is OrangeFS 2.8.6. Among the available nodes, we select eight as client computing nodes, eight as HServers, and eight as SServers. By default, the hybrid OrangeFS file system is built on four HServers and four SServers.



(a) Read throughput



(b) Write throughput

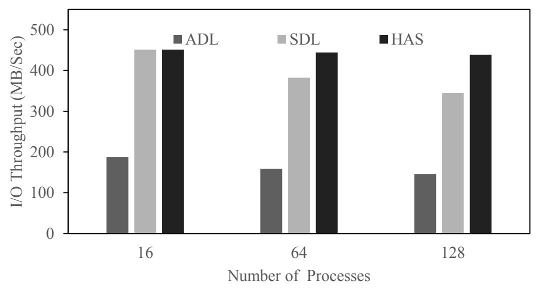
Fig. 6. Throughputs of IOR under different layout schemes with different I/O modes

We compare the proposed layout scheme HAS with two other data layout schemes: the application-aware scheme (ADL) [12] and the storage-aware scheme(SDL) [17]. In ADL, file data is placed across the hybrid file servers with one of the three policies according to the application's access pattern, but each server is assigned a fixed-size file stripe. In SDL, the file stripe sizes for the hybrid servers are determined by the server performance but only *1-DH* policy is chosen, without fully considering application access patterns. We use the popular benchmark IOR [15] and BTIO [20] to test the performance.

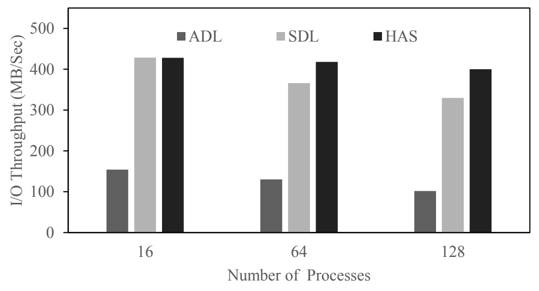
B. IOR Benchmarks

IOR provides three APIs, MPI-IO, POSIX, and HDF5. We only use the MPI-IO interface. Unless otherwise specified, IOR runs with 32 processes, each of which performs I/O operations on an individual 256MB parallel file with request size of 512KB. We provide two sets of experiments, varying application characteristics and varying storage characteristics. We illustrate the importance of considering both application and storage characteristics, by comparing to schemes which only consider one type of characteristics.

1) Varying Application Characteristics: We vary the following application related traits: I/O operation type, number of processes, and request size. First we run IOR with sequential and random read and write I/O operations. Figure 6 shows the throughput of IOR. We observe that HAS outperforms ADL and SDL. By using the optimal data distribution for HServers and SServers, HAS improves read performance up to 189.7% over ADL with all I/O access patterns, and write performance up to 242.7%. Compared with SDL, HAS improves the performance up to 23.8% for reads and 21.1% for writes.

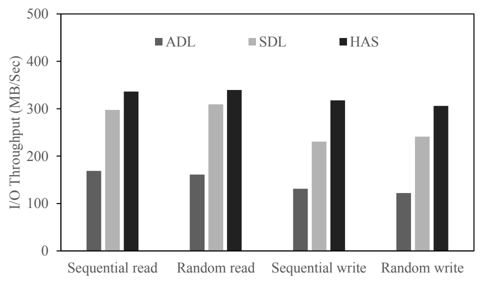


(a) Read throughput

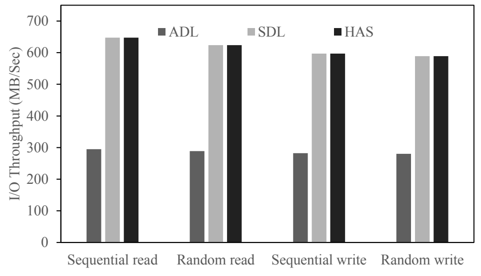


(b) Write throughput

Fig. 7. Throughputs of IOR with varied number of processes



(a) Request size is 128K



(b) Request size is 1024K

Fig. 8. Throughputs of IOR with varied request sizes

Although ADL accounts for I/O operation type variation, HAS has superior performance than ADL because it considers storage server performance differences. HAS provides optimal performance for read and write operations, but SDL degrades in performance because its lack of application awareness.

Then we evaluated the layout schemes with different number of processes. The IOR benchmark is executed under the random access mode with 16, 64 and 128 processes. As displayed in Figure 7, the result is similar to the previous test. HAS has the best performance among the three schemes. Compared with ADL, HAS improves the read performance by 140.3%, 179.7%, and 200.2% respectively with 16, 64 and 128 processes, and write performance by 174.3%, 200.7%, and 292.7%. Compared with SDL, HAS achieves similar performance for 16 processes. For 64 and 128, HAS improves read performance by 16.1% and 27.3% respectively, and write performance by 14.2% and 21.3%. When the number of processes is large, the *1-DV* policy, implemented in HAS, provides better performance than the *1-DH* layout used by SDL. As the number of processes increase, the performance of the hybrid PFS decreases because more processes lead to server I/O contention in HServers and SServers. These results show that HAS scales excellently with the number of I/O processes.

Finally, the I/O performance is examined with different request sizes. We set the request size to 128KB and 4096KB, and the number of processes to 32. From Figure 8(a), we can observe that HAS can improve the read performance up to 110.3%, and write up to 151.6% in comparison with ADL. Compared with SDL, HAS also has better performance: the read performance is increased up to 13.4%, and write performance is increased up to 37.7%. As the request size increases, *1-DH* tends to be the best layout policy. For example when the request size is 4096KB, HAS selects the same data layout policy as SDL, *1-DH*. These results validate that HAS can choose appropriate data distribution for HServers and SServers when the request size varies.

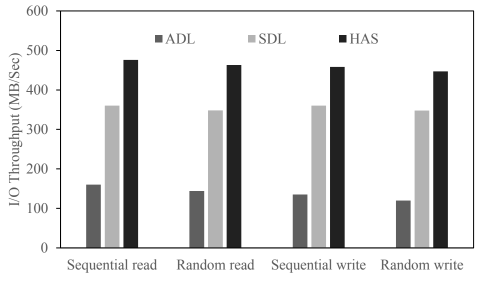
Our results validate that it is essential for a data layout scheme to account for application traits in order for a system to provide peak storage performance.

2) Varying System Characteristics: We examined the I/O performance with different server configurations. We varied the numbers of HServers and SServers with the ratios of 5:3 and 6:2. Figure 9 shows the bandwidth of IOR with different file server configurations. Based on the results, HAS can improve I/O throughput for both read and write operations. When the ratio is 5:3, HAS improves the read and write performance by up to 171.6% and 232.4% respectively, when compared to ADL. Compared with SDL, HAS increases the read performance by 21.9%, and write performance by 17.1%. When the ratio is 6:2, the performance gap is decreased because the server configuration is more homogeneous. In the experiments, the read and write performance disparity between HAS and ADL enlarges as the number of SServers increase because HAS is storage device conscious.

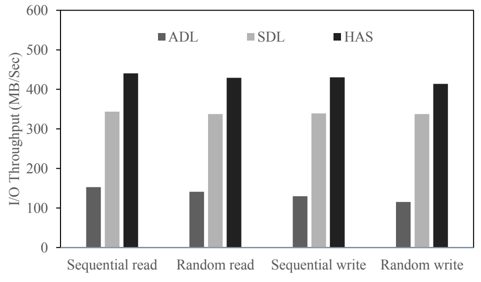
By varying the system characteristics, we prove that the consideration of system traits is essential to optimal data distribution.

C. BTIO benchmark

Apart from the synthetic benchmark above, we also use the BTIO benchmark [20] from the NAS Parallel Benchmark (NPB3.3.1) suite to evaluate the proposed scheme. BTIO represents a typical scientific application with interleaved intensive computation and I/O phases. BTIO uses a



(a) 5HServers : 3SServers



(b) 6HServers : 2SServers

Fig. 9. Throughputs of IOR with varied file server configurations

Block-Tridiagonal (BT) partitioning pattern to solve the three-dimensional compressible Navier-Stokes equations.

We consider the Class *B* and *epio* subtype BTIO workload in the experiments. That is, we write and read a total size of 1.69GB data. We use 16, 36, and 64 compute processes since BTIO requires a square number of processes. Each process accesses its own independent file. Output files are distributed across six HServers and two SServers on the hybrid OrangeFS file system.

As shown in Figure 10, compared to ADL and SDL, HAS achieves better throughput and scalability. Compared to ADL, HAS improves the performance by 153.1%, 157.6%, and 175.2% with 16, 36, 64 processes, respectively. For SDL, HAS achieves the improvement by up to 48.2%.

All the experiment results have confirmed that the proposed HAS scheme is a promising method to improve the performance of hybrid PFSs. It helps parallel file systems provide high performance I/O service to meet the growing data demands of many HPC applications.

V. Related work

A. I/O Request Stream Optimization

Much research has focused on reorganizing I/O request streams to minimize access time spent on I/O device and network. Generally, such optimizations are implemented at the I/O middleware layer. For example, instead of accessing multiple small, noncontiguous requests, data sieving [21] applies the strategy of accessing a contiguous chunk created by gathering the noncontiguous requests. Datatype I/O [22] and List I/O techniques [23] allow noncontiguous I/O requests to be converted into a single I/O request, thereby limiting the number of total requests. Collective I/O [21] also rearranges I/O accesses into a larger contiguous request, but considers the multiprocess level instead of a single process.

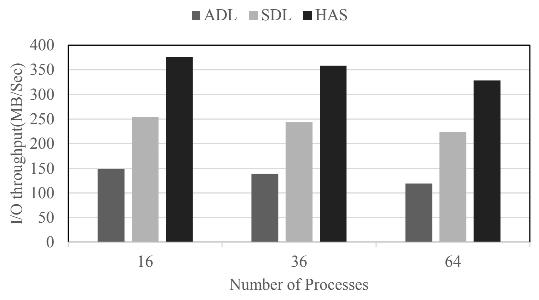


Fig. 10. Throughputs of BTIO under different data placement schemes

B. Data Layout in HDD-based File Systems

Parallel file systems have different data layout strategies, which allow for numerous data layout optimization methods [12]. Several techniques, including data partition [10], [11], data migration [24], and data replication [12], [13], [25], are applied to optimize data layouts depending on I/O workloads. Segment-level layout scheme logically divides a file to several parts and appoints an optimal stripe size for each part [26]. Another methodology, server-level adaptive layout strategy, selects different stripe sizes depending upon the type of the file server [14]. PARLO is designed for accelerating queries on scientific datasets by applying user specified optimizations [9]. Tantisiriroj et. al [27] uses HDFS-specific layout optimizations [28] to improve the performance of PVFS. However, all these works are designed for homogeneous HDD-based file systems, and can't be applied to heterogeneous environments.

C. Data Layout in SSD-based File Systems

SSDs are commonly integrated into parallel file systems due to their performance benefits. For now, most SSDs are used as a cache to HDDs, e.g. Sievestore [29], iTransformer [30], and iBridge [31]. Hybrid storage with SSDs is another popular technique to exploit their merits, including I-CASH [32] and Hystor [33]. Although effective, the vast majority of research is focused on a single file server. Data layout optimizations in Hybrid PFSs have not received their needed attention. CARL [8], our previous work, selects and situates file regions with high access costs onto SSD-based file servers at the I/O middleware layer, but the region cannot be placed onto both SSDs and HDDs. PADP [17] and PSA [34] employ stripe size variation to improve the performance of hybrid PFSs, yet the schemes are only optimized for the one-horizontal *(1-DH)* layout policy, without fully considering the application's access patterns. This work achieves an optimal data layout accounting for both application access patterns and server performance by choosing the least expensive layout under three typical layout policies.

VI. Conclusions

In this study, we propose a heterogeneity-aware selective (HAS) data layout scheme for parallel file systems with both

HDD and SSD-based servers. HAS alleviates the inter-server load imbalance by varying the file stripe sizes or the number of files on different servers based on their storage performance. In addition, HAS selects the optimal data layout from three types of candidates according to application access patterns to further improve I/O efficiency. We introduce a data access cost model and a linear programming optimization method to determine the appropriate data layout method for given application patterns and system traits. In principle, HAS improves hybrid parallel file system performance by matching data layout with both application characteristics and storage capabilities. We have developed and presented the proposed HAS data layout optimization scheme in OrangeFS. Experimental results show that HAS improves the I/O performance by up to 292.7% over the existing file data layout schemes. In the future, we plan to propose more intelligent data layout schemes for applications with complex access patterns.

Acknowledgment

The authors are thankful to the anonymous reviewers and the members of the SCS research group for their valuable feedback and comments toward this study. This work is supported in part by a research grant from Huawei Technologies Co, Ltd., the US National Science Foundation under Grant No. CNS-0751200, CCF-0937877 and CNS-1162540, and Natural Science Foundation of Hubei Province of China under Grant No. 2014CFB239.

References

[1] "Orange File System," http://www.orangefs.org/.

[2] S. Microsystems, "Lustre File System: High-performance Storage Architecture and Scalable Cluster File System," Tech. Rep. Lustre File System White Paper, 2007.

[3] F. Schmuck and R. Haskin, "GPFS: A shared-disk File System for Large Computing Clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, 2002, pp. 231–244.

[4] R. Latham, R. Ross, B. Welch, and K. Antypas, "Parallel I/O in Practice," Tech. Rep. Tutorial of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2013.

[5] A. Caulfield, L. Grupp, and S. Swanson, "Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications," in *Proceedings of the Fourteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)*, 2009.

[6] M. Zhu, G. Li, L. Ruan, K. Xie, and L. Xiao, "HySF: A Striped File Assignment Strategy for Parallel File System with Hybrid Storage," in *Proceedings of the IEEE International Conference on Embedded and Ubiquitous Computing*, 2013, pp. 511–517.

[7] S. He, X.-H. Sun, and B. Feng, "S4D-Cache: Smart Selective SSD Cache for Parallel I/O Systems," in *Proceedings of the International Conference on Distributed Computing Systems*, 2014.

[8] S. He, X.-H. Sun, B. Feng, X. Huang, and K. Feng, "A Cost-Aware Region-Level Data Placement Scheme for Hybrid Parallel I/O Systems," in *Proceedings of the IEEE International Conference on Cluster Computing*, 2013.

[9] Z. Gong, D. A. B. II, X. Zou, Q. Liu, N. Podhorszki, S. Klasky, X. Ma, and N. F. Samatova, "PARLO: PArallel Run-time Layout Optimization for Scientific Data Explorations with Heterogeneous Access Patterns," in *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2013.

[10] Y. Wang and D. Kaeli, "Profile-Guided I/O Partitioning," in *Proceedings of the 17th Annual International Conference on Supercomputing*, 2003, pp. 252–260.

[11] S. Rubin, R. Bodik, and T. Chilimbi, "An Efficient Profile-Analysis Framework for Data-Layout Optimizations," *ACM SIGPLAN Notices*, vol. 37, no. 1, pp. 140–153, 2002.

[12] 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 International Symposium on High Performance Distributed Computing*, 2011, pp. 37–48.

[13] J. Jenkins, X. Zou, H. Tang, D. Kimpe, R. Ross, and N. F. Samatova, "RADAR: Runtime Asymmetric Data-Access Driven Scientific Data Replication," in *Proceedings of the International Supercomputing Conference*. Springer, 2014, pp. 296–313.

[14] H. Song, H. Jin, J. He, X.-H. Sun, and R. Thakur, "A Server-Level Adaptive Data Layout Strategy for Parallel File Systems," in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum*, 2012, pp. 2095–2103.

[15] "Interleaved Or Random (IOR) Benchmarks," http://sourceforge.net/projects/ior-sio/, 2014.

[16] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives," in *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, 2009, pp. 181–192.

[17] S. He, X.-H. Sun, B. Feng, and F. Kun, "Performance-aware data placement in hybrid parallel file systems," in *Proceedings of the 14th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, 2014.

[18] 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 *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2008, pp. 1–12.

[19] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai, "Automatic Identification of Application I/O Signatures from Noisy Server-Side Traces," in *Proceedings of the 12th USENIX conference on File and Storage Technologies*, 2014, pp. 213–228.

[20] "The NAS parallel benchmarks," www.nas.nasa.gov/publications/npb.html, 2014.

[21] R. Thakur, W. Gropp, and E. Lusk, "Data Sieving and Collective I/O in ROMIO," in *The Seventh Symposium on the Frontiers of Massively Parallel Computation*, 1999, pp. 182–189.

[22] 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, pp. 326–335.

[23] A. Ching, A. Choudhary, K. Coloma, L. Wei-keng, R. Ross, and W. Gropp, "Noncontiguous I/O Accesses through MPI-IO," in *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2003, pp. 104–111.

[24] 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*, San Francisco, California, 2009, pp. 183–196.

[25] H. Huang, W. Hung, and K. G. Shin, "FS2: Dynamic Data Replication in Free Disk Space for Improving Disk Performance and Energy Consumption," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, 2005, pp. 263–276.

[26] 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 (CCGrid)*, 2011, pp. 414–423.

[27] W. Tantisiriroj, S. Patil, G. Gibson, S. Seung Woo, S. J. Lang, and R. B. Ross, "On the Duality of Data-Intensive File System Design: Reconciling HDFS and PVFS," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011, pp. 1–12.

[28] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies*, 2010, pp. 1–10.

[29] T. Pritchett and M. Thottethodi, "SieveStore: a Highly-Selective, Ensemble-level Disk Cache for Cost-Performance," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010, pp. 163–174.

[30] X. Zhang, K. Davis, and S. Jiang, "iTransformer: Using SSD to Improve Disk Scheduling for High-performance I/O," in *Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium*, 2012, pp. 715–726.

[31] X. Zhang, K. Liu, K. Davis, and S. Jiang, "iBridge: Improving Unaligned Parallel File Access with Solid-State Drives," in *Proceedings of 27th IEEE International Parallel and Distributed Processing Symposium*, 2013.

[32] Q. Yang and J. Ren, "I-CASH: Intelligently Coupled Array of SSD and HDD," in *Proceedings of the IEEE 17th International Symposium on High PerformanceComputer Architecture*, 2011, pp. 278–289.

[33] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems," in *Proceedings of the international conference on Supercomputing*, 2011, pp. 22–32.

[34] S. He, Y. Liu, and X.-H. Sun, "PSA: A Performance and Space-Aware Data Layout Scheme for Hybrid Parallel File Systems," in *Proceedings of the Data Intensive Scalable Computing Systems Workshop*, 2014, pp. 563–576.