A Heterogeneity-Aware Region-Level Data Layout for Hybrid Parallel File Systems

Shuibing He[123], Xian-He Sun[2], Yang Wang[4], Antonis Kougkas[2], Adnan Haider[2]
[1]School of Computer, Wuhan University, Wuhan, Hubei 430072, China
[2]Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA
[3]State Key Laboratory of High Performance Computing
[3]National University of Defense Technology, Changsha, Hunan 410073, China
[4]Shenzhen Institute of Advanced Technology, Chinese Academy of Science, Shenzhen, China
heshuibing@whu.edu.cn, {sun, ywang358}@iit.edu, {akougkas, ahaider3}@hawk.iit.edu

***Abstract*—Parallel file systems (PFS) are commonly used in high-end computing systems. With the emergence of solid state drives (SSD), hybrid PFSs, which consist of both HDD and SSD servers, provide a practical I/O system solution for data-intensive applications. However, most existing PFS layout schemes are inefficient for hybrid PFSs due to their lack of awareness of the performance differences between heterogeneous servers and the workload changes between different parts of a file. This lack of recognition can result in severe I/O performance degradation. In this study, we propose a heterogeneity-aware region-level (HARL) data layout scheme to improve the data distribution of a hybrid PFS. HARL first divides a file into fine-grained, varying sized regions according to the changes of an application's I/O workload, then chooses appropriate file stripe sizes on heterogeneous servers based on the server performance for each file region. Experimental results of representative benchmarks show that HARL can greatly improve the I/O system performance.**

***Index Terms*—Parallel I/O System; Parallel File system; Solid State Drive; Data Layout**

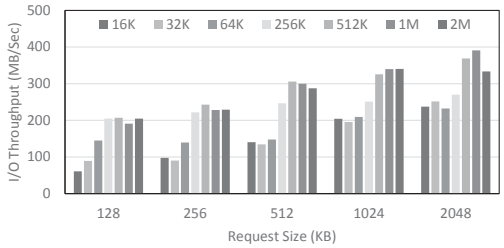
I. Introduction

Many large-scale applications have been more and more data intensive over the past decades, and I/O performance has become the bottleneck of computer systems. To tackle this problem, parallel file systems (PFS), such as OrangeFS [1], Lustre [2], GPFS [3] and PanFS [4], were introduced in high-performance computer systems. By serving a client request concurrently from multiple file servers, the aggregate I/O bandwidth is largely improved. However in order to fully utilize file servers, one must account for both the application and underlying server characteristics.

New emerging storage technologies, such as flash-based solid state drives (SSD), provide a possible alternative for I/O system design. Unlike HDDs, SSDs are composed of semiconductor chips and provide higher performance than HDDs [5]. While SSD is ideal for performance, completely replacing HDDs with SSDs in a large cluster is not a widely adopted solution due to the concern of the monetary cost. Thus, a hybrid PFS, which contains both HDD servers (HServer) and SSD servers (SServer), is more practical for HPC systems under a limited cost budget [6], [7].

While hybrid PFSs are promising, their efficiency relies on an efficient file data layout, which is an algorithm defining how a file's data is distributed on available nodes. Most existing

(a) I/O time of each server under a fixed I/O pattern and stripe size



(b) Throughput with varied I/O patterns and stripe sizes

Fig. 1. Performance statistics of IOR in a hybrid PFS. In (a), server 1-6 are HServers, and server 7-8 are SServers. In (b), the legend "#K" denotes the data layout with a fixed-size stripe of #KB on each server.

layout schemes distribute file data across multiple servers with a fixed-size stripe [8], as shown in Figure 2(a). This can provide concurrent data access from multiple servers and come with even data placement on each server. Although widely used and simple to implement, these schemes are designed and suitable for homogeneous servers. When applied to hybrid PFSs, these schemes will raise the following challenges.

First, the performance gap between HServers and SServers can significantly degrade the performance of PFSs. SServers always have higher performance than HServers, thus require less I/O time to complete the same amount of data accesses. However, current layout methods assign identical stripes for HServers and SServers, which can lead to severe load imbalance among heterogeneous servers. To illustrate this issue, we ran IOR [9] with 512KB request size and 16 processes on a hybrid OrangeFS file system with the default layout (Stripe size is 64KB). Figure 1(a) shows the I/O time on

each server, normalized to the minimum of all servers. We can observe that the slow HServers (Server 1-6) take roughly 350% I/O time compared with fast SServers (Server 7-8), which means that the potential of the high-performance SServers are significantly underutilized.

Second, complex I/O workloads may also compromise the efficiency of I/O systems. Many applications may issue various I/O requests to different parts of a large file [10], [11]. Request sizes can be large at one chunk of the file but small at another; request types can be read operation in one I/O phase but write in another. However, traditional layout schemes adopt a fixed-size stripe for the whole file, when in actuality the stripe size is only suitable for certain I/O workloads [10]. Figure 1(b) shows the performance of IOR with varied request sizes from 128KB to 2048KB under fixed stripe sizes from 16KB to 2MB. We can see that there is a huge variation in I/O bandwidth under different I/O workloads and stripe sizes. It shows that while traditional file-level striping methods may be efficient for certain parts of a file, but may suffer from suboptimal overall I/O system performance.

In this paper, we propose a heterogeneity-aware region-level (HARL) data layout scheme to address the challenges existing in the current data distribution of PFSs. Since fixed-size stripe is not optimal for either heterogeneous servers or complex I/O workloads, HARL relies on a server and application aware file stripe allocation scheme to determine the optimal stripe sizes on heterogeneous servers. Specially, it first divides a file into fine-grained regions according to the changes of an application's I/O workload; then, HARL assigns appropriate file stripe sizes on heterogeneous servers based on their storage performance for each file region. It essentially represents a deviation from the traditional one dimensional fixed-size stripe layout to a two-dimensional varied-size stripe layout. In this way, HARL significantly speeds up the I/O system performance by mitigating load imbalance among heterogeneous servers and increasing I/O efficiency of data accesses in each file region. Moreover, HARL requires no modifications to data-intensive applications and can be extended to any hybrid file system with two or more file server performance profiles.

Specifically, this study makes the following contributions.

- A mathematical cost model, which considers I/O patterns, system architecture, network overhead, storage performance and data layout characteristics, is introduced to evaluate the data access time of one file request in a hybrid PFS.
- A data layout scheme, which logically divides a file into regions with similar workloads and then optimizes each region by adjusting the stripe sizes of HServers and SServers based on the cost model, is presented to optimize the hybrid file system performance.
- A prototype of the heterogeneity-aware region-level data layout scheme is implemented and integrated into MPICH2 [12]. This implementation is transparent to applications and can be applied to different parallel file systems to increase its portability.
- A thorough evaluation of HARL has been conducted with the IOR benchmark and BTIO benchmark. Experimental results show that HARL can significantly improve the I/O throughput of hybrid parallel file systems.

The rest of this paper is organized as follows. Section II discusses the related work. The design and implementation of HARL is described in section III. Section IV presents the performance evaluation with commonly used benchmarks. Finally, the conclusions are summarized in section V.

II. Related work

In this section we briefly discuss more related work on improving parallel I/O system performance.

I/O Access Reorganization: A great deal of research has focused on reorganizing I/O accesses at the parallel I/O middleware layer. For example, instead of accessing multiple small, noncontiguous requests, data sieving [13] applies the strategy of accessing a contiguous chunk created by gathering the noncontiguous requests. Datatype I/O [14] and List I/O techniques [15] allow noncontiguous I/O requests to be converted into a single I/O request, thereby limiting the number of total requests. Collective I/O [13] also optimizes by rearranging I/O accesses into a larger contiguous request, but considers multiple processes of a parallel program instead of an individual process. For write optimization, PLFS [16] redirects multiple parallel requests to a set of efficiently reorganized log-formatted files to generate more sequential write requests, but the read performance of these files may not be good due to the inevitable data restructuring.

Data Layout in HDD-based File Systems: Parallel file systems support different data layout strategies, which allow for numerous data layout optimization methods. Several techniques, including data partition [17], [18], data migration [19], and data replication [8], [20], [21] 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 [10]. Another methodology, server-level adaptive layout strategy, selects different stripe sizes depending upon the type of the file server [22]. PARLO is designed for accelerating queries on scientific datasets by applying user specified optimizations [23]. AdaptRaid confronts load imbalance in heterogeneous disk arrays [24] using an adaptive number of blocks, which cannot be implemented in PFSs.

Data Layout in SSD-based File Systems: SSDs are commonly integrated into parallel file systems due to their performance benefits. A popular method is to use SSDs as a cache of traditional HDDs, e.g. Sievestore [25], iTransformer [26], and iBridge [27]. Another widely used approach is to utilize SSDs as a part of data storage, such as I-CASH [28] and Hystor [29]. Wu et al. [30] discusses the data placement and scheduling tradeoffs for hybrid storage. Although effective, the vast majority of research is focused on single file servers. S4D-Cache [7] uses all SSD-based file servers as a cache and selectively caches performance-critical data on these high performance servers. CARL [31] selects and places file regions with high access costs onto SSD-based file servers at the

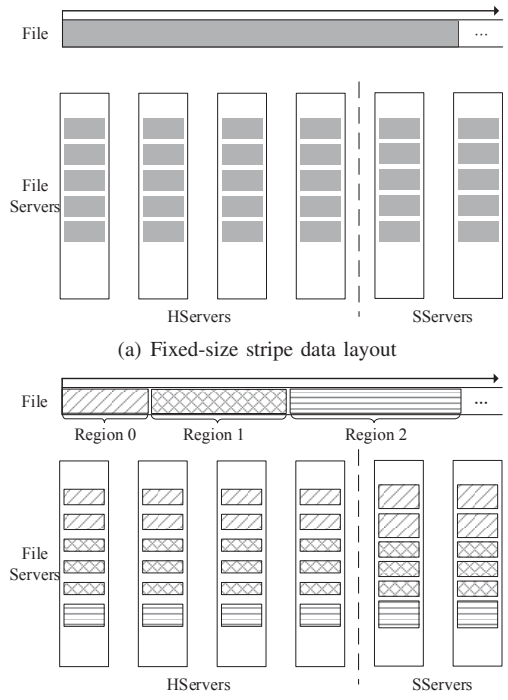


Fig. 2. **Two data layout schemes in a hybrid parallel file system.** This figure shows how a file's data are distributed on HServers and SServers, focusing on the stripe size configuration. The height of the rectangle on each server represents the stripe size assigned to them. While case (a) uses a fixed-size stripe for each server within the whole file, case (b) divides a file into multiple regions and uses varied-size stripes for HServers and SServers to distribute data on each region.

I/O middleware layer, but the region cannot be placed onto both SSDs and HDDs. PADP [32] and PSA [33] employ stripe size variation to improve the performance of hybrid PFSs. HAS [34] adaptively selects the optimal data layout for heterogeneous parallel file systems with specific access patterns.

The above mentioned techniques are effective in improving the performance of PFSs. However, there is little effort devoted on data layout considering both heterogeneous servers in a hybrid PFS and complex I/O workloads at different part of a file. In contrast, HARL uses adaptive file striping method to address this issue.

III. Heterogeneity-aware Data Layout Scheme

A. Overview of HARL

The proposed data layout scheme, HARL, aims to optimize the hybrid PFS layout by using varied-size file stripes instead of fixed size. To accommodate both heterogeneous servers and complex I/O workloads, HARL adopts the idea of "divide and conquer" to achieve the optimal data layout. First, it divides a large file into several small regions such that each region has similar I/O workloads. Then it determines the appropriate file stripe sizes on heterogeneous servers based on their storage performance for each region.

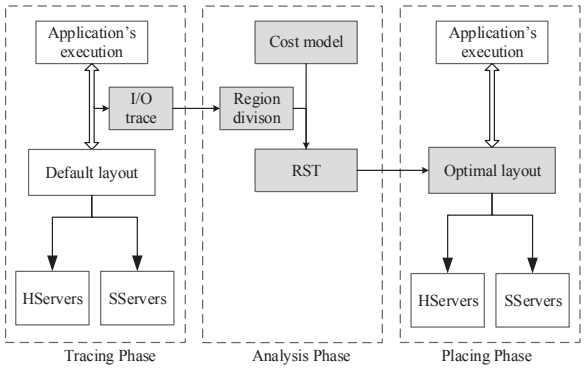


Fig. 3. The procedure for HARL scheme

Figure 2(b) illustrates the idea of the heterogeneity-aware region-level data layout scheme. In this example, HARL divides a file into three adjacent regions and assigns different stripe sizes on HServers and SServers for each region. Specially, since SServers have higher I/O performance, SServers are usually allocated with larger stripe sizes than HSServer in each region, so that each server can finish their I/O requests nearly at the same time. Compared with the traditional layout (Figure 2(a)), HARL is a fine-grained, adaptive data layout scheme, which can significantly alleviate the load imbalance among heterogeneous servers and improve the hybrid PFS performance.

Many data-intensive applications have predictable I/O patterns [17], [35], [36]. For example, the BTIO application [37], an I/O kernel responsible for solving block-tridiagonal matrices on a three dimensional array, has this feature. For BTIO, once the size of the array, the number of time steps, the write interval, and the number of processes are given, the I/O behaviors can be accurately predicted before the program executes. Since the program often run many times and these patterns do not fluctuate significantly, it provides an opportunity for HARL to achieve the optimal data layout based on I/O behavior analysis.

Figure 3 shows the procedure of HARL, which includes three phases. In the *Tracing Phase*, the run-time statistics of data accesses are collected into a trace file during the application's first execution. In the *Analysis Phase*, by analyzing the I/O trace, the large file is divided into different regions according to the application's I/O characteristics, then each region's stripe sizes are determined based on a data access cost model. In the *Placing Phase*, the file is placed on the underlying heterogeneous servers at runtime with the optimal file stripes obtained in the *Analysis Phase*. Through these three phases, HARL can largely improve the application's I/O performance in later runs.

B. I/O Trace Collection

A *trace collector* is responsible for collecting run-time file access information of parallel applications. While there are some techniques and tools that can be used for data analysis, we use IOSIG, which is an I/O pattern collection and analysis

tool developed in our previous work [38], to capture the information required by HARL. IOSIG is a pluggable library of MPI-IO, which supports MPI-IO and standard POSIX IO interfaces. IOSIG can help to gather all the information of file operations, including file access type, operation time, and other process related data. After running the applications with the *trace collector*, we can get process ID, MPI rank, file descriptor, type of operation, offset, request size, and time stamp information. To facilitate the region division and guide the optimal data layout, the collector sorts all file read and write requests in ascending order in terms of their offsets.

C. File Region Division

Since fixed stripe sizes on servers are not able to provide optimal performance for the whole file, as discussed in Section I, HARL divides a file into fine-grained regions and applies special stripe size optimization for each region. One may logically divide the address space of a file into regions by a fixed chunk size (e.g. 64MB or 128MB). While this method is simple, it is difficult to select a proper region size for varying I/O patterns in real systems. As opposed to this approach, HARL adopts a varied-size region division algorithm, as shown in Algorithm 1.

Algorithm 1: File Region Division Algorithm

Input : Sizes of file requests: $r_0, ..., r_{n-1}$; Offset of file requests: $o_0, ..., o_{n-1}$
Output: Offset of each file region $O_0, ..., O_{m-1}$; Average request size for each file region: $A_0, ..., A_{m-1}$

1 $sum = 0$; $cv\_prev = 0$; $reg = 0$ /*region #*/ ;
2 $reg\_init = 0$ /*The first request served by this region */;
3 $threshold = 100\%$;
4 **for** $i = 0; i < n; i++$ **do**
5 $\quad sum += r_i$;
6 $\quad avg = \frac{sum}{i - reg\_init + 1}$;
7 $\quad std = \sqrt{\sum_{k=reg\_init}^{i} (r_k - avg)^2 / (i - reg\_init + 1)}$;
8 $\quad cv\_new = std/avg$;
9 $\quad$ **if** $(100 * |cv\_new - cv\_prev|)/cv\_prev < threshold$ **then**
10 $\quad\quad cv\_prev = cv\_new$;
11 $\quad$ **else**
12 $\quad\quad sum = 0$ /*Restart with new CV */;
13 $\quad\quad cv\_prev = 0$;
14 $\quad\quad$ /* Set offset and average request size in region: reg */ ;
15 $\quad\quad O_{reg} = o_{reg\_init}$;
16 $\quad\quad A_{reg} = avg$;
17 $\quad\quad reg\_init = i + 1$ /*The first request served for next region will be $i + 1$ */;
18 $\quad\quad$ /*Created region now increment for next region */
$\quad\quad reg++$;
19 $\quad$ **end**
20 **end**

The algorithm's goal is to identify continuous file chunk accessed with closest I/O patterns, so that a given data layout may benefit more I/O requests. Starting from file offset 0, the algorithm uses *average request size* as a common feature to find the proper splitting points. It reads the first two entries of the requested size from the trace file and calculates the coefficient of variation (CV), the result of dividing the standard deviation by the average request size in the current sample. It continually adds the next request and calculates the CV until the trace ends. If the new CV value falls close to the previous one, namely, the percentage difference between the new CV value and the previous one is less than 100% (line 9), it continues adding the next entry and repeats the calculations. Otherwise, it logs the offset, creates a splitting point, starts a new region, and restarts calculations with a new CV. As a normalized measure of dispersion of data distribution, CV is very sensitive to changes in the average request size and allows us to detect the point where the application changes the I/O behavior. At the end, the algorithm returns a list of file regions with their average request sizes.

One potential issue is that this algorithm may generate too many regions, which leads to substantial extra metadata management overhead and compromises the final I/O performance. To overcome this, we limit the number of created region by adjusting the *threshold* value. If the number of the regions is greater than the number from the fixed-size region division, as in the segment-level layout scheme [10], the *threshold* increases from 100% to a higher value. This tuning can loosen the algorithm's sensitivity to the request size variation and result in less metadata overhead.

TABLE I
PARAMETERS IN COST ANALYSIS MODEL

I/O Pattern Parameters	
o	Offset of the file request
r	Size of the file request
op	Type of the file request (read or write)
Architecture Parameters	
M	Number of HDD servers (HServers)
N	Number of SSD servers (SServers)
Network Parameters	
t	Unit data network transfer time
Storage Parameters	
α_h^{min}	Minimum startup time on HServer
α_h^{max}	Maximum startup time on HServer
β_h	Unit data transfer time on HServer
α_{sr}^{min}	Minimum startup time for read on SServer
α_{sr}^{max}	Maximum startup time for read on SServer
β_{sr}	Unit data transfer time for read on SServer
α_{sw}^{min}	Minimum startup time for write on SServer
α_{sw}^{max}	Maximum startup time for write on SServer
β_{sw}	Unit data transfer time for write on SServer
Data Layout Parameters	
h	Stripe size on HServer
s	Stripe size on SServer

D. Access Cost Model

To obtain the optimal stripe size on each server for a given file region, we introduce an analytical model to evaluate the data access cost of one file request in a hybrid PFS. The cost is a function of different parameters listed in Table I, which fully consider the application, system and data layout characteristics.

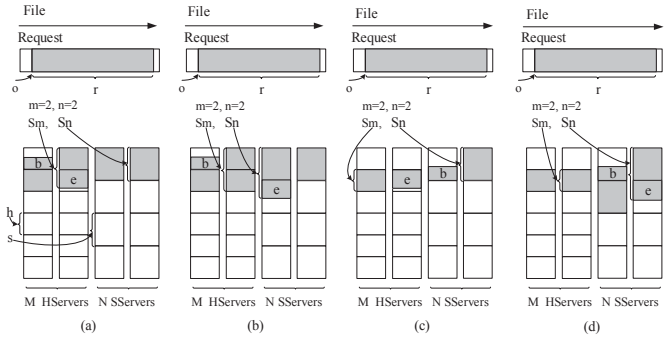


Fig. 4. Four typical cases of file sub-request distribution on servers. (a): Both beginning and ending sub-requests are located on HServers; (b): Beginning sub-request is on HServers but ending sub-request is on SServers; (c): Beginning sub-request is on SServers but ending sub-request is on HServers; (d): Both beginning and ending sub-requests are on SServers.

Note that the storage parameters show distinct features of heterogeneous servers. First, SServer has a much smaller start up time than HServer. Second, SServer has a smaller data transfer time than HServer. Third, SServer usually has a slower write performance than its read because write operations require time-consuming garbage collection and wear leveling [5].

The cost is defined as the I/O completion time of each file request, which includes three parts: the network transfer time T_X, the storage startup time T_S, and the storage transfer time T_T. T_X is the data transfer time on network, T_S is the consumption before data operations on disks, and T_T is the time spent on actual data read/write operations.

The request cost T is determined by the maximal cost of all sub-requests. Assume the sub-requests are distributed on the m ($m \in [0, M]$) HServers and n ($n \in [0, N]$) SServers, and the maximal sub-request sizes on HServers and SServers are s_m and s_n, then we can calculate the request cost as following.

T_X is related with the data size and the network data transfer rate. It is determined by the maximal network transfer cost of all sub-requests on HServers and SServers. Thus

$$T_X = \max\{s_m t, s_n t\} \tag{1}$$

T_S is determined by the maximal startup time on the $m+n$ servers. Let α denote the startup time in each HServer, then the startup time of the m sub-requests can be a variable $X = max(\alpha_1, \alpha_2, \cdots, \alpha_m)$, where α_i ($1 \leqslant i \leqslant m$) has an independent identical distribution as α. Assume α follows an uniform distribution on [α_h^{min}, α_h^{max}], then the probability function of α is $P(\alpha < x) = (x - \alpha_h^{min})/(\alpha_h^{max} - \alpha_h^{min})$, where $x \in [\alpha_h^{min}, \alpha_h^{max}]$, and the probability density function of X is

$$f(x) = \frac{m \times (x - \alpha_h^{min})^{m-1}}{(\alpha_{sr}^{max} - \alpha_{sr}^{min})^m}, \alpha_h^{min} \leqslant x \leqslant \alpha_h^{max} \tag{2}$$

Hence, the startup time on the m HServers is

$$T_h^S = \int_{\alpha_h^{min}}^{\alpha_h^{max}} x f(x) dx = \alpha_h^{min} + \frac{m}{m+1}(\alpha_h^{max} - \alpha_h^{min}) \tag{3}$$

Similarly, the startup time for read sub-requests on the n SServers is

$$T_{sr}^S = \alpha_{sr}^{min} + \frac{n}{n+1}(\alpha_{sr}^{max} - \alpha_{sr}^{min}) \tag{4}$$

Based on Equation (3) and (4), the overall startup time for a file read request is

$$T_{SR} = \max\{T_h^S, T_{sr}^S\} \tag{5}$$

T_T is also determined by the maximal storage transfer time of all sub-requests. For a read request, it can be calculated as

$$T_{TR} = \max\{s_m \beta_h, s_n \beta_{sr}\} \tag{6}$$

Based on Equation (1), (5) and (6), the overall cost of a file read request is

$$T = T_X + T_{SR} + T_{TR} \tag{7}$$

The Equations (5), (6) depict the cost for reads, startup and transfer time for writes (T_{SW} and T_{TW}) will be similar except we exchange the read parameters with write. Thus the overall cost of a write request is

$$T = T_X + T_{SW} + T_{TW} \tag{8}$$

From the above equations, we can see that T depends on four parameters: s_m, s_n, m and n, which can be calculated according to the stripe sizes h and s. We assume the file data are distributed on the 0 to M+N-1 servers in a round-robin way, and let $S = M * h + N * s$, $r_b = \lfloor o/S \rfloor$, $r_e = \lfloor (o+r)/S \rfloor$, $l_b = o - r_b * S$, and $l_e = (o+r) - r_e * S$, then the server number of the beginning and ending sub-requests are $n_b = (l_b < M * h)?\lfloor l_b/h \rfloor : \lfloor (l_b - M * h)/s \rfloor + M$, $n_e = (l_e < M * h)?\lfloor l_e/h \rfloor : \lfloor (l_e - M * h)/s \rfloor + M$, the size of the beginning and ending fragment are $s_b = (l_b < M * h)?\lfloor h - l_b \% h \rfloor : h - (l_e - M * h)\% s$, $s_e = (l_e < M * h)?\lfloor h - l_e \% h \rfloor : s - (l_e - M * h)\% s$. Based on the locations where the file request begins and ends, the sub-request distributions fall into four cases, as shown in Figure 4. Due to space limitation, we only describe how to calculate these parameters for case (a) where the request begins and ends at certain HServers. Let $\triangle_r = r_e - r_b$, $\triangle_c = n_e - n_b$, then the four critical parameters are calculated as in Figure 5. By following the same arguments, we can derive the parameters for other cases.

Condition		Sm	Sn	m	n
Δr=0	Δc=0	s_b	0	Δc +1	0
	Δc=1	max{s_b, s_e}	0	Δc +1	0
	Δc>1	h	0	Δc+1	0
Δr≥1	Δc =0	max{ Δr*h-h+s_b+s_e, Δr*h}	Δr*s	M	N
	n_b+1=M and n_e=0	max{ Δr*h-h+s_b, Δr*h-h+s_e}	Δr*s	Δr =1?2: M	N
	n_b+1!=M or n_e!=0	Δr *h	Δr*s	Δc <-1?(M+1+ Δc):M	N

Fig. 5. The calculation of critical parameters s_m, s_n, m and n in case (a) of Figure 4. These parameters are rated with stripe sizes h and s.

From the cost model, one can find that the access time of a file request can be significantly impacted by the server stripe sizes h and s.

E. Stripe Sizes Determination

Based on the above model for one file request, HARL uses a heuristic iterative algorithm to find the optimal stripe sizes on HServers and SServers for each region. The goal is to minimize the data access cost of all file requests in that region instead of a single request.

Algorithm 2: Region Stripe Size Determination

Input : File region: reg including file request $R_0, ..., R_{k-1}$, Average request size $\overline{R}$ in Reg
Output: optimal stripe sizes: H for HServer, S for SServer

```
1  step ← 4KB;
2  opt_cost ← ∞;
3  for h ← 0; h ≤ R̄; h ← h + step do
4      for s ← h + step; s ≤ R̄; s ← s + step do
5          for i ← 0; i < k; i ← i + 1 do
6              Reg_cost ← 0;
7              if operation_type(R_i) = Read then
8                  T_i ← Calculate cost of R_i according to
                   Equation (7) ;
9              else
10                 T_i ← Calculate cost of R_i according to
                   Equation (8) ;
11             end
12             Reg_cost ← Reg_cost + T_i;
13         end
14         if Reg_cost < opt_cost then
15             opt_cost ← Reg_cost;
16             H ← h;
17             S ← s;
18         end
19     end
20 end
```

Algorithm 2 shows the procedure of determining the optimal stripe sizes. Starting from h equaling 0, the loop iterates h in 'step' increments while h is less than $\overline{R}$. We use the average request size $\overline{R}$ because we use it to divide the region in Algorithm 1 and it is a good metric to describe the common feature of the workloads. The extreme configuration we do consider is where h is $\overline{R}$, which means dispatching the file request data only on one HServer may obtain better I/O performance. In the second loop, s starts from a size which is larger than h because this configuration can lead to load balance among heterogeneous servers. We also consider the extreme case where a file request data only placed on one SServer. For each pair of stripe sizes configuration, the loop iterates to calculate the total access cost of all file requests in that region, according to the proposed data access cost in Section III-D. Note that the request cost is accumulated based on the request type (line 9 and 12) since the read operations come with different performance as write. Finally, the pair of stripe sizes leading to minimal region access cost (Reg_cost) is chosen. The 'step' value is 4KB (line 3), which can be chosen by the user. Finer 'step' values result in more precise h and s values, but with increased cost calculation overhead. However, the computational overhead of this algorithm is acceptable because the calculations are simply arithmetic operations and run off-line.

RST

Region #	File_offset	HServer stripe size	SServer stripe size
0	0	16KB	64KB
1	128MB	36KB	144KB
2	192MB	26KB	80KB
⋮	⋮	⋮	⋮

Fig. 6. The data structure of the RST table in HARL scheme

To guide data placement, the optimal stripe sizes for each region is stored into a global *region stripe table* (RST). Figure 6 shows an example of the data structure of RST. In this example, the file consists of multiple regions, and the stripe sizes for the first three regions are {16KB, 64KB}, {36KB, 144KB}, and {26KB, 80KB} respectively. Although the metadata includes more information, its size is not too large because the number of regions is limited in the region division algorithm in Section III-C. Moreover, if adjacent regions have the same optimal stripe sizes, the two regions are combined into a larger region. This can further reduce the metadata management overhead.

F. Heterogeneity-Aware Data Placement

In the *Placing Phase*, the file is placed on the underlying heterogeneous servers with optimal stripe sizes for each fine-grained file region. A PFS commonly includes three components. The file clients issue requests on behalf the applications, the servers are responsible for storing file data, and the metadata servers (MDS) contain the description information of the files. During a file operation, a client first contacts MDS to get the files metadata, then it interacts with servers directly. To perform the optimal data placement, MDSs look up the RST table according to the request's offset and length, and return this information to the client. Then, the client writes the file data on each server with the optimal stripe size from RST.

G. Implementation

The proposed layout scheme can be implemented either in the PFS or I/O middleware layer. The former solution requires specific file metadata communication between clients and servers to support the region-level striping strategy, which is not currently supported by PFSs. In order to maintain its portability and achieve a simple implementation, HARL is integrated into the I/O middleware layer, which lies above various PFSs.

We implement HARL within MPICH2 [12] above OrangeFS [1]. In the *Analysis Phase*, we use one file server in the parallel file system to test the startup time α and data transfer time β for HServers and SServers with read/write patterns. These parameters can vary with different I/O patterns. In addition, we use a pair of nodes (one client node and one file server) to estimate the network transfer time t. We repeat the tests thousands of times (the number is configurable), and then calculate their average values, which are used as the parameter values.

In the *Placing Phase*, HARL logically maps a large file into multiple OrangeFS files, each representing a separate file region with similar I/O workloads. In MPICH2, a *region-to-file mapping table* (R2F) is used to record the translation from a logical file region to a physical OrangeFS file. For each region, data is distributed on underlying servers with the optimal stripe sizes stored in RST. This can be implemented by leveraging the existing varied-size striping mechanism in OrangeFS. RST and R2F are stored in the same directory as the applications, and are loaded when MPI_Init() is triggered and unloaded when MPI_Finalize() is executed. Furthermore, the MPI_File_read/write() (and other variants of read/write) are modified, so file requests can be automatically forwarded to the files in the PFSs with optimized stripe sizes.

IV. Performance Evaluation

A. Experimental Setup

The experiments were conducted on a 65-node SUN Fire Linux cluster. Each computing node has two AMD Opteron(tm) processors, 8GB memory and a 250GB HDD. The operating system is Ubuntu 9.04 and the parallel file system is OrangeFS v2.8.6. All nodes are equipped with Gigabit Ethernet interconnection, and eight nodes are equipped with additional PCI-E X4 100GB SSD.

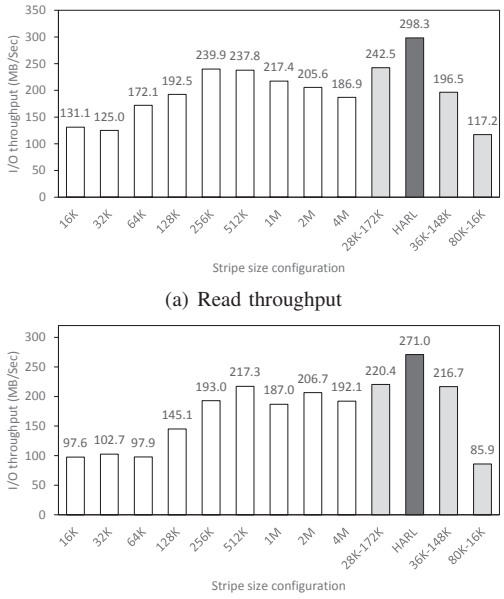
Eight nodes are used as computing nodes, eight as HServers, and eight as SServers. All SServers and HServers are accessed through one OrangeFS. By default, six HServers and two SServers are used to build the hybrid OrangeFS file system, and the file is striped over the file servers in a round-robin fashion. In the experiments, we compared three data layout schemes: the fixed-size stripe, randomly-chosen stripe and the proposed HARL scheme.

The widely-used parallel file system benchmarks, IOR [9] and BTIO [37], are used to test the hybrid file system performance.

B. IOR Benchmark

1) Read and Write Results: The experiments were conducted to compare the I/O performance of the hybrid PFS with the proposed data layout scheme, HARL, and two other strategies, which use a fixed-size or randomly chosen file stripe. For the following tests, IOR benchmark runs with 16 processes, and the request size is kept to 512KB unless otherwise specified. Each process is responsible for accessing its own $1/16$ of a 16GB shared file and continuously issues requests with random offsets.

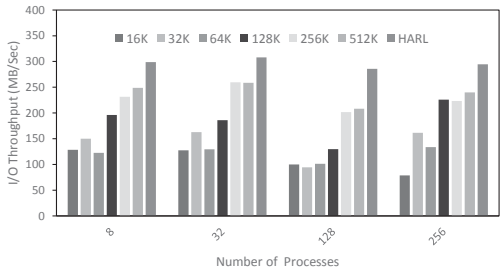
Figure 7 demonstrates the I/O performance of the hybrid file system with different layouts. In this figure, layout '64K' means the stripe sizes are 64KB for all file servers, and '36K-148K' means the stripe size is 36K for HServers and 148K for SServers. The rest of the layouts have similar meaning. It is observed that the proposed heterogeneity-aware layout can achieve I/O performance improvement for both read and write operations. While the performances of the fixed-size and randomly chosen stripe schemes vary with the adopted stripe size, HARL provides the best performance. With the optimal data layout of $\{32KB, 160KB\}$ and $\{36KB, 148KB\}$ for reads and writes respectively, HARL improves the I/O performance by 73.4% and 176.7% over the default layout with a fixed-size stripe of 64KB. Compared with other layouts with different *but* fixed-size stripes, HARL improves the performance up to 138.6 % for reads and 177.6 % for writes. Compared with the randomly chosen stripe strategies, the read performance can improve to 154.5% and write performance can improve to 215.4%. The experiments prove HARL performs optimally and the stripe size determining formula is effective.



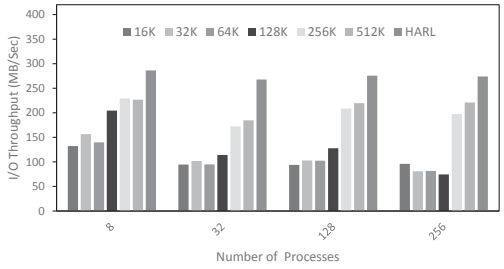
(a) Read throughput

(b) Write throughput

Fig. 7. Throughputs of IOR with different layouts

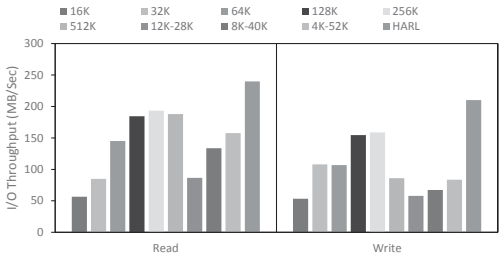


(a) Read throughput

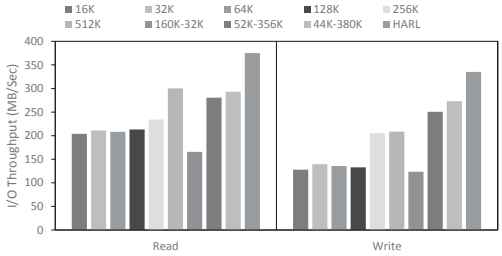


(b) Write throughput

Fig. 8. Throughputs of IOR with various number of processes

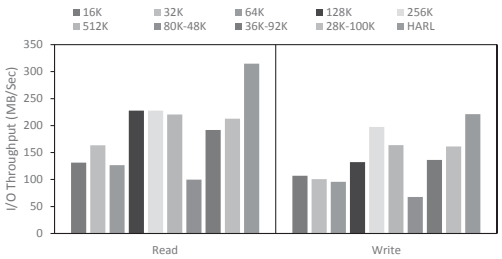


(a) 128KB

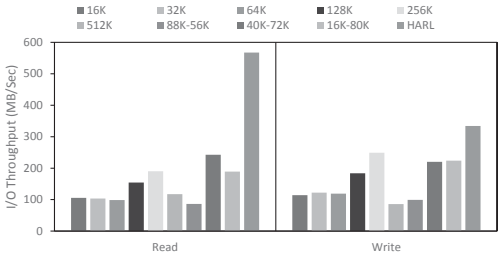


(b) 1024KB

Fig. 9. Throughputs of IOR with various request sizes



(a) 7HServers : 1SServer



(b) 2HServers : 6SServers

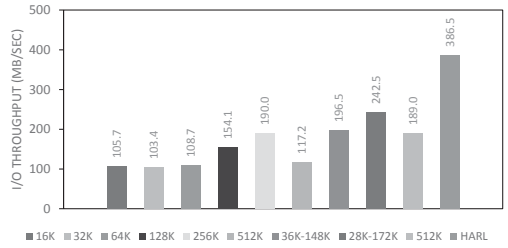
Fig. 10. Throughputs of IOR with various file server configurations

2) Varying Number of Processes: The I/O performance is also evaluated with a varied number of processes. The IOR benchmark is executed with 8, 32, 128 and 256 processes at a fixed request size of 512KB. As shown in Figure 8, the results are similar to the previous test. HARL improves I/O performance for both read and write operations. With different number of processes, the I/O throughput increases to 144.1%, 141.8%, 202.7% and 274.1% for reads compared with layout schemes with a fixed-size stripe, and 116.4%, 182.7%, 192.8%, and 268.3% for writes, respectively. Compared with the default layout (stripe size of 64KB), the read performance achieves a 144.1%, 138.1%, 182.3%, and 120.2% improvement, and write performance achieves a 104.8%, 182.2%, 168.5%, and 235.1% improvement. The results illustrate that HARL has high scalability in terms of number of processes.

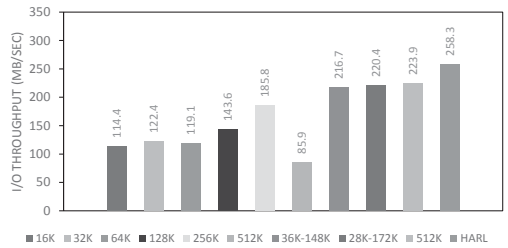
3) Varying Request Sizes: In Figure 9, the I/O performance is examined with varied request sizes. The IOR benchmark is executed with request sizes of 128KB and 1024KB. HARL improves read performance from 24.1% to 325.0%, and write performance from 32.4% to 293.5%, in comparison with conventional layout methods. In terms of the default layout with 64KB stripe size, HARL achieves an 80.1% improvement for read and 147.1% for write operations. Compared with layout strategies which use randomly varied stripe sizes, the read performance boosts from 20.6% to 222.3%, and write performance increases from 22.7% to 263.1%. When the request size is 128KB, the optimal stripe size pair is {0KB, 64KB}; thus, distributing the file only on the two SServers offers the highest I/O performance. When the request size is 1024KB, HARL distributes the file on both HServers and SServers for higher I/O performance.

4) Varying File Server Configurations: The I/O performance is examined with varied ratios of HServers to SServers. The OrangeFS is built using HServers and SServers with the ratios of 7:1 and 2:6. The request size is kept to 512KB. Figure 10 shows the average I/O throughput with different file server configurations. As the results depict, HARL improves I/O performance for both data reads and writes. Read performance increases from 37.6% to 556.1%, and write performance improves from 112.2% to 288.7% in comparison with other layout methods. Compared with the default layout with a 64KB stripe size, HARL achieves a 474.9% improvement for reads and a 180.3% for writes. In the experiments, read and write performance improved as the number of SServers increased. This is because the I/O performance of SServers is efficiently utilized by the heterogeneity-aware layout scheme. If the number of SServers is small, HARL distributes the file on both SServers and HServers. However, if the number of SServers is greater, the file is placed only on high-performance SServers.

5) Varying I/O Workloads: All the above results have clearly confirmed the efficiency of HARL with uniform I/O workloads. In the experiments, we evaluated HARL under varied I/O accesses. In order to simulate the complicated non-uniform I/O workload, we modified IOR benchmark to access a four-region data file. The size of each region was 256MB, 1024MB, 2048MB and 4096MB. For each region, IOR issues requests with different request sizes. Figure 11 shows the average I/O throughput of the hybrid PFS with different data layout strategies. From the results, it can be easily observed that HARL improves read performance from 59.4% to 265.8%, and write performance from 17.2% to 200.7% compared with other layout methods. Compared with the default data layout with a 64KB fixed stripe size, HARL achieves a 255.6% improvement for reads and 116.9% for writes. The results indicate that the new region-level layout scheme, which divides a file into regions with similar workloads, is capable of increasing performance at a large scale for complex I/O



(a) Read throughput



(b) Write throughput

Fig. 11. I/O throughputs with non-uniform workloads

workloads compared with the existing file-level data layout schemes.

C. BTIO Benchmark

Apart from IOR benchmark above, we also used BTIO benchmark to evaluate HARL. BTIO represents a typical scientific application with interleaved intensive computation and read/write mixed I/O phases. BTIO uses a Block-Tridiagonal (BT) partitioning pattern to solve the three-dimensional compressible Navier-Stokes equations. We consider the Class A and full subtype BTIO workload in the experiments. That is, BTIO writes and reads a total size of 1.69GB data with collective I/O functions. We use 4, 16, and 64 compute processes since BTIO requires a square number of processes. Output file is striped across six HServers and two SServers. Figure 12 displays the aggregate I/O throughputs. Compared with the default layout with 64KB stripe size, HARL achieves 163.5%, 116.9%, and 114.8% improvement with 4, 16, 64 processes, respectively. For other varied but fixed-size striping methods, HARL also demonstrates performance advantages.

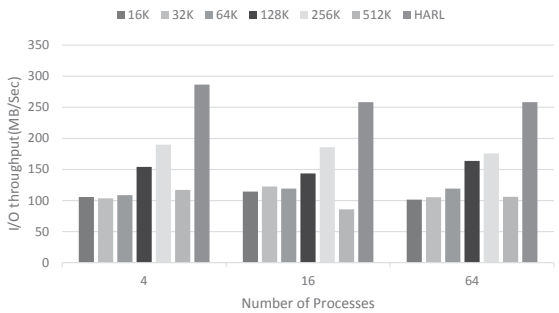


Fig. 12. I/O throughputs of BTIO benchmark with different layouts

D. Discussion

While making all file servers complete their I/O accesses almost at the same time, HARL would potentially lead to more storage space consumption on SServers. Fortunately, most file systems do not fully utilize the storage space in the underlying devices. Steege et. al found that data centers typically utilize 50% of their storage capacity [39]. In a practical system, this issue is not frequently encountered since the capacities of current SSDs are increasing quickly. In the worst case, where there is a possibility of an SServer running out of space, we could use a data migration method to balance the storage space by moving data from SServers to HServers, so the remaining available space on SServers can be guaranteed for new incoming requests. This problem can also be addressed by selectively storing users' performance-critical data in a hybrid PFS, while storing the rest of the data in a traditional PFS on HServers.

While HARL is currently implemented for a single application, it can also apply to multiple applications with varying I/O workloads. We identify the I/O access patterns at the MPI file level, and do not distinguish between requests coming from the same application or from different applications. For the latter case, we may apply our method on different workloads separately to find their individual data access patterns.

V. Conclusions

In this study, we have proposed a heterogeneity-aware region-level (HARL) data layout scheme, which distributes data across HDD and SSD file servers with the consideration of application workload and server I/O performance. HARL divides a file into fine-grained regions with similar workloads, and adopts varied stripe sizes on HServers and SServers for each region based on the storage performance of the servers. We have developed and presented the HARL layout optimization scheme, which includes dividing the file into fine-grained regions, determining the stripe sizes of HServers and SServers in each region, and implementing the optimal layout scheme under a runtime system. In essence, HARL provides improved matching of data access characteristics of applications with data access capabilities of file servers in a hybrid PFS. Experimental results with representative benchmarks show that HARL is promising and a viable solution for hybrid PFSs: the I/O performance improves from 20.6% to 556.1% for reads and 22.7% to 288.7% for writes.

In the future, we would like to extend our cost model to accommodate more than two server performance profiles. Another direction is to explore on-line data layout and data migration methods to make heterogeneous I/O systems more intelligent and efficient.

Acknowledgment

The authors are thankful to Xin Huang, Bo Feng, Kun Feng, and Ning Liu for their help toward this work. This study is supported in part by the Natural Science Foundation of Hubei Province of China under Grant No. 2014CFB239, the

Openfund from HPCL under Grant No.201512-02, the National Science Foundation of China under Grant No. 61373040 and 61173137, the Ph.D. Programs Foundation of Ministry of Education of China under Grant No.20120141110073, the US National Science Foundation under Grant CCF-0937877 and CNS-1162540.

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] D. Nagle, D. Serenyi, and D. Serenyi, "The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, 2004.

[5] 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.

[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] 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.

[9] "Interleaved Or Random (IOR) Benchmarks." [Online]. Available: http://sourceforge.net/projects/ior-sio/

[10] 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.

[11] "Application I/O Traces: Anonymous LANL App2," http://institutes.lanl.gov/plfs/maps/, 2014.

[12] A. N. Lab, "MPICH2 : A High Performance and Widely Portable Implementation of MPI." [Online]. Available: http://www.mcs.anl.gov/research/project-detail.php?id=2

[13] 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.

[14] 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.

[15] 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.

[16] 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 *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–12.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] T. Cortes and J. Labarta, "Taking Advantage of Heterogeneity in Disk Arrays," *Journal of Parallel and Distributed Computing*, vol. 63, no. 4, pp. 448–464, 2003.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] X. Wu and A. N. Reddy, "Exploiting Concurrency to Improve Latency and Throughput in a Hybrid Storage System," in *Proceedings of IEEE International Symposium onModeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010, pp. 14–23.

[31] 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.

[32] 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.

[33] S. He, Y. Liu, and X.-H. Sun, "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.

[34] S. He, X.-H. Sun, and A. Haider, "HAS: Heterogeneity-Aware Selective Data Layout Scheme for Parallel File Systems on Hybrid Servers," in *Proceedings of 29th IEEE International Parallel and Distributed Processing Symposium*, 2015.

[35] Y. Yin, J. Li, J. He, X.-H. Sun, and R. Thakur, "Pattern-Direct and Layout-Aware Replication Scheme for Parallel I/O Systems," in *Proceedings of 27th IEEE International Parallel and Distributed Processing Symposium*, 2013.

[36] 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.

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

[38] Y. Yin, S. Byna, H. Song, X.-H. Sun, and R. Thakur, "Boosting Application-Specific Parallel I/O Optimization Using IOSIG," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2012, pp. 196–203.

[39] P. Steege, "50% Storage Utilization: Are Data Centers Half Empty or Half Full?" http://storageeffect.media.seagate.com/2009/01/storage-effect/50-storage-utilization-are-datacenters-half-empty-or-half-full/, 2014.