Enhancing Hybrid Parallel File System through Performance and Space-Aware Data layout

Journal:	*International Journal of High Performance Computing*
Manuscript ID	hpc-15-0011.R1
Manuscript Type:	Full Length Article
Date Submitted by the Author:	08-Oct-2015
Complete List of Authors:	He, Shuibing; Wuhan University, Computer School ; Illinois Institute of Technology, Department of Computer Science Liu, Yan; Hunan University, College of Computer Science and Electronic Engineering Wang, Yang; Chinese Academy of Science, Shenzhen Institute of Advanced Technology Sun, Xian-He; Illinois Institute of Technology, Department of Computer Science
Keywords:	Parallel I/O System, Parallel File system, Data Layout, Solid State Drive, Hybrid I/O System
Abstract:	Hybrid parallel file systems (PFS), which consist of SSD servers (SServer) and HDD servers (HServer), have recently attracted growing attention. Compared to a traditional HServer, an SServer consistently provides improved storage performance but lacks storage space. However, most current data layout schemes do not consider the differences in performance and space between heterogeneous servers, and may significantly degrade the performance of the hybrid PFSs. In this paper, we propose PSA, a novel data layout scheme, which maximizes the hybrid PFS's performance by applying adaptive varied-size file stripes. PSA dispatches data on heterogeneous file servers not only based on storage performance but also storage space. We have implemented PSA within OrangeFS, a popular parallel file system in the HPC domain. Our extensive experiments with representative benchmarks, including IOR, HPIO, MPI-TILE-IO, and BTIO, show that PSA provides superior I/O throughput than the default and performance-aware file data layout schemes.

Enhancing Hybrid Parallel File System through Performance and Space-Aware Data layout

Shuibing He[†*], Yan Liu[‡], Yang Wang[§], Xian-He Sun[*]
[†]Computer School, Wuhan University, Wuhan, Hubei, China
[‡] College of Computer Science and Electronic Engineering, Hunan University, China
[§]Shenzhen Institute of Advanced Technology, Chinese Academy of Science, Shenzhen, China
[*]Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA
heshuibing@whu.edu.cn, liuyan@hnu.edu.cn, yang.wang1@siat.ac.cn, sun@iit.edu

Abstract

Hybrid parallel file systems (PFS), which consist of SSD servers (SServer) and HDD servers (HServer), have recently attracted growing attention. Compared to a traditional HServer, an SServer consistently provides improved storage performance but lacks storage space. However, most current data layout schemes do not consider the differences in performance and space between heterogeneous servers, and may significantly degrade the performance of the hybrid PFSs. In this paper, we propose PSA, a novel data layout scheme, which maximizes the hybrid PFSs performance by applying adaptive varied-size file stripes. PSA dispatches data on heterogeneous file servers not only based on storage performance but also storage space. We have implemented PSA within OrangeFS, a popular parallel file system in the HPC domain. Our extensive experiments with representative benchmarks, including IOR, HPIO, MPI-TILE-IO, and BTIO, show that PSA provides superior I/O throughput than the default and performance-aware file data layout schemes.

Index Terms

Parallel I/O System; Parallel File system; Data Layout; Solid State Drive; Hybrid I/O System

I. Introduction

Many large scale applications in science and engineering have become more and more data-intensive [1]. For example, Table I shows the data requirements of a few representative applications at Argonne National Laboratory in 2012 [2]. The generated data of these applications reach several terabytes per year. Such large data requirements are putting unprecedented pressure on computer I/O systems to store data effectively. In the meanwhile, storage devices have a slower performance improvement than CPUs during the past three decades. While processor speeds have increased nearly 50% each year, the access latency of a single hard disk drive (HDD) has only reduced by roughly 7% [3]. As a result, I/O system has become the major performance bottleneck for many applications in high performance computing (HPC) domain.

To accommodate growing volumes of data, parallel file systems (PFS), such as PVFS [4], OrangeFS [5], Lustre [6], and GPFS [7], have been developed to achieve high aggregate I/O throughput by leveraging the parallelism of multiple HDD-based file servers. However, due to the diversity of data access patterns, it is not always enough to improve I/O performance by simply adding more storage servers. For example, in the face of non-contiguous small requests, PFSs still perform poorly [8] because of the low degree of server parallelism as well as the frequent disk head movements on each server due to random accesses [9]. Hence, fully utilizing the underlying file servers is still a challenging task.

New emerging storage technologies, such as flash based solid state drives (SSD), have received widespread attention in revolutionizing I/O system design [10]. SSDs have orders of magnitude higher performance, lower power consumption, and a smaller thermal footprint over traditional HDDs [11]. While SSDs are ideal storage media for PFSs in terms of performance, it is not an economical option to completely deploy SSDs in a large-scale system due to the high costs of SSDs. Furthermore, HDDs still have several benefits in HPC domains, such as high capacity and decent peak bandwidth for large sequential requests. Therefore, hybrid storage systems, which consist of HDD-based file servers (HServer) and SSD-based file servers (SServer), provide practical solutions for data-intensive applications [8], [12]–[15].

These hybrid systems are a cost effective way to deliver capacity and bandwidth which is important to nearly all classes of systems. Generally SServers can be used as a cache tier [8] or a regular storage tier [12], [14] in a hybrid storage system. The caching architecture and the one-tiered architecture are suitable for different system configurations and application access patterns. For example, the caching architecture performs well for small random requests and high-end SServers, but fail to fully utilize of I/O parallelism from multiple servers for large accesses and ordinary SServers. Detailed comparison of these architectures is out of the scope of this paper. In this work, we focus on the one-tiered architecture, where both SServers and HServers work as storage servers of a hybrid PFS.

The effectiveness of a hybrid PFS relies on the effectiveness of its data layout scheme. In existing parallel file systems, the stripe-based data layout approach is commonly used to distribute data among available file servers. This traditional scheme dispatches a large file across multiple servers with a fixed-size stripe in a round-robin fashion. While resulting in even data placement on servers, this scheme will lead to decreased I/O performance because of the non-uniform access distribution in the

TABLE I
DATA REQUIREMENTS FOR SELECT 2012 INCITE APPLICATIONS AT ARGONNE LEADERSHIP COMPUTING FACILITY OF ANL [2].

Project	**On-line** (TBytes)	**Off-line** (TBytes)
Supernovae Astrophysics	100	400
Combustion in Reactive Gases	1	17
CO2 Absorption	5	15
Seismic Hazard Analysis	600	100
Climate Science	200	750
Energy Storage Materials	10	10
Stress Corrosion Cracking	12	72
Nuclear Structure and Reactions	6	30
Reactor Thermal Hydraulic Modeling	100	100
Laser-Plasma Interactions	60	60
Vaporizing Droplets in a Turbulent Flow	2	4

workloads [16]. To alleviate this issue, numerous strategies have been studied on data layout optimization, such as adjusting the file stripe sizes to rearrange loads among servers [9], [17] or optimizing the file stripe distribution method according to the data access patterns [18]. However, these approaches mainly focus on homogeneous PFSs with identical HServers, and may not work well in hybrid PFSs due to the following reasons.

First, the storage performance of each file server is not differentiated in existing layout schemes. HServer and SServer can have different storage performance behaviors due to their distinct internal structure [11]. A high-speed SServer can finish storing data in a local SSD faster than a low-speed HServer; thus, HServer is often the straggler in the service of a large file request in a parallel environment. When directly applied to the hybrid PFSs, existing layout schemes will result in severe load imbalance among file servers even under uniform workloads, which can significantly degrade the performance of the hybrid I/O system.

Second, traditional layout schemes have an assumption that each file server has an identical, sufficient storage space to accommodate file data. However, most current SSDs have relatively smaller capacities than HDDs because they're more expensive [19]. Existing data layout schemes focus on promoting the performance balance among servers with little attentions paid on the storage space balance [14]. Consequently, SSDs may quickly run out of their limited space when more data are dispatched on them. These one-sided designs may have hidden flaws that may impair their potential effectiveness for improving the overall I/O performance for prolonged time.

In this paper we propose PSA, a performance and space-aware data layout scheme which carefully arranges data layout to improve the hybrid PFS performance. Unlike traditional schemes, PSA distributes file data on different types of servers using adaptive stripe sizes. Additionally, PSA dispatches large file stripes on HServers than SServers, so that more file requests are allowed to be served by both HServers and SServers rather than only HServers within a given SSD capacity. Since the two types of servers working together are likely to provide better I/O performance than HServers, PSA leads to substantial performance improvement for all file requests. The proposed data layout scheme creates a better balance between storage performance and space of heterogeneous file servers, and can be extended to systems with various types of file servers, system configuration, and I/O patterns.

Specifically, we make the following contributions.

- We find that performance-aware data layout policy can only obtain sub-optimal performance of hybrid PFSs with low-capacity SServers.
- We introduce a cost model to evaluate file request completion time on heterogeneous HServers and SServers, and file request completion time only on homogeneous HServers.
- Based on the proposed cost model, we propose a performance and space-aware algorithm to determine the appropriate file stripe sizes for HServer and SServers in a hybrid PFS.
- We implement the prototype of the PSA scheme under OrangeFS, and have conducted extensive tests to verify the benefits of the PSA scheme. We evaluate PSA with representative benchmarks, including IOR, HPIO, MPI-TILE-IO, and BTIO. Experiment results illustrate that PSA can significantly improve the hybrid I/O system performance.

The rest of this paper is organized as follows. Background and motivation are presented in Section II. The design and implementation of PSA are described in section III. Section IV presents the performance evaluation. Section V discusses the

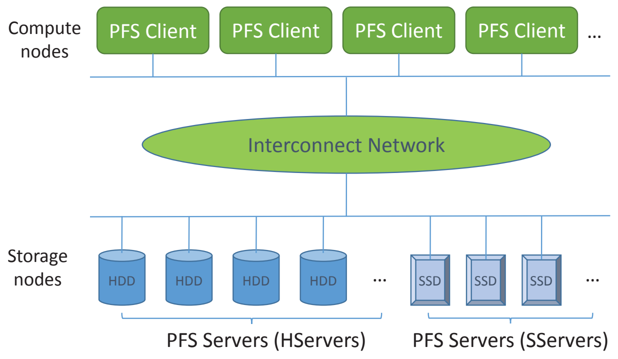


Fig. 1. The architecture of a hybrid parallel file system

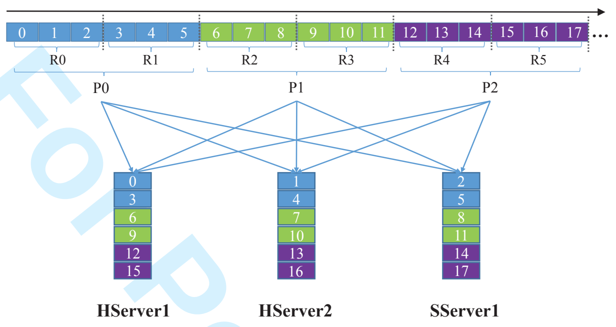


Fig. 2. Traditional data layout scheme with fixed-size stripe on file servers

related work, and Section VI concludes the paper.

II. Background and Motivation

We first introduce our target environment for the proposed layout scheme. Next, we introduce the traditional layout policies and the corresponding problems when they are applied to hybrid PFSs.

A. *The Hybrid Parallel File System Architecture*

A PFS mainly includes three components: clients, servers and metadata servers (MDS). A client provides a file interface for users to access data, a server serve data to the rest of the cluster, and a MDS contains information about the data distribution on the servers. Upon a file operation, a file client first contacts MDS to get the file metadata information, then interacts with file servers directly. A PFS can have one or multiple MDSs and each MDS can locate either on the same physical node as file server or on a separate node.

Figure 1 shows the system architecture of a typical hybrid parallel file system (The MDSs are not depicted), which includes two kinds of file servers: HDD-based file server represented as HServers and SSD-based file server as SServers. Generally, SServers have relatively higher storage performance than HServers. This hybrid architecture provides promising solutions for cost-constrained storage systems. However, the potential benefits cannot be realized unless we carefully consider the file data layout in the hybrid PFSs.

B. *Traditional Fixed-size Striping Data Layout Schemes*

In order to keep pace with the processing capabilities in HPC clusters, PFSs such as PVFS [2] and Lustre [3] are designed to improve I/O performance. In PFSs, data layout schemes are responsible for defining the way a file's data are distributed on the underlying servers. Files in PFSs are often organized in fixed-sized stripes, and they are dispatched onto the underlying servers in a round-robin fashion.

Figure 2 illustrates the idea of the traditional data layout in PFSs. In this example, a file's data are placed on the three server with a fixed-size stripe. This policy can provide an even data placement in each file server as well as good I/O performance for many data access patterns in tradition parallel file systems. As this layout scheme is relatively simple and effective, they are widely used in many PFSs. For example, in OrangeFS, this scheme is the default layout method named "*simple striping*".

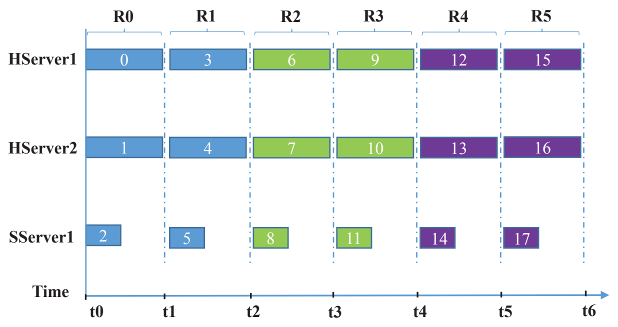


Fig. 3. The I/O times of sub-requests on different file servers. With fixed-size file stripe, SServer will finish their sub-requests faster than HServer, leading to significant load imbalance among file servers.

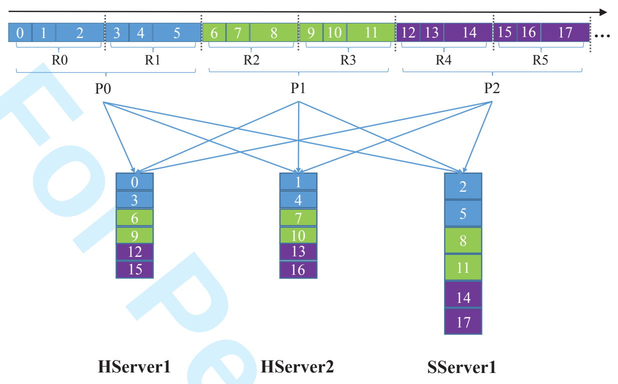


Fig. 4. Performance-aware data layout scheme. High-speed HServers are assigned with larger stripes, so that all servers finish their sub-requests almost simultaneously. However, SServers run out their limited space quickly and the remaining requests will be served only by HServers .

C. *Motivation Example*

Although traditional data layout strategies are suitable for homogeneous PFSs, they may significantly degrade the overall I/O performance of hybrid PFSs. Figure 2 demonstrates a representative example of a typical file access pattern in current HPC systems with the fixed-size file striping data layout. For simplicity, we assume to have three processes (P0-2), six file requests (R0-5), and each process has two requests. We also assume that there are two HServers and one SServer. To be specific, we assume each request size is three times the file stripe size, so that all servers can contribute to the overall I/O performance.

By the default fixed-size layout method, each request is divided into three sub-requests. For example, R0 is served by sub-request #0-2 and R1 by sub-request #3-5, as shown in Figure 2. While each sub-request has the same size, their I/O completion time is significantly different due to the existing performance disparity between HServers and SServers. For example, I/O time of sub-request #2 is smaller than that of sub-request #0, as shown in Figure 3. Because I/O completion time of a request is determined by the slowest sub-request, each request time equals that of the sub-request's time on HServer. As we can see, due to the existing file striping assignment, each SServer continues to stay idle in the service of file requests, which results in severe I/O performance degradation.

There exists a possible solution to overcome this problem by taking the file server performance into account when deciding the stripe size of each file server [14]. As illustrated in Figure 4, by assigning SServer with a larger stripe than HServers, all servers can finish their sub-requests simultaneously and the load imbalance is alleviated. However, some SSDs often have relatively smaller storage space than HDDs. The one-sided design will make SSDs quickly run out of their limited space; thus all the remaining requests are only served by the low-speed HServers, which can provide relatively low I/O performance. To show the difference of underlying servers carrying out the requests, we categorize all file requests in a hybrid file system into two types: *heterogeneous* and *homogeneous*. A heterogeneous request refers to the case where the request is served by both HServers and SServers; a homogeneous request refers to the one served only by the slow HServers. Generally, heterogeneous requests have better I/O performance than *homogeneous* requests because they are involved in more storage servers. Therefore, if we can increase the ratio of heterogeneous requests over all file requests through optimized data layouts, the overall file system performance can be largely improved.

III. Design and Implementation

In this section, we first introduce the basic idea of the proposed data layout scheme. Then we describe the cost model and algorithm used to determine the optimal stripe size for each server. Finally, we present the implementation of PSA.

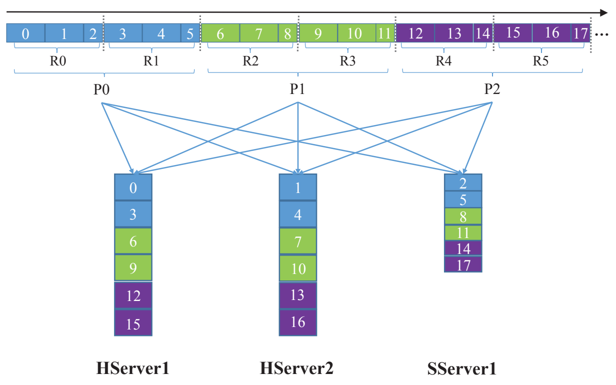


Fig. 5. Performance and space-aware data layout scheme. SServers are assigned with smaller stripes, so that there are more requests served by . With a given SSD capacity, the overall I/O performance of file requests can be improved.

A. The Basic Idea of PSA

The proposed data layout scheme (PSA) aims to improve hybrid PFSs with performance and space-aware adaptive file stripes. Instead of assigning SServers with larger file stripes as performance-aware strategy [14], the basic idea of PSA is to assign HServer with larger file stripes and SServers with smaller stripes. Since the storage space of SServer is consumed more gradually, this layout scheme can lead to more heterogeneous requests from the given client's data accesses. As a result, we can get the globally optimized I/O performance for all file requests rather than the local optimization for certain requests.

As explained previously, we assume that HServer has enough space to accommodate data and SServer only has limited space for file requests. Figure 5 illustrates the file data distribution on the underlying servers after we assign the stripe sizes for HServers and SServers using our strategy. To show the performance comparison, we assume that there are 20 file requests from clients. For performance-aware data layout scheme (PA), we assume each SServer has space for 6 sub-requests, as shown in Figure 4. Thus, there are 6 heterogeneous requests and 14 homogeneous requests among all requests. For the proposed performance and space-aware scheme (PSA), we assume each SServer can absorb 20 sub-requests as each SServer is allocated with smaller stripe size. In this case, all file requests belong to heterogeneous requests. While the performance of heterogeneous file requests in PSA layout cannot be better than that of PA, PSA leads to a large number of heterogeneous file requests. We assume the I/O time for heterogeneous requests under PA and PSA strategy is $2T$ and $4T$ respectively, and the I/O time for homogeneous requests in PA is $8T$, then the overall I/O time for all requests under PA and PSA strategy is $2T \times 6 + 8T \times 14 = 124T$ and $4T \times 20 = 80T$, respectively. This validates that PSA can improve the overall file system performance.

It is worth noting that in our strategy, it is not necessary to require all file requests to be served by both HServers and SServers. We only attempt to increase the number of heterogeneous requests, when there is a possibility of performance optimization.

In practice, determining the appropriate file stripe sizes based on storage performance and space is not easy for several reasons. First, the performance of each file server can be impacted significantly by both I/O patterns and storage media. Even under the same I/O patterns, HServer and SServer can have different performance behaviors. Second, for given file requests and storage space on each SServer, different stripe sizes will result in various heterogeneous request and homogeneous request ratios, which can largely impact the overall parallel file system performance.

B. An Analytic Data Access Cost Model

To identify the optimal data layout with appropriate stripe sizes for HServers and SServers, we built an analytical cost model to evaluate the data access time in a parallel computing environment. The critical parameters are listed in Table II. Since SServers and HServers have distinct storage media, they exhibit different storage characteristics. First, T_s for SServer is much smaller than HServer's. Second, β_s is several times smaller than β_h, which means SServers have a performance advantage over HServers for large requests, but not as significant for small requests. Finally, write performance of SServer is lower than read performance because write operations on SSDs lead to numerous background activities, including garbage collection and wear leveling. Due to these device-aware critical parameters, the cost model can effectively reflect the performance of requests on various types of file servers.

The cost is defined as the I/O completion time of a file request in a hybrid PFS, which mainly includes two parts: the network transmission time, T_{NET}, and the storage access time, T_{STO}. Generally, T_{NET} consists of T_E, which is the network connection for data transmission, and T_X, which is the data transferring time on network. T_{STO} consists of T_S and T_T, the former is the startup time on server, and the latter is the actual data read/write time on storage media.

Since both heterogeneous requests and homogeneous requests may exist in hybrid PFSs due to the limited space of SServers, we calculate their I/O costs respectively. For heterogeneous file requests, we use a previous cost model [14] to evaluate the

TABLE II
PARAMETERS IN COST ANALYSIS MODEL

Symbol	Meaning
p	Number of client processes
c	Number of processes on one I/O client node
m	Number of HServers
n	Number of SServers
h	Stripe size on HServer
s	Stripe size on SServer
S	Data size of one request
e	Cost of single network connection establishing
t	Network transmission cost of one unit of data
α_h	Startup time of one I/O operation on HServer
β_h	HDD transfer time per unit data
α_s	Startup time of one I/O operation on HServer
β_s	SSD transfer time per unit data

TABLE III
DATA ACCESS COST FOR HETEROGENEOUS REQUESTS ON BOTH HSERVERS AND SSERVERS

Condition	Network cost T_{NET}		Storage cost T_{STO}
	Establish T_E	Transfer T_X	Startup T_S+R/W T_T
$p \leq c(m+n)$	$c(m+n)e$	$\max\{cSt, pht, pst\}$	$p * \max\{\alpha_h + h\beta_h, \alpha_s + s\beta_s\}$
$p > c(m+n)$	pe	$\max\{cSt, pht, pst\}$	$p * \max\{\alpha_h + h\beta_h, \alpha_s + s\beta_s\}$

data access cost. In this model, we assume the requests are fully distributed on all HServers and Servers as Figure 5, namely $m \times h + n \times s = S$, then the cost model can be calculated as in Table III. More details about constructing the data access cost can be found in our previous research [14].

For homogeneous file requests, since the previous model [14] does not work for them, we introduce a new cost model to calculate their data access times. In this case, we assume these requests are distributed only on all HServers with a stripe size of S/m. We choose this stripe configuration because it leads to good load balance among file servers as well as optimal overall I/O performance. With these assumptions, the corresponding cost is defined as the formulas in Table IV. Hence, our model considers the space limitation of SServer and can accommodate diverse file requests.

C. *Optimal Stripe Size for File Servers*

Based on the proposed cost model, we devise a heuristic iterative algorithm to determine the appropriate stripe sizes for HServers and SServers, as displayed in Algorithm 1. Starting from s_h equaling $S/(M+N)$, the loop iterates s_h in 'step' increments while s_h is less than S/M. Different from previous work [14] where SServer serves larger sub-requests, this configuration intends to make SServer serves smaller sub-requests so that SServer can contribute more file requests to improve the overall I/O performance. The extreme configuration we do consider is where h is S/M, which means dispatching file request data only on HServers may obtain better I/O performance. For each stripe size pair for HServers and SServers, namely $< s_h, s_s >$, the loop iterates to calculate the total access cost of all file requests, either with formulas in Table III if they are distributed on both HServers and SServers, or with formulas in Table IV to calculate if they are distributed only on HServers.

TABLE IV
DATA ACCESS COST FOR HOMOGENEOUS REQUESTS ON HSERVERS

Condition	Network cost T_{NET}		Storage cost T_{STO}
	Establish T_E	Transfer T_X	Startup T_S+R/W T_T
$p \leq cm$	cme	$\max\{cSt, pSt/m\}$	$p\alpha_h + ph\beta_h$
$p > cm$	pe	$\max\{cSt, pSt/m\}$	$p\alpha_h + ph\beta_h$

Finally, the stripe size pair that leads to minimal total data access cost is chosen. The ‘step’ value, as shown in line 3 of Algorithm 1, is 4KB. The user can choose finer ‘step’ values resulting in more precise S_h and S_s values, but with increased cost calculation overhead. However, computational overhead for executing this algorithm is acceptable because the calculations are simply arithmetic operations and run off-line.

Algorithm 1: Stripe Size Determination Algorithm

```
Input  : File requests: $R_0, R_1, ..., R_{k-1}$, SServer Capacity: $C_s$,
Output: optimal stripe sizes: $S_H$ for HServer, $S_S$ for SServer
1  $l \leftarrow \frac{S}{m+n}$;
2  $h \leftarrow \frac{S}{m}$;
3  $step \leftarrow 4KB$;
4  for $s_h \leftarrow l; s_h \leq h; s_h \leftarrow s_h + step$ do
5      $s_s \leftarrow (S - m * s_h)/n$;
6      $j \leftarrow \frac{C_s}{S_s}$ /*j is the number of heterogeneous requests*/;
7      for $i \leftarrow 0; i < k; i \leftarrow i + 1$ do
8          Determine request type of $R_i$ based on its offset, length, and $j$;
9          if $R_i$ is a heterogeneous request then
10             /* $R_i$ is distributed on $m + n$ servers */;
11             $T_i \leftarrow$ Calculate the access cost of $R_i$ according to the formulas in Table III ;
12         else
13             /* $R_i$ is distributed only on $m$ HServers*/;
14             $T_i \leftarrow$ Calculate the access cost of $R_i$ according to the formulas in Table IV ;
15         end
16         $Total\_cost \leftarrow Total\_cost + T_i$;
17     end
18     if $Total\_cost < Opt\_cost$ then
19         $Opt\_cost \leftarrow Total\_cost$;
20         $S_H \leftarrow s_h$;
21         $S_S \leftarrow s_s$;
22     end
23 end
```

Once the optimal stripe sizes for HServers and SServers are determined, PFSs can distribute file data with the optimal data layout to improve the hybrid PFSs performance.

D. Implementation

The proposed stripe size optimization requires a prior knowledge of applications’ access characteristics. Fortunately, many HPC applications access their files with predictable I/O patterns and they often run multiple times [20]–[22]. For instance, BTIO [23], an I/O kernel responsible for solving block-tridiagonal matrices on a three dimensional array, is one of these applications. 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 accesses can be accurately predicted before the program executes. This feature provides an opportunity to achieve the performance and space-aware data layout based on I/O trace analysis.

Based on the above observation, we implemented the proposed PSA data layout scheme in OrangeFS [5] with I/O access analysis. OrangeFs is a new branch of the PVFS2 file system [4], which is widely used in the HPC domain. Figure 6 shows the procedure of the optimal data layout scheme, which mainly consists of three phases: *estimation*, *determination*, and *placement*.

In the *estimation* phase, we obtain the related parameters in the cost model. For a given system, the network parameters, such as e and t, the storage parameters, such as α_h,β_h,α_s,β_s, and the system parameters, such as m and n can be regarded as constants. We use one file server in the parallel file system to test the storage parameters for HServers and SServers with sequential/random and read/write patterns. We use a pair of one client node and one file server to estimate the network parameters. All these tests are repeated thousands of times, and we use the average values in the cost model.

In the *determination* phase, we use a trace collector to obtain the run-time statistics of data accesses during the application’s first execution. Based on the I/O trace, we obtain the application’s I/O pattern related parameters, such as c, p, and S. Combined with the parameters obtained in the estimation phase, we use the cost model and Algorithm 1 to determine the optimal file stripe sizes for HServers and SServers.

In the *placement* phase, we distribute the file data with the optimal data layout for later runs of the applications. The OrangeFS file system supports an API for implementing specific variable stripe distribution. The variable stripe distribution is similar to simple stripe, except the stripe size can be configured differently on various file servers. 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 of directories where the application files are located. When a new file is created, we use the “*pvfs2-touch*” command with the “-l” option to specify the order of the file servers, so that the proper file stripe size can be applied to the

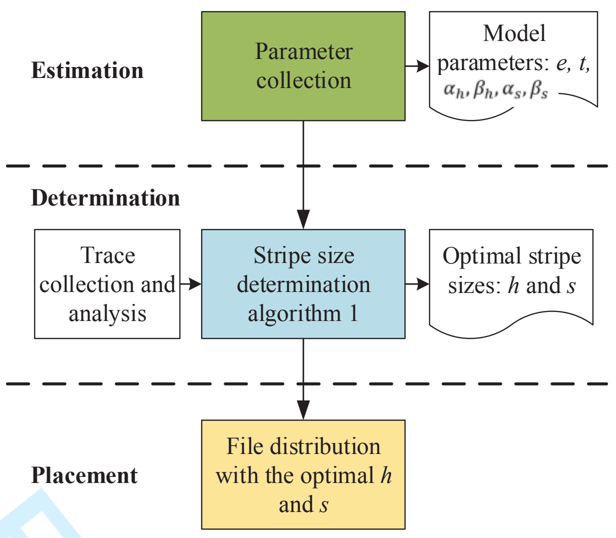

Estimation
Parameter collection
Model parameters: e, t, $\alpha_h, \beta_h, \alpha_s, \beta_s$
Determination
Trace collection and analysis
Stripe size determination algorithm 1
Optimal stripe sizes: h and s
Placement
File distribution with the optimal h and s

Fig. 6. The procedure of the performance and space-aware data layout scheme

corresponding file servers. For POSIX interface, we use the "*setfattr*" command to reach the similar data layout optimization goal.

IV. Performance Evaluation

In this section, we evaluate the performance of the proposed data layout scheme with benchmark-driven experiments.

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 parallel file system is OrangeFS 2.8.6.

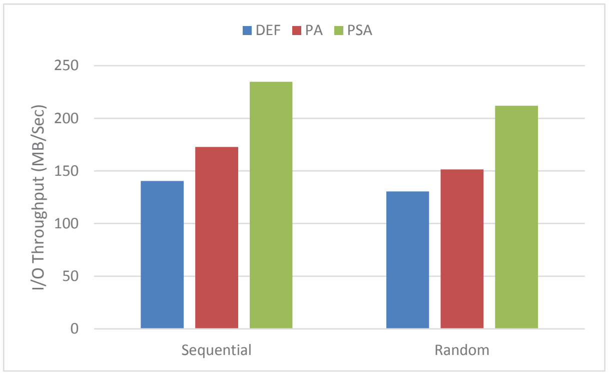
Among the available nodes, we select eight as client computing nodes, eight as HServers, and eight as SServers. By default, the hybrid OrangeFS file system is built on four HServers and four SServers. As discussed, a parallel file will be divided into two parts if SServers run out of space. The first part is distributed on all file servers, and the other part is placed only on HServers. We compare PSA with other two data layout schemes: the default scheme (DEF) and the performance-aware scheme (PA). In DEF, the first part of the file is placed across all servers with a fixed-size stripe of 64KB; in PA, the stripe sizes for HServers and SServers in the first part of the file are determined by storage performance as discussed in [14]. For the second part of the file, all schemes distribute the file on HServers with a stripe size of S/m, where S is the request size and m denotes the number of HServers.

We use the popular benchmark IOR [24], HPIO [25], MPI-TILE-IO [26], and BTIO [23], to test the performance of the parallel file system. For simplicity, we will use stripe size pair <h, s> to denote that the stripe sizes on HServers and SServers are h and s respectively.

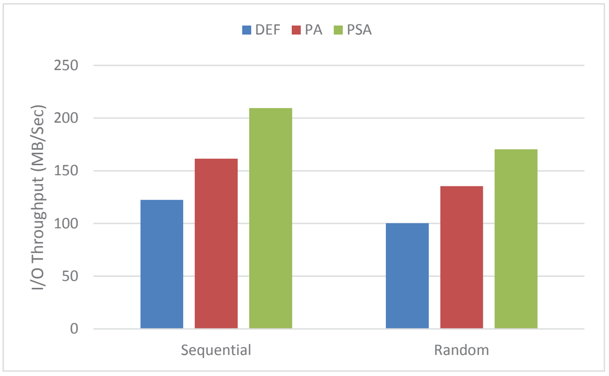
B. The IOR Benchmark

IOR is a parallel file system benchmark providing three APIs, MPI-IO, POSIX, and HDF5. We only use MPI-IO interface in the experiments. Unless otherwise specified, IOR runs with 16 processes, each of which performs I/O operations on a 16GB shared file with request size of 512KB. To simulate the situation that SServers have relatively smaller space than HServers, we limit the storage space of each SServer to 1GB.

1) Different Type of I/O Operations: First we test IOR with sequential and random read and write I/O operations. From Figure 7, we observe that PSA has optimal I/O performance compared to the other data layout schemes. By using the optimal stripe sizes for HServers and SServers, PSA improves read performance up to 66.9% over DEF with all I/O access patterns, and write performance up to 77.1%. Compared with PA, PSA improves the performance up to 39.8% for reads and 29.7% for writes. For PA, the optimal stripe sizes for sequential and random read and write, are <28KB, 100KB>, <20KB, 108KB>, <24KB, 104KB>, and <36KB, 92KB> respectively. For PSA, the optimal stripe sizes for sequential and random read and write, are <120KB, 8KB>, <120KB, 8KB>, <116KB, 12KB>, and <120KB, 8KB> respectively. This demonstrates both PA and PSA schemes adopt various file stripes for different I/O operations. However, by allocating small stripe sizes for SServers, PSA can makes better trade-off between SSD's performance and space to improve the overall I/O performance. PSA's read performance exceeds its write performance because SSDs performs better for read operations than write, as described in Section III-B. The experiments prove PSA performs optimally and the stripe size determining formula is effective.

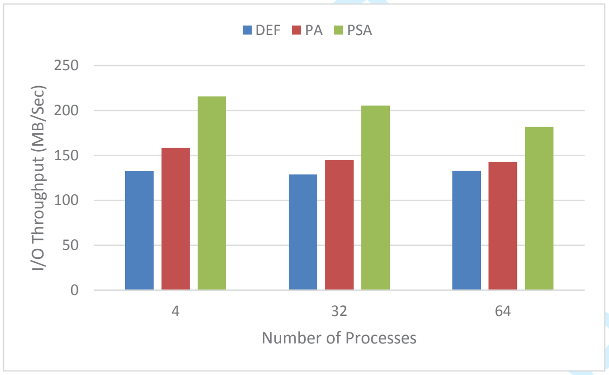


(a) Read throughput

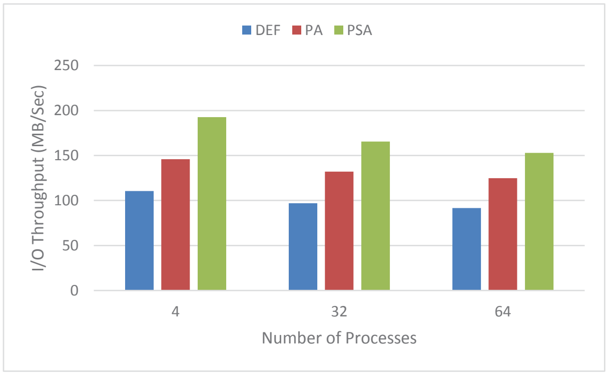


(b) Write throughput

Fig. 7. Throughputs of IOR under different layout schemes with different I/O modes



(a) Read throughput



(b) Write throughput

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

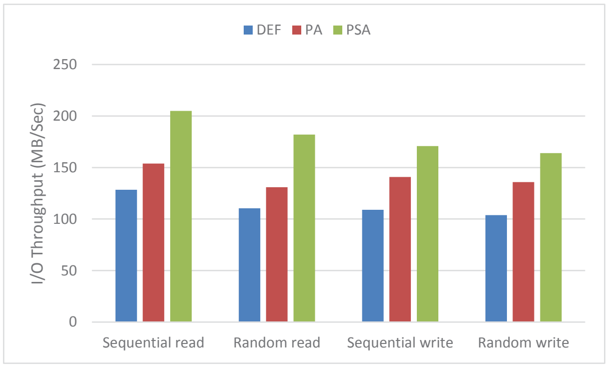
2) Different Number of Processes: The I/O performance is also evaluated with different number of processes. The IOR benchmark is executed under the random access mode with 8, 32 and 64 processes.

As displayed in Figure 8, the result is similar to the previous test. PSA has the best performance among the three schemes. Compared with DEF, PSA improves the read performance by 62.8%, 59.5%, and 36.7% respectively with 4, 32 and 64 processes, and write performance by 74.3%, 70.9%, and 66.7%. Compared with PA, PSA improves read performance by 36.2%, 41.9%, and 27.1% with 4, 32 and 64 processes, and write performance by 32.1%, 25.4%, and 22.3%. As the number of processes increase, the performance of the hybrid PFS decrease because more processes lead to severer I/O contention in HServers. These results show that PSA has excellent scalability with the number of I/O processes.

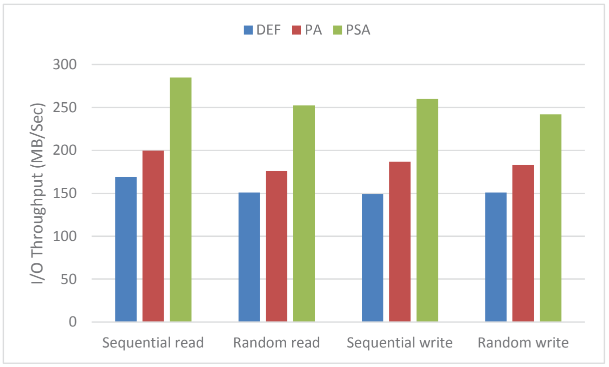
3) Different Request Sizes: In this test, the I/O performance is examined with different request sizes. The IOR benchmark is executed with request sizes of 128KB and 1024KB and the number of processes is fixed to 16. From Figure 9(a), we can observe that PSA can improve the read performance by up to 68.7%, and write by up to 74.4% in comparison with DEF scheme. Compared with PA, PSA also has better performance: the read performance is increased by up to 43.4%, and write performance is increased by up to 38.9%. We also find that PSA provides higher performance improvement for large request size because large requests benefit more from both HServers and SServers than only HServers. These results validate that PSA can choose appropriate stripe sizes for HServers and SServers when facing different request sizes.

4) Different Server Configurations: The I/O performance is examined with different ratios of SServers to HServers. The OrangeFS is built using HServers and SServers with the ratios of 5:3, and 3:5.

Figure 10 shows the I/O bandwidth of IORwith different file server configurations. Based on the results, PSA can improve I/O throughput for both read and write operations. When the ratio is 5:3, PSA improves the read and write performance by up to 58.6% and 68.2% respectively, when compared to DEF. Compared with PA scheme, PSA increases the read performance by 35.3%, and write performance by 28.6%. When the ratio is 3:5, we can observe that PSA has similar behavior. In the experiments, read and write performance improve as the number of SServers increase because the I/O performance of heterogeneous requests benefits more from more SServers. By using the optimal stripe sizes determined by the performance and space-aware layout method in this paper, PSA can significantly improve the hybrid file system performance with every file server configuration.

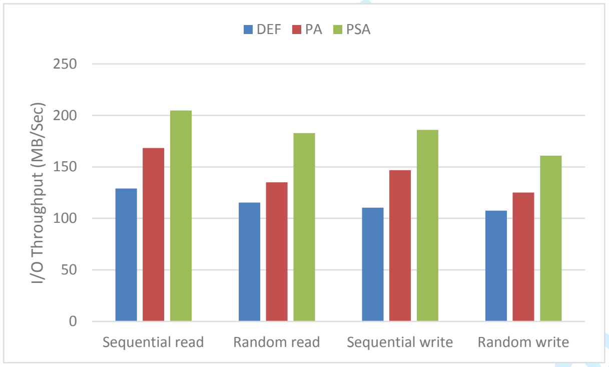


(a) Request size is 128K

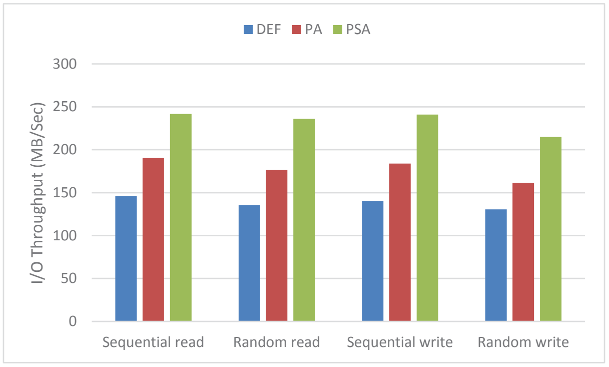


(b) Request size is 1024K

Fig. 9. Throughputs of IOR with varied request sizes



(a) 5HServers : 3SServers



(b) 3HServers : 5SServers

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

C. The HPIO Benchmark

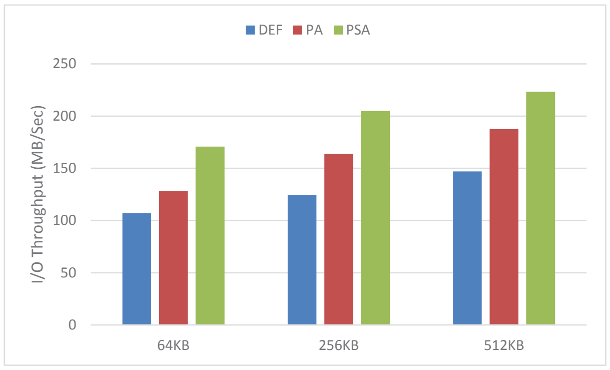
HPIO is a program developed by Northwestern University and Sandia National Laboratories to evaluate I/O system performance [25]. This benchmark can generate various data access patterns by changing three parameters: region count, region spacing, and region size. In our experiment, we set the number of process to 32, the region count to 1024, the region spacing to 0, and vary the region size from 64KB to 512KB. In addition, we build the file system on four HServers and four SServers, and we limit all SServers to contribute 1/3 storage space for all requests in each test to simulate the case where SServer can't provide enough space.

As shown in Figure 11(a), compared with DEF, PSA can increase read throughput by 59.4%, 64.5%, and 51.8% with request size 64KB, 256KB, and 512KB, respectively. In contrast to PA, PSA obtains a performance improvement of 33.2%, 25.1%, and 19.2% respectively. For write operations, PSA also exhibits performance benefits over DEF and PA scheme, as presented in Figure 11(b). This result means that PSA is effective with respect to HPIO benchmark. Similar to previous tests, PSA provides higher I/O performance for large region size (large request size) because large requests benefit more for both HServers and SServers.

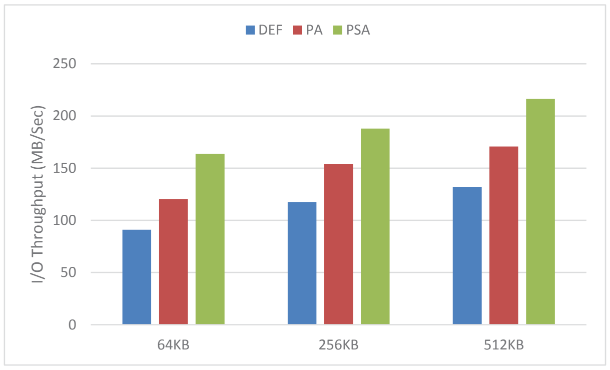
D. The MPI-Tile-IO Benchmark

MPI-Tile-IO is an MPI I/O test program designed by the Parallel I/O Benchmarking Consortium [26]. It treats the entire data file as a two-dimensional dense dataset and tests the performance of different data access patterns. Each process accesses a chunk of data based on the size of each tile and the size of each element. In the tests, we set the number of elements in the X and Y directions to 8 and 8, the size of each element to 32KB, and vary the number of processes between 100 and 400. We also set both overlap_x and overlap_y as 0, and perform collective I/O operations. Moreover, we configure the OrangeFS file system on four HServers and four SServers, and make SServers to hold 1/4 shared file data.

Figure 12 shows the aggregated I/O throughputs. Compared with DEF, PSA improves the read performance by 41.9%, 38.2%, and 34.1% respectively with 100, 200 and 400 processes, and write performance by 50.8%, 56.7%, and 59.2%. Compared with PA, PSA improves read performance by 28.2%, 30.8%, and 27.9% with 100, 200 and 400 processes, and write performance by 26.9%, 30.7%, and 32.2%. As the number of processes increase, the performance of the hybrid PFS decrease because more processes lead to severer I/O contention in HServers. However, PSA still has the best performance among the three schemes.

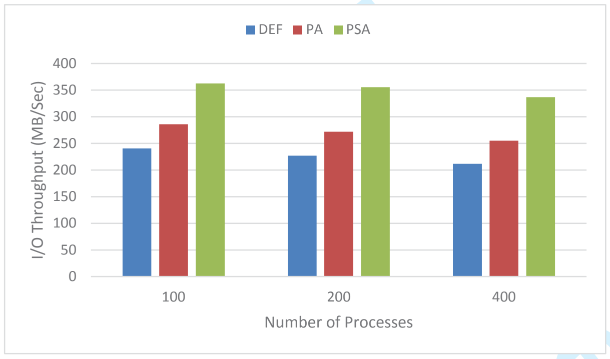


(a) Throughput for read

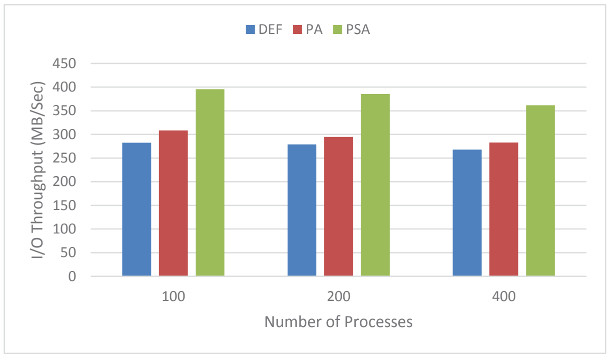


(b) Throughput for write

Fig. 11. Throughputs of HPIO with varied region sizes.



(a) Throughput for write



(b) Throughput for read

Fig. 12. Throughputs of MPI-Tile-IO with varied number of processes.

E. The BTIO benchmark

In addition to the benchmarks above, we use the BTIO benchmark [23] 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 Block-Tridiagonal (BT) partitioning pattern to solve the three-dimensional compressible Navier-Stokes equations.

We consider the Class C and full subtype BTIO workload in the experiments. That is, we write and read a total size of 6.64GB data with collective I/O functions for its IO operations. We use 4, 16, and 64 compute processes as BTIO requires a square number of processes. Output file is striped across six HServers and two SServers on the hybrid OrangeFS file system. In the experiments, we limit each SServer to hold 0.5GB file data.

As shown in Figure 11, compared to DEF and PA scheme, PSA achieves better throughput and scalability. Compared to DEF, PAS improves the performance by 66.1%, 47.7%, and 53.7% with 4, 16, 64 processes, respectively. For PA, PSA achieves the improvement by up to 38.9%. The experimental result confirms that PSA helps to improve hybrid file system performance.

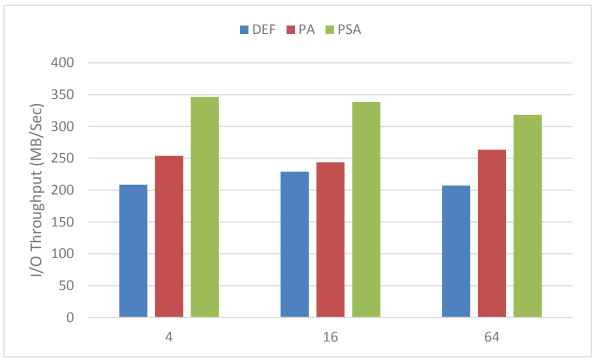


Fig. 13. Throughput of BTIO under varied number of processes

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

V. Related work

A. Data Layout in HDD-based File Systems

Data layout optimization is an effective approach to improve the performance of file systems. Parallel file systems generally provide several data layout strategies for different I/O workloads [18], including simple stripe, two dimensional stripe, and variable stripe. Widely adopted techniques for data partition [20], [27] and replication [18], [28], [29] are utilized to optimize data layout depending on I/O workloads.

Simple stripe layout schemes are unable to obtain high performance for applications that access I/O systems erratically. Segment-level layout scheme logically divides a file into several sections such that an optimal stripe size is assigned for each section with non-uniform access patterns [9]. Server-level adaptive layout strategies adopt various stripe sizes on different file servers to improve the overall I/O performance of parallel file systems [17]. PARLO utilizes various data layout polices to accelerate scientific applications with heterogeneous access patterns at I/O middleware layer [30]. However, these efforts are suitable for heterogeneous file servers. AdaptRaid addresses the load imbalance issue in heterogeneous disk arrays [31] with adaptive number of blocks, which can not be obtained in PFSs. Tantisiriroj *et al.* [32] use HDFS-specific layout optimizations [33] to improve the performance of PVFS. However, similar to most of previous works, this study is designed for homogeneous HDD-based file systems, and can't be applied to heterogeneous environments.

B. Data Layout in SSD-based File Systems

SSDs, which exhibit noticeable performance benefits over traditional HDDs, are commonly integrated into parallel file systems to improve I/O performance. Currently, most SSDs are used as a cache to traditional HDDs, e.g. Sievestore [34], iTransformer [35], and iBridge [36]. SSD-based hybrid storage is another popular method which utilizes the full potential of SSDs, such as I-CASH [37] and Hystor [38]. Yet, the vast majority of these techniques are done on a single file server.

For hybrid parallel I/O systems, our earlier work CARL [13] selects and places file regions with high access costs onto the SSD-based file servers at the I/O middleware layer. S4D-Cache [8] uses all SSD-based file servers as a cache and selectively caches performance-critical data on these high performance servers. HyCache+ [39] uses fast storage media, such as memory or SSD to provide a scalable high-performance caching middleware to improve the I/O performance of parallel file systems. Welch *et al.* place small files and file metadata onto SSDs, and large file extents onto HDDs [40]. However, in these studies the HDD-based and SSD-based file servers work independently and the parallelism of servers is hard to be utilized.

Meager amount of effort is devoted to data layout in a hybrid PFS, yet this knowledge is commonly needed when aging HDD file servers are replaced by new SSD-base file servers. PADP [14] uses varied-size stripes to improve the performance of hybrid PFSs, but the stripe sizes are only optimized for server storage performance. HAS [41] achieves an optimal data layout accounting for both application access patterns and server performance by choosing the least expensive layout from three typical layout candidates. These techniques are effective in improving the performance of hybrid PFSs, but they do not consider the space limitation of high-performance servers. Hybrid PFSs will lead issues of performance and space disparities between heterogeneous servers, and this work helps to deal with these challenges in hybrid storage architecture.

VI. Conclusions

With the availability of solid state drives (SSD), hybrid parallel file systems (PFS) have become common and promising in engineering practice. Compared to a traditional HDD, an SSD commonly has higher storage performance but smaller storage space. In this study, we have proposed a performance and space-aware data layout (PSA) scheme, which distributes data across HDD-based and SSD-based file servers with adaptive stripe sizes. PSA determines file stripe size on each server not only based on storage performance but also space. We have developed and presented the proposed PSA data layout optimization scheme in OrangeFS. In essence, PSA provides a better matching of data access characteristics of an application with the storage capabilities of file servers in a hybrid file system. Experimental results show that PSA is feasible and promising. PSA improves I/O performance by 36.7% to 74.4% over the default file data layout scheme, and it provides 20.6% to 43.5% better I/O performance than the performance-aware data layout scheme. In the future, we plan to extend the data layout scheme to any hybrid PFS with two or more file server performance profiles and complex I/O access patterns.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable feedback and comments, which substantially improved the quality of this paper. This work is supported in part by the National Science Foundation of China under Grant No.61572377, the Natural Science Foundation of Hubei Province of China under Grant No.2014CFB239, the Open-fund from HPCL under Grant No.201512-02, and the US National Science Foundation under Grant CNS-1162540.

References

[1] M. Kandemir, S. W. Son, and M. Karakoy, "Improving I/O performance of Applications through Compiler-DirectedCode Restructuring," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, 2008, pp. 159–174.

[2] 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.

[3] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Morgan Kaufmann, 2011.

[4] P. H. Carns, I. Walter B. Ligon, R. B. Ross, and R. Thakur, "PVFS: A Parallel Virtual File System for Linux Clusters," in *Proceedings of the 4th Annual Linux Showcase and Conference*, 2000, pp. 317–327.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] J. Ou, J. Shu, Y. Lu, L. Yi, and W. Wang, "EDM: an Endurance-aware Data Migration Scheme for Load Balancing in SSD Storage Clusters," in *Proceedings of 28th IEEE International Parallel and Distributed Processing Symposium*, 2014.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] A. W. Leung, S. Pasupathy, G. R. Goodson, and E. L. Miller, "Measurement and Analysis of Large-Scale Network File System Workloads," in *USENIX Annual Technical Conference*, 2008.

[17] 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.

[18] 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.

[19] H. Kim, S. Seshadri, C. L. Dickey, and L. Chiu, "Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches," in *Proceedings of the 12th USENIX conference on File and Storage Technologies*, 2014, pp. 33–45.

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

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

[22] 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.

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

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

[25] A. Ching, A. Choudhary, W.-k. Liao, L. Ward, and N. Pundit, "Evaluating I/O Characteristics and Methods for Storing Structured Scientific Data," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, 2006.

[26] "MPI-Tile-IO Benchmark," http://www.mcs.anl.gov/research/projects/pio-benchmark/, 2014.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] D. Zhao, K. Qiao, and R. Ioan, “HyCache+: Towards Scalable High-Performance Caching Middleware for Parallel File Systems,” in *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 26-29 May 2014 2014, pp. 267–276.

[40] B. Welch and G. Noer, “Optimizing A Hybrid SSD/HDD HPC Storage System Based on File Size Distributions,” in *Proceedings of the IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2013, pp. 1–12.

[41] 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.

Summary of Changes and Responses to Reviewers' Comments

Dear Editor:

We respect and are very grateful to reviewers for their thorough and insightful comments and suggestions, which we believe have improved the quality of our paper considerably. Here in, we summarize all the changes that we have carried out in the revised manuscript. In the letter of response below, we have listed specific comments provided by the editor and reviewers and our responses.

We hope that this enhanced new version of the manuscript will meet your expectations for publication in International Journal of High Performance Computing Applications (IJHPCA).
We look forward to hearing from you!

Yours sincerely,

Dr. Shuibing He, Dr. Yan Liu, Dr. Yang Wang, and Dr. Xian-He Sun

Summary of Changes

We revised our manuscript based on reviewers' comments. We also improved this major revision to be more readable and compact within page limit. Because we fit all the content in the main file version, we don't need a supplementary file to present additional description and results. Here are some detailed changes in this major revision.

1) In "Introduction", we revised paragraph 2, we changed "proposed" to "developed". We modified the term of "terrible behavior" to make it clearer.
2) In "Introduction", we added a new paragraph (paragraph 4) to discuss the two possible types of architecture for hybrid I/O systems. The caching architecture and the one-tiered storage architecture are suitable for different system configurations and application access patterns. We pointed out they are independent research directions and their comparison is out of the scope of this paper. Therefore, we only focus on the one-tiered storage architecture in this study.
3) In "Introduction", we revised paragraph 8, the second contribution of the paper and other sentences to be more accurate and precise.
4) In "Background and Motivation", we revised section 2.3.1 "Motivation Example". We added the definitions of "*heterogeneous* requests" and "*homogeneous* requests". A heterogeneous request refers to the request that is served by both HServers and SServers; a homogeneous request refers to the one served only by the slow HServers. With these new definitions, readers would clearly understand the meaning of the terms.
5) In "Design and Implementation", we changed terms of "hybrid requests", "pure requests", "hybrid servers" and "pure servers". This will help the readers to understand the meaning of the terms. We also revised some sentences of this part to be more accurate and precise.
6) In "Design and Implementation", we changed Figure 5 and 6 to Table III and IV in Sub-section "An Analytic Data Access Cost Model". We also revised the references of Table III and IV throughout the paper.
7) In "Design and Implementation", we revised paragraph 1 of Sub-section "Implementation" to make it more precise.
8) In "Related Work", we added three references to include the most related work in Sub-section "Data Layout in SSD-based File Systems". We gave detailed discussion on these new references. Moreover, we summarized the difference between PSA with S4D-Cache in the revised manuscript.
9) Redraw all figures with consistent fonts to achieve a proper display for readers.

10) Typo and grammatical errors were corrected.

We appreciate all the detailed comments by the IJHPCA reviewers, and we provide our responses as follows.

Reviewer: 1

1. Using past performance data to train a file system how to optimize for future use can be problematic. Both how an application is used over the lifetime of an exploration and how and what data is used/generated changes. Be cautious placing a lot of trust in such a technique to guarantee much of anything.

Response: Whether or not the past performance data is effective depends on the application area. In HPC, many applications have predictive file access patterns, as we showed in the article. Therefore, exploiting historical performance data can benefits the future computation. For these applications, we use I/O profiling to optimize the system performance of their future runs. This approach is also used in many other studies [1-4].

.

2. In a least 1 place, Section III-C, the authors refer to a hybrid file server. There are no such things in this paper. They are either SSD or HDD servers, but a hybrid file system. Please correct this and other instances of that language to make sure it is clear.

Response: Thank you for your careful observation. The word "hybrid" may have different meanings in different contexts. In the case of "a hybrid parallel file system", it refers to a file system that uses HServers and SServers together to store striped data in different chunk sizes. For file requests, we categorize them in a hybrid file system into two types: *heterogeneous* and *homogeneous* in the revised version. A heterogeneous request refers to the request that is served by both HServers and SServers; a homogeneous request refers to the one served only by the slow HServers. Accordingly, "hybrid file servers" means both HServers and SServers. "Pure file servers" means HServers. We revised the definition of file requests in the last paragraph in Sub-section "Motivation Example" of Section "Background and Motivation". We changed "hybrid request" into "heterogeneous request", "pure request" into "homogeneous request", "hybrid file servers" into "HServers and SServers", and "pure servers" into "HServers" in the revised manuscript. Also, we corrected other instances of the terms throughout the paper. With these new definitions, readers would clearly understand the meaning of the terms.

3. In the same section, how can the calculation be run off-line? It seems like a dynamic reaction to current system state would be more effective. It is a reasonable first step, but not a final step.

   Response: As we pointed out in Section III.D, PSA is implemented with three phases: estimation, determination and placement. The first phase runs for the first run of the application, the second phase conducts calculation based on the trace collection. As the specified system parameters can be determined in advance, we can compute the cost model offline on any computing node of the HPC cluster. Of course, our model can by calculated dynamically if some instant values of these parameters can be measured. But by now, we are just focusing on the static case and computing the model in an offline fashion. We will investigate the on-line approach in the future work.

Reviewer: 2

1. Wouldn't the total sum of time to write the data be the same, i.e. you have x capacity at speed1 and y capacity at speed2, the total no matter how you slice it is the same. Although the response time to the applications may be fairer, rather than some see fast times some seeing slow times. I could see optimization of the overhead of the writes, by putting frequently requested data on faster media.

   Response: Yes, putting frequently requested data on faster media is a good way to improve performance, there are lots of researches on that. They distinguish the data into “hot” and “cold” data, which are respectively stored on SSDs and HDDs. However, these approaches can be effective for I/O requests only if they have strong temporal locality. Unfortunately, in HPC environments, workloads often exhibit much weaker temporal locality [5], making these strategies less attractive. Therefore, we do not consider the case where frequently accessed hot data are placed on high-performance SServers.

2. Mostly self-examining experiments, comparing this to other work would make it stronger. Or maybe compare to other authors work-see ref -- 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.

   Response: Because the fixed-size stripping method (DEF in this paper) is the default layout for existing parallel file systems, we compared PSA with it in this paper. Moreover, to the best of our knowledge, since the performance-aware data layout (PA in short) is the first effort in the data layout optimization for hybrid parallel file system, we also compared PSA with it while it was proposed by ourselves.

We would like to conduct performance evaluations with other layout policies based on the reviewer's suggestions. In terms of "S4D-Cache: Smart Selective SSD Cache for Parallel I/O Systems", it is the authors' own work, which focuses on a multi-tiered storage architecture. While the caching architecture and the one-tiered architecture are two possible organizations to utilize SSDs in a HPC system, they are independent research directions and the comparison of them is beyond the scope of this paper. Therefore, we only focus on the one-tiered storage architecture in this study. We pointed out this in the added fourth paragraph in Section "Introduction", and also summarized the difference between PSA with S4D-Cache in Sub-section "Data Layout in SSD-based File Systems" of Section "Related Work" in our revised manuscript.

3. Could this be a more general investigation with a model for storage element with arbitrary cost/speed/capacity?

   Response: Good question. Yes, in fact this could be a general data placement solution for storage element with arbitrary cost/speed/capacity. However, in this paper, our model only focuses on storage speed and storage capacity. In terms of storage cost, we only give qualitative analysis instead of quantitative evaluation. In the future work, we will fully consider the three aspects of storage devices via a more comprehensive access model to achieve a more cost-effective data layout optimization.

4. In general I would think most systems you have hybrid nodes with both SSD and HDD, should these be partitioned per server and scheduled as you suggest or is caching a better approach for these designs? Or are you just considering those systems as higher performing HDD systems?

   Response: As we pointed in response of question 2, SSD servers can be used either as a cache tier in a multi-tiered storage system or as regular storage nodes in a one-tiered storage system. Each of this architecture has its benefits and shortcomings. In fact, whether using a multi-tiered architecture or a one-tiered architecture depends on the system configurations and applications' access patterns. For example, for small non-contiguous I/O requests and high-end SSDs, the cache architecture provides higher performance; but for large I/O requests and low-performance SSDs (or high-end HDDs), the one-tiered organization is the better one.

We showed this in our recent paper [6]. However, this paper only focuses on the one-tiered storage system and its layout optimization.

5. Page 1, "terrible behaviors" is not descriptive, explain what behavior gives poor performance, assuming metadata access? "Have been proposed" GPFS and Lustre are more than proposed they are in large scale production systems.

   Response: "Terrible behaviors" means “frequent disk head movements on each server due to random accesses”, which can largely decrease I/O performance. We have revised it accordingly in the second paragraph in Section “Introduction”. We also changed "have been proposed" to “have been developed” since these file system are widely deployed in large-scale production systems.

6. Would like to see more comparisons against cache based system to show the relative performance gains that could be achieved. With a cache based system there is a loss of usable space to take into account.

   Response: As we answered in question 2 and 4, the caching architecture and the one-tiered storage architecture are suitable for different system configurations and application access patterns. Each of them has its benefits and shortcomings. We think their comparison is out of the scope of this paper. We have gave the primary general conclusions in reference [6]: for small non-contiguous I/O requests and high-end SSDs, the cache architecture provides higher performance; but for large I/O requests and low-performance SSDs, the one-tiered storage is the better one. We will investigate this in a new paper in our future work.

7. What about using SSD for metadata and HDD for data, it is an obvious split and how would that compare for performance?

   Response: This is another research direction. In fact, the idea is similar to the previous work [7], which places small files and file metadata onto SSDs, and large file extents onto HDDs. However, these schemes cannot simultaneously utilize HDDs and SSDs for large requests. We also added the following reference in the Section “Related Work” and discussed the difference between it with our work. Of course, there are different ways to use HDD and SSD in the hybrid I/O system, and what we selected is to treat them equally in logic and store the data in the same nature in this paper.

[7] B. Welch and G. Noer, "Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions," in *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, 2013, pp. 1-12.

8. Would really like to see a comparison between your system and one of the systems specifies in "related work - section B."

   Response: Thanks for your suggestions. In fact, the “PA” in the experiment part is the policy of “PADP” specified in “Related work-section B”, as we pointed out in Section “Experimental Setup” of Section “Performance Evaluation”. For other studies, we think they focus on different points, including system architecture (such as server performance, server number), target workloads (such as request size, request locality), etc. We will compare them in more detail in our future work. But, by now we only compare them verbally.

9. Nice to see the data from Argonne, but is there newer data? That is 3yrs old, and the field is growing. Could you project the data now and for the future or update with more current incite data?

   Response: Yes, the data looks like a bit old. We would like to update it, but it is not currently available to us while Argonne has the most recent data.

10. BTIO and IOR give you static IO patterns, in most production environments there is a variety of applications with various IO workloads, it seems with your proposal you would have to tag the optimal pattern with every application and only after tracing and then manage that configuration with the data. Is there a general stripe setting/ratio that would give mostly good performance for a given workload?

    Response: No, the stripe setting is application dependent. There is not a general one that would be suitable for a given workload.

11. To truly have adaptive striping you would need dynamic information from the writers, it seems you only have tracing for static patterns, how will this adapt to an application with multiple write sections with differing IO characteristics. Is there a plan to make this more dynamic? RE: figure 8.

    Response: PSA requires the prior knowledge of user’s access patterns to direct the data layout. In this paper, we only focus on the static patterns. It is a good suggestion to consider the dynamic patterns from multiple write sections, and we will investigate this in the future work.

12. If the sole focus is on installations that have HDD and are slowly adding SSD, it seems this is a more narrow work, and may have a shorter lifetime applicability? How would you rebut this perception?

    Response: This is our next step. By now, we only focus on the performance that could be improved by combining HDD and SSD.

13. Related work: related would that you should contrast:

    S4D

    Hycache+

    Xuechen Zhang; Ke Liu; Davis, K.; Song Jiang, "iBridge: Improving Unaligned Parallel File Access with Solid-State Drives," Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on , vol., no., pp.381,392, 20-24 May 2013

    doi: 10.1109/IPDPS.2013.21 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6569827&isnumber=6569783.

    Response: we compared PSA with the selected systems in detail in Sub-section "Data Layout in SSD-based File Systems" of Section "Related Work" in the revised manuscript.

14. Misc:

    52KB ->512KB page 10

    Response: We fixed these issues in the revised manuscript.

To Reviewer: 3

1. My concern with the proposed striping model, where the disk and SSDs are part of a common tier, is that the performance of the disk under a congested workload (i.e. not dedicated benchmark runs) will significantly impact the overall performance. This is why most implementations adopt a tiered model. Having both storage classes in a common tier will provide more top end performance since you are effectively aggregating the bandwidth of both storage systems, but it will likely perform poorly for real world workloads. I feel the authors should at least explain why their proposed approach is better than some of the existing tiered models.

   Response: Thanks for your comments. As we pointed out in response of question 2 to reviewer 2, SSD servers can be used either as a cache in a multi-tiered storage system or as regular storage nodes in a one-tiered storage system. Whether using caching architecture or a one-tiered architecture depends on the system configurations and applications' access patterns. For example, for small non-contiguous I/O requests and high-end SSDs, the cache architecture provides higher performance; but for large I/O requests and low-performance SSDs, the second architecture leads to better performance. We showed this in our recent paper [6]. However, this paper only focuses on the one-tired storage system. We added new explanation and pointed out this in the fourth paragraph in Section "Introduction" in the revised manuscript.

2. Another concern with the approach is the layout is pre-calculated (not dynamic). Under mixed workloads (typical of real world conditions), the system wouldn't be able to adapt to changing workloads.

   Response: Our design targets those applications with regular access patterns, instead of the mixed workloads that may exhibit unpredictable I/O behaviors.

3. All of the evaluation uses synthetic benchmarks. Please consider including a benchmark from a real-world science case that may have more mixed and complex I/O patterns.

   Response: We use standard benchmark applications to evaluate our approach**.** However, the suggestion is very good, we will include real applications as benchmark in our future work.

4. Evaluation system: While I try not to criticize evaluation systems too strongly because I understand that teams have to make use of the resources they have access to, I am concerned that the evaluation system is not representative of a typical modern day HPC system. The system has a meager amount of memory and is connected via Gigabit Ethernet. The hard drive based systems have a single hard drive. The SSDs are commodity class SSDs with (likely) very poor performance. Most HPC systems would have more modern processors, IB or 10Gb networks, and much more capable storage systems. It is difficult to extrapolate the performance of the proposed design to a real system. Perhaps the authors could obtain access to a newer system or use a cloud based solution for benchmarking.

   Response: Agree, in most cases, it is better to use the newest system as the test bed. We are also interested in evaluating the proposed layout scheme with a newer system. Unfortunately, the older system is the stable and ready system available to us. In fact, although the configuration of our test bed is not that high, our model is conceptual and generic enough to reflect any most recent HPC platform. The newest system often introduce more performance and space disparity between heterogeneous storage servers. This requires even more efforts on performance and space-aware layout solution for the best system I/O performance. Therefore, we believe our proposed data layout solution could be more valuable to direct the new I/O system design.

5. It isn't clear that the authors made any attempts to find the optimal block size for the DEFAULT runs. I am concerned that most of the benefit is coming from the algorithm tuning up the block size which may be accounting for most of the performance gain. Did the authors explore the block size for the fixed block size solution or just pick a small default size? How does the performance look if they choose a larger fixed block size?

   Response: Yes, most of the performance benefits are coming from the algorithm tuning up the block sizes. Indeed, we also have adjusted the block sizes with respect to HDD and SSD to evaluate the performance changes and selected the best one as our choice to store data. The performance is still very poor if we choose a larger fixed block size. Recent study [6] also showed similar results.

6. I also had a few minor comments:

   intro: "terrible behaviors on each file server". I'm not sure what you mean by this statement. Please re-word.

   intro: "which are cost contained and old hardware must be used efficiently". I think all systems are cost constrained. I would suggest you say that Hybrid systems are a cost effective way to deliver capacity and bandwidth which is important to nearly all classes of systems.

   page 5: What is a "pure request"?

   page 5: "since previous model" -> "since the previous model"

   page 5: "can accommodate to diverse" -> "can accommodate diverse"

   Response: "Terrible behaviors on each file server" means “frequent disk head movements on each server due to random accesses”, which can largely decrease I/O performance.” "Pure request" refers to a request that served only by HServers. We revised “pure request” as “homogeneous request” and gave its definition in the last paragraph of Sub-section “Motivation Example” in Section “Background and Motivation”. We also fixed other issues accordingly in the revised manuscript..

Reference

[1] 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, pp. 345-356.

[2] Y. Wang and D. Kaeli, "Profile-Guided I/O Partitioning," in *Proceedings of the 17th annual international conference on Supercomputing*, San Francisco, CA, USA, 2003, pp. 252-260.

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

[4] 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.

[5] D. Huang, X. Zhang, W. Shi, M. Zheng, S. Jiang, and F. Qin, "LiU:Hiding Disk Access Latency for HPC Applications with a New SSD-Enabled Data Layout," in *Proceedings of the IEEE 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems(MASCOTS'13)*, 2013.

[6] S. He, X.-H. Sun, Y. Wang, A. Kougkas, and A. Haider, "A Heterogeneity-Aware Region-Level Data Layout Scheme for Hybrid Parallel File Systems," in *Proceedings of the 44th International Conference on Parallel Processing*, 2015.

[7] B. Welch and G. Noer, "Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions," in *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, 2013, pp. 1-12.