PSA: A Performance and Space-Aware Data Layout Scheme for Hybrid Parallel File Systems

Shuibing He, Yan Liu, Xian-He Sun
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois, USA
{she11, yliu258, sun}@iit.edu

***Abstract*—The underlying storage of hybrid parallel file systems (PFS) is composed of both SSD-based file servers (SServer) and HDD-based file servers (HServer). Unlike 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 using a representative benchmark 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**

I. Introduction

Recently, many large scale applications in science and engineering have become more and more data-intensive [1], and I/O performance is regarded as a critical bottleneck of high performance computing (HPC) domain. To address this problem, parallel file systems (PFS), such as OrangeFS [2], Lustre [3], and GPFS [4], have been proposed to achieve high aggregate I/O throughput by leveraging the parallelism of multiple file servers with hard disk drives (HDD). However, fully utilizing the underlying file servers is still a challenging task.

At the same time, flash-based solid state drives (SSD) have been deployed in storage systems to further improve performance [5]. SSDs have orders of magnitude higher performance, lower power consumption, and a smaller thermal footprint over traditional HDDs [6]. While being an ideal storage media for PFSs, SSDs are not an economical option in HPC to completely replace HDDs in a large-scale deployment. Therefore, hybrid PFSs, which consist of HDD-based file servers (HServer) and SSD-based file servers (SServer), provide practical solutions for data-intensive applications [7], [8]. This type of hybrid PFS is common in large-scale I/O systems, which are cost constrained and old hardware must be used efficiently.

In parallel file systems, the stripe-based data layout scheme is commonly used to distribute data among all available file servers. This traditional scheme dispatches a large file across multiple file servers with a fixed-size stripe in a round-robin fashion. While this scheme results in uniform data placement on servers, the I/O load of each server may be imbalanced because of the non-uniform access distribution in the workloads [9]. In order to alleviate this issue, numerous different strategies have been studied on data layout optimization, such as adjusting the file stripe sizes to rearrange loads among servers [10], [11] or optimizing the file stripe distribution method according to the data access patterns [12]. 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 [6]. 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 parallel environments. 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 [13]. Existing data layout schemes focus on promoting the performance balance among servers with little attentions paid on the 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 order to address these problems, 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,

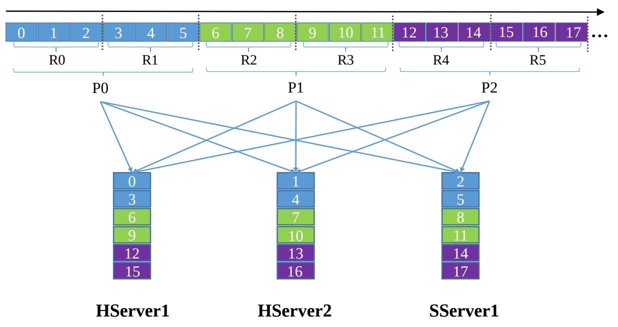


Fig. 1. Traditional data layout scheme with fixed-size stripe on file servers

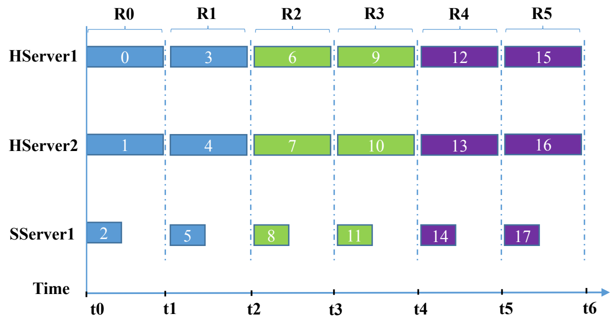


Fig. 2. Using fixed-size file stripe, SServer finishes the sub-requests faster than HServer, leading to a load imbalance.

PSA dispatches large file stripes on HServers than SServers, so that more file requests are allowed to be served by hybrid servers (comprised of HServers and SServers) rather than only HServers within a given SSD capacity. Since the hybrid servers are likely to provide better I/O performance than HServers, PSA leads to improvement in the overall performance of 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 categories of file servers, system configuration, and I/O patterns.

Specifically, we make the following contributions.

- We extend an analytical model to evaluate the I/O completion time of each request on file servers with HServers and SServers and file servers with only HServers.
- We propose a performance and space-aware algorithm based on the model to determine the appropriate file stripe sizes for HServer and SServers in hybrid PFSs.
- We implement the prototype of the PSA scheme under OrangeFS, and have conducted extensive tests to verify the benefits of the PSA scheme. Experiment results illustrate that PSA can significantly improve I/O performance.

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

II. Background and Motivation

A. Fixed-size Striping Data Layout

In order to keep pace with the processing capabilities in HPC clusters, PFSs such as OrangeFS [2] and Lustre [3] are designed to improve the performance of I/O subsystems. Files in PFSs are often organized in fixed-sized stripes, and they are dispatched onto the underlying servers in a round-robin fashion. Figure 1 illustrates the idea of the traditional data layout in PFSs. By providing even data placement in file servers and good I/O performance in many situations, this layout scheme is widely used in many PFSs. For example, in OrangeFS it is the default layout method, which is named “simple striping”.

B. Motivation Example

While traditional data layout strategies are suitable for homogeneous PFSs, they may significantly degrade the overall I/O performance of hybrid PFSs.

Figure 1 demonstrates a representative example of the fixed-size file striping data layout and a typical file access pattern in current HPC systems. 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 1. While each sub-request has the same size, their I/O completion time is significantly different due to the existing performance disparity in HServers and SServers. For example, I/O time of sub-request #2 is smaller than that of sub-request #0, as shown in Figure 2. 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 consideration when deciding the stripe size of each file server [14]. As illustrated in Figure 3, 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 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, we classify all requests in the hybrid PFS as hybrid requests and pure requests. Hybrid requests are served by both HServers and SServers, and pure requests are served only by the slow HServers. Generally, hybrid requests can lead to better I/O performance than pure requests as they are involved in more hardware resources.

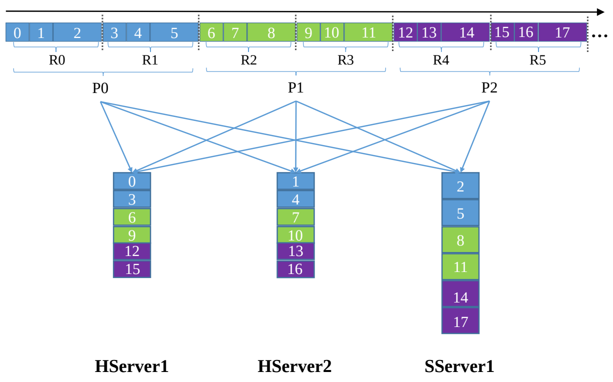


Fig. 3. Performance-aware data layout scheme. High-speed HServers are assigned with larger stripes, so that all servers finish their sub-requests almost simultaneously. However, SServers run out their limited space quickly and the remaining requests will be served only by HServers .

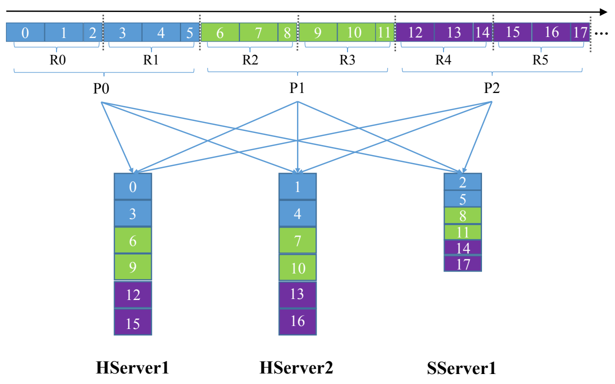


Fig. 4. Performance and space-aware data layout scheme. SServers are assigned with smaller stripes, so that there are more requests served by hybrid file servers. With a given SSD capacity, the overall I/O performance of file requests can be improved.

Therefore, if we can increase the ratio of hybrid 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 our 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.

A. *The Basic Idea of PSA*

Our 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, the basic idea of PSA is to assign HServer with larger file stripes and SServers with smaller stripes. Since the space of SServer is consumed more gradually, numerous hybrid requests are involved in the clients data accesses. As a result, we can get global optimal 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 4 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 3. Thus, there are 6 hybrid requests and 14 pure 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 hybrid requests. While the performance of hybrid file requests in PSA layout cannot be better than that of PA, PSA leads a large number of hybrid file requests. We assume the I/O time for hybrid requests under PA and PSA strategy is $2T$ and $4T$ respectively, and the I/O time for pure 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.

TABLE I
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 SSServers
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

Notice that in our strategy, it is not necessary to require all file requests to be served by hybrid file servers. We only attempt to increase the number of hybrid requests, when there is a possibility of performance optimization. In practice, determining appropriate file stripe sizes based on storage performance and space is not easy for many 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 have different performance behaviors. Second, for given file requests and SSD space, different stripe sizes lead to various proportion of hybrid requests to pure requests, which can largely impact the overall parallel file system performance.

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

Fig. 5. Cost formulas for requests on all Hservers and SServers

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

Fig. 6. Cost formulas for requests on all HServers

B. An Analytic Data Access Cost Model

To identify the optimal data layout with appropriate pair of stripe sizes for each HServer and SServer, we built an analytical cost model to evaluate the data access time in a parallel computing environment. The critical parameters are in Table I. Since SServers and HServers have distinct storage media, they have 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 various type of requests on heterogeneous file servers.

The cost is defined as the I/O completion time of each data access in hybrid PFSs, which mainly includes two parts: the network transmission time, T_{NET}, and the storage access time, T_{STOR}. 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_{STOR} consists of T_S and T_T, the former is the startup time, and the latter is the actual data read/write time on storage media.

Since hybrid requests and pure requests exist in hybrid PFSs due to the limited space of SServers, we calculate their I/O costs respectively. For hybrid file requests, we use previous cost model [14] to evaluate the data access cost. We assume the requests are fully distributed on all HServers and Servers as Figure 4, namely $m \times h + n \times s = S$, then the cost model can be calculated as in Figure 5. More details about constructing the data access cost can be found in our previous research [14].

Previous model [14] does not work for pure requests. For these requests, we assume they are distributed only on all HServers with stripe size of S/m, which leads to optimal I/O performance. The cost is defined as the formulas in Figure 6. Our model considers the space limitation of SServer and is an extension of previous model [14].

C. Optimal Stripe Size for File Servers

Based on the 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 pair of stripe sizes configuration, the loop iterates to calculate the total access cost of all file I/O requests, either with formulas in Figure 5 if the request is distributed on the hybrid file servers or formulas in Figure 6 if they are only on HServers. Finally, the pair of stripe sizes 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 $s_h \leftarrow l; s_h \le h; s_h \leftarrow s_h + step$ do
5      $s_s \leftarrow (S - m * s_h)/n$;
6      $j \leftarrow \frac{C_s}{S_s}$ /*j is the number of hybrid 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 hybrid request then
10             /* $R_i$ is distributed on $m + n$ servers */;
11             $T_i \leftarrow$ Calculate access cost of $R_i$ according to the
               formulas in Figure 5 ;
12         else
13             /* $R_i$ is distributed only on $m$ HServers*/;
14             $T_i \leftarrow$ Calculate access cost of $R_i$ according to the
               formulas in Figure 6 ;
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

Many HPC applications access their files with predictable access patterns and they often run multiple times [15]–[17]. This provides an opportunity to achieve the performance and space-aware data layout based on I/O trace analysis. We implemented the performance and space-aware data layout scheme in OrangeFS [2], which is a popular parallel file system in the HPC domain. The procedure of the PSA scheme includes the following three phases.

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 *layout 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 *data 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 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 operating system is Ubuntu 9.04, and the parallel file system is OrangeFS 2.8.6.

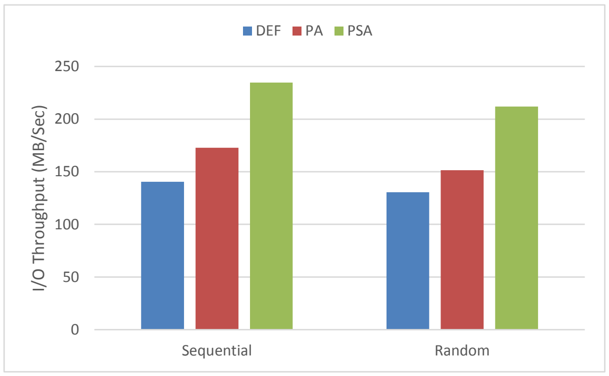
Among the available nodes, we select eight as client computing nodes, eight as HServers, and eight as SServers. By default, the hybrid OrangeFS file system is built on four HServers and four SServers. As discussed, a parallel file will be divided into two parts if SServers run out of space. The first 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 [18] to test the performance of the parallel file system. 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. 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.

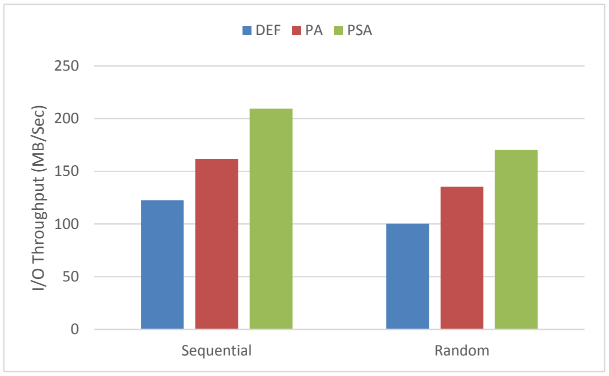
B. Experiment Results

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 effectiv.

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.

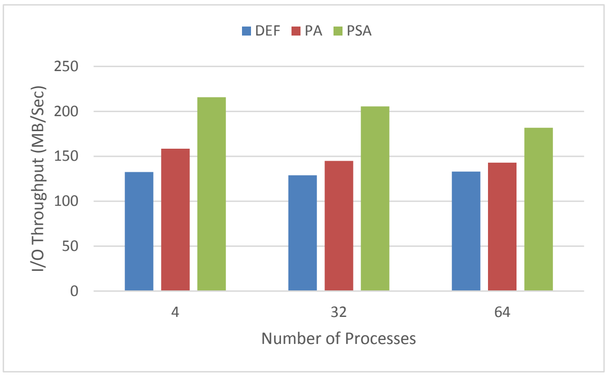


(a) Read throughput

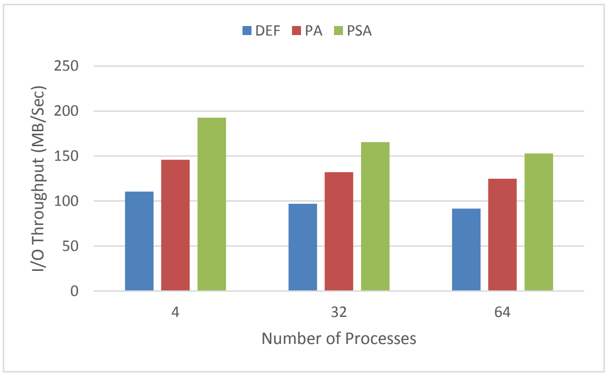


(b) Write throughput

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



(a) Read throughput

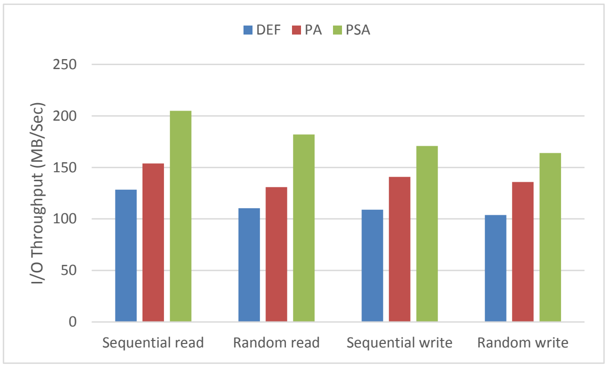


(b) Write throughput

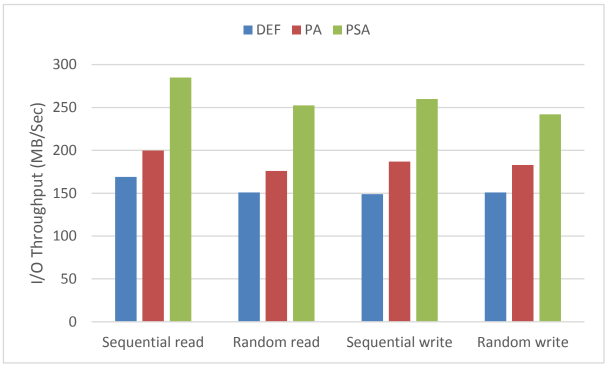
Fig. 8. Throughputs of IOR with varied number of processes

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 the hybrid file servers than the pure HServers. These results validate that PSA can choose appropriate stripe sizes for HServers and SServers when facing different request sizes.

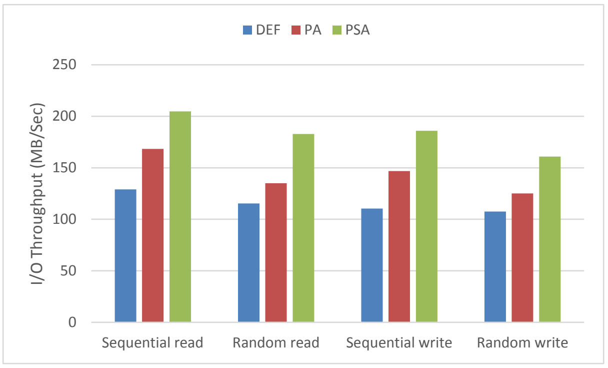


(a) Request size is 128K

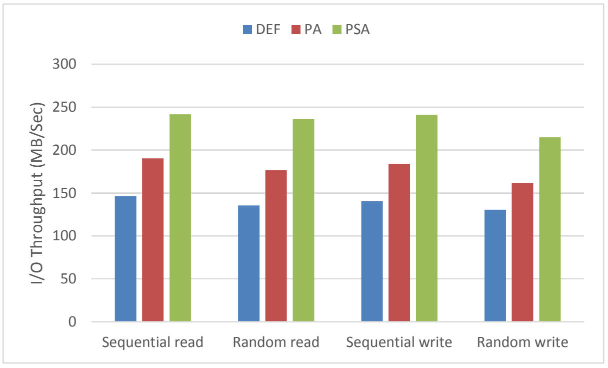


(b) Request size is 1024K

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

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 hybrid 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.

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 [12], including simple stripe, two dimensional stripe, and variable stripe. Widely adopted techniques for data partition [15], [19] and replication [12], [20], [21] 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 [10]. Server-level adaptive layout strategies adopt various stripe sizes on different file servers to improve the overall I/O performance of parallel file systems [11]. PARLO utilizes various data layout polices to accelerate scientific applications with heterogeneous access patterns at I/O middleware layer [22]. However, these efforts are suitable for heterogeneous file servers. AdaptRaid addresses the load imbalance issue in heterogeneous disk arrays [23] with adaptive number of blocks, which can not be obtained in PFSs.

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 [24] and iTransformer [25]. SSD-based hybrid storage is another popular method which utilizes the full potential of SSDs, such as I-CASH [26] and Hystor [27]. Yet, the vast majority of these techniques are done on single file servers. Our earlier work CARL [8] selects and places file regions with high access costs onto the SSD-based file servers at the I/O middleware layer, but the HDD-based and SSD-based file servers work independently. PADP [14] uses varied-size stripes to improve the performance of hybrid PFSs, but the stripe sizes are only optimized for server storage performance.

These techniques are effective in improving the performance of PFSs. 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. 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), HDD-SSD hybrid parallel file systems (PFS) have become common 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 the 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. 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 in our future work.

Acknowledgment

This work is supported in part by a research grant from Huawei Technologies Co, Ltd., the US National Science Foundation under Grant No. CNS-0751200, CCF-0937877 and CNS-1162540, and Natural Science Foundation of Hubei Province of China under Grant No. 2014CFB239.

References

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

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

[4] 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.

[5] 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.

[6] 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.

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

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

[9] 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.

[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] 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.

[12] H. Song, Y. Yin, Y. Chen, and X.-H. Sun, “A Cost-Intelligent Application-Specific Data Layout Scheme for Parallel File Systems,” in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, 2011, pp. 37–48.

[13] 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.

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

[16] 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.

[17] 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.

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

[19] 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.

[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] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.