Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
2016, Vol. 30(4) 396–410

DOI: 10.1177/1094342016631610
hpc.sagepub.com
SAGE

# Enhancing hybrid parallel file system through performance and space-aware data layout

**Shuibing He[1,2], Yan Liu[3], Yang Wang[4], Xian-He Sun[5] and Chuanhe Huang[1]**

**Abstract**

Hybrid parallel file systems (PFSs), which consist of solid-state drive servers (SServer) and hard disk drive 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 article, we propose performance and space-aware (PSA) scheme, 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 PFS in the high-performance computing 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.

**Keywords**

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 (Kandemir et al., 2008). For example, Table 1 shows the data requirements of a few representative applications at Argonne National Laboratory in 2012 (Latham et al., 2013). The generated data of these applications reach several terabytes per year. Such large data requirements are putting unprecedented pressure on computer input/output (I/O) systems to store data effectively. In the meanwhile, storage devices have a slower performance improvement than central processing units during the past three decades. While processor speeds have increased nearly by 50% each year, the access latency of a single hard disk drive (HDD) has only reduced by roughly 7% (Hennessy and Patterson, 2011). 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 (Carns et al., 2000), OrangeFS (Orange File System), Lustre (Microsystems, 2007), and GPFS (Schmuck and Haskin, 2002), 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 noncontiguous small requests, PFSs still perform poorly (He et al., 2014b) because of the low degree of server parallelism as well as the frequent disk head movements on each server due to random accesses (Song et al., 2011b). Hence, fully utilizing the underlying file servers is still a challenging task.

[1]State Key Laboratory of Software Engineering Computer School, Wuhan University, Wuhan, Hubei, China
[2]State Key Laboratory of High Performance Computing National University of Defense Technology, Changsha, Hunan, China
[3]College of Computer Science and Electronic Engineering, Hunan University, China
[4]Shenzhen Institute of Advanced Technology, Chinese Academy of Science, Shenzhen, China
[5]Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

**Corresponding author:**
Shuibing He, State Key Laboratory of Software Engineering Computer School, Wuhan University, Wuhan, Hubei, China.
Email: heshuibing@gmail.com

**Table 1.** Data requirements for select 2012 INCITE applications at Argonne Leadership Computing Facility of ANL (Latham et al., 2013).

| Project | Online (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 |

New emerging storage technologies, such as flash-based solid-state drives (SSDs), have received widespread attention in revolutionizing I/O system design (Ou et al., 2014). SSDs have orders of magnitude higher performance, lower power consumption, and a smaller thermal footprint over traditional HDDs (Chen et al., 2009). 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 (He et al., 2013, 2014a, 2014b, 2014c; Zhu et al., 2013).

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 (He et al., 2014b) or a regular storage tier (He et al., 2014c; Zhu et al., 2013) 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 article. 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 PFSs, 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 nonuniform access distribution in the workloads (Leung et al., 2008). 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 (Song et al., 2011b, 2012) or optimizing the file stripe distribution method according to the data access patterns (Song et al., 2011a). 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 (Chen et al., 2009). 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 are more expensive (Kim et al., 2014). Existing data layout schemes focus on promoting the performance balance among servers with little attentions paid on the storage space balance (He et al., 2014c). 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 article we propose performance and space-aware (PSA) scheme, a performance and space-aware data layout scheme that 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 suboptimal 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 PSA 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 article 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 related work, and Section VI concludes the article.

## 2. 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.

### 2.1. The hybrid PFS architecture

A PFS mainly includes three components: clients, servers, and metadata servers (MDSs). 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 PFS (the MDSs are not depicted), which includes two kinds of file servers: HServers and 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.

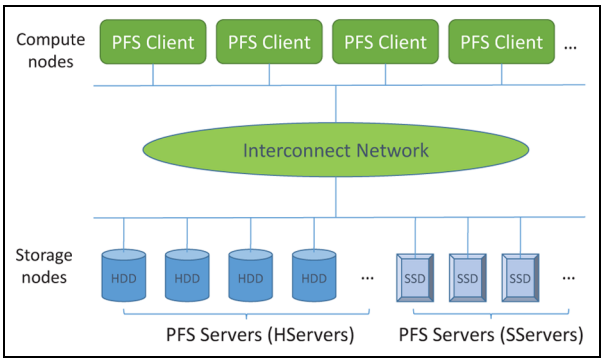


**Figure 1.** The architecture of a hybrid PFS. PFS: parallel file system.

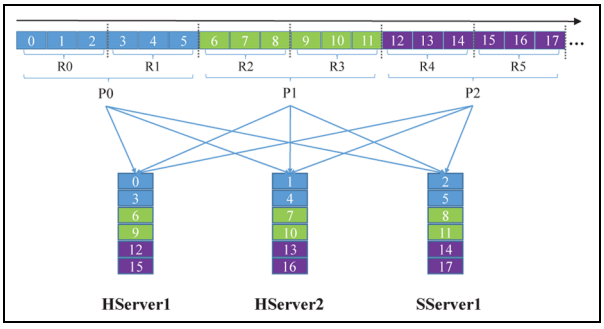


**Figure 2.** Traditional data layout scheme with fixed-size stripe on file servers.

### 2.2. Traditional fixed-size striping data layout schemes

In order to keep pace with the processing capabilities in HPC clusters, PFSs such as PVFS (Latham et al., 2013) and Lustre (Hennessy and Patterson, 2011) 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 PFSs. As this layout scheme is relatively simple and effective, it is widely used in many PFSs. For example, in OrangeFS, this scheme is the default layout method named "*simple striping*".

### 2.3. 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

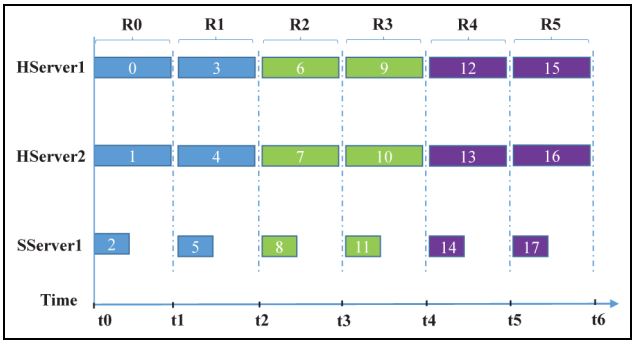


**Figure 3.** The I/O times of sub-requests on different file servers. With fixed-size file stripe, SServer will finish their sub-requests faster than HServer, leading to significant load imbalance among file servers. I/O: input/output; SServer: solid-state drive server.

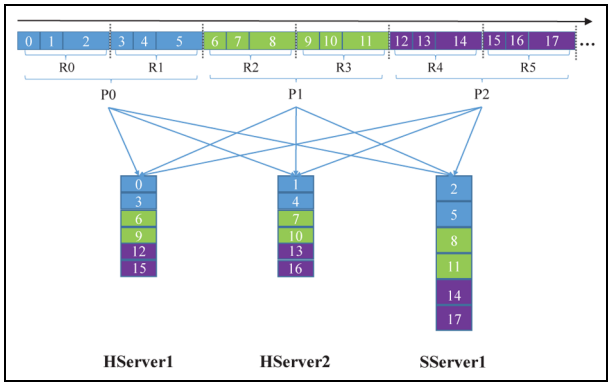


**Figure 4.** Performance-aware data layout scheme. High-speed HServers are assigned with larger stripes, so that all servers finish their sub-requests almost simultaneously. However, SServers run out their limited space quickly and the remaining requests will be served only by HServers. SServer: solid-state drive server; HServer: hard disk drive server.

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 (He et al., 2014c). 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.

## 3. 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.

### 3.1. 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 (He et al., 2014c), 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

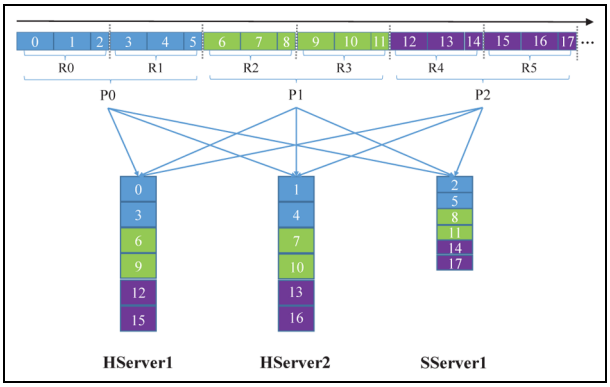


**Figure 5.** PSA data layout scheme. SServers are assigned with smaller stripes, so that there are more requests served by . With a given SSD capacity, the overall I/O performance of file requests can be improved. SServer: solid-state drive server; SSD: solid-state drive; I/O: input/output; PSA: performance and space-aware.

**Table 2.** 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 |
| $\alpha_h$ | Startup time of one I/O operation on HServer |
| $\beta_h$ | HDD transfer time per unit data |
| $\alpha_s$ | Startup time of one I/O operation on HServer |
| $\beta_s$ | SSD transfer time per unit data |

SServer: solid-state drive server; HServer: hard disk drive server; SSD: solid-state drive; I/O: input/output.

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 six sub-requests, as shown in Figure 4. Thus, there are six heterogeneous requests and 14 homogeneous requests among all requests. For the proposed PSA scheme, 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 PFS performance.

### 3.2. 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 2. Since SServers and HServers have distinct storage media, they exhibit different storage characteristics. First, $T_s$ for SServer is much smaller than that for HServer. Second, $\beta_s$ is several times smaller than $\beta_h$, 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 (He et al., 2014c) to evaluate the data

**Table 3.** 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 $T_T$ |
| $p \leqslant c(m+n)$ | $c(m+n)e$ | $\max\{cSt, pht, pst\}$ | $p * \max\{\alpha_h + h\beta_h, \alpha_s + s\beta_s\}$ |
| $p > c(m+n)$ | $pe$ | $\max\{cSt, pht, pst\}$ | $p * \max\{\alpha_h + h\beta_h, \alpha_s + s\beta_s\}$ |

SServer: solid-state drive server; HServer: hard disk drive server.

**Table 4.** 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 $T_T$ |
| $p \leqslant cm$ | $cme$ | $\max\{cSt, pSt/m\}$ | $p\alpha_h + ph\beta_h$ |
| $p > cm$ | $pe$ | $\max\{cSt, pSt/m\}$ | $p\alpha_h + ph\beta_h$ |

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 3. More details about constructing the data access cost can be found in our previous research (He et al., 2014c).

For homogeneous file requests, since the previous model (He et al., 2014c) 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 4. Hence, our model considers the space limitation of SServer and can accommodate diverse file requests.

## 3.3. 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 3.3. 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 (He et al., 2014c) 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 $\langle s_h, s_s \rangle$, the loop iterates to calculate the total access cost of all file requests, either with formulas in Table 3 if they are distributed on both HServers and SServers, or with formulas in Table 4 to calculate if they are distributed only on HServers. 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 4 KB. 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.

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.

## 3.4. 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 (Liu et al., 2014; Wang and Kaeli, 2003; Zhang and Jiang, 2010). For instance, BTIO (The NAS parallel benchmarks, 2016), an I/O kernel responsible for solving block-tridiagonal (BT) 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 PSA data layout based on I/O trace analysis.

Based on the above observation, we implemented the proposed PSA data layout scheme in OrangeFS (Orange File System) with I/O access analysis. OrangeFs is a new branch of the PVFS2 file system

**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
l ← S/(m+n);
h ← S/m;
step ← 4KB;
for s_h ← l; s_h ≤ h; s_h ← s_h + step do
    s_s ← (S − m * s_h)/n;
    j ← C_s/S_s /*j is the number of heterogeneous requests*/;
    for i ← 0; i<k; i ← i + 1 do
        Determine request type of R_i based on its offset, length, and j;
        if R_i is a heterogeneous request then
            /*R_i is distributed on m + n servers */;
            T_i ← Calculate the access cost of R_i according to the formulas in Table 3;
        else
            /*R_i is distributed only on m HServers*/;
            T_i ← Calculate the access cost of R_i according to the formulas in Table 4;
        end
        Total_cost ← Total_cost + T_i;
    end
    if Total_cost<Opt_cost then
        Opt_cost ← Total_cost;
        S_H ← s_h;
        S_S ← s_s
    end
end
```

(Carns et al., 2000), 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 $\alpha_h$, $\beta_h$, $\alpha_s$, and $\beta_s$, and the system parameters, such as $m$ and $n$ can be regarded as constants. We use one file server in the PFS 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.

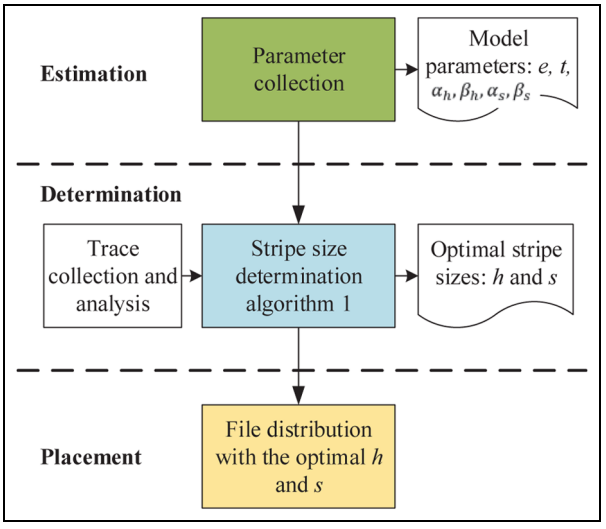


**Figure 6.** The procedure of the PSA data layout scheme. PSA: PSA: performance and space-aware.

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 corresponding file servers. For POSIX interface, we use

the "*setfattr*" command to reach the similar data layout optimization goal.

## 4. Performance evaluation

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

### 4.1. Experimental setup

We conducted the experiments on a 65-node SUN Fire Linux cluster, where each node has two AMD Opteron(tm) processors, 8 GB memory, and a 250 GB HDD. Sixteen nodes are equipped with additional OCZ-REVODRIVE 100-GB SSD. All nodes are equipped with Gigabit Ethernet interconnection. The PFS 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 (DEF) scheme and the PA scheme. In DEF, the first part of the file is placed across all servers with a fixed-size stripe of 64 KB; in PA, the stripe sizes for HServers and SServers in the first part of the file are determined by storage performance as discussed in the report by (He et al., 2014c). 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 (2016), HPIO (Ching et al., 2006), MPI-TILE-IO (2016), and BTIO (The NAS parallel benchmarks, 2016) to test the performance of the PFS. For simplicity, we will use stripe size pair $\langle h, s\rangle$ to denote that the stripe sizes on HServers and SServers are $h$ and $s$, respectively.

### 4.2. The IOR benchmark

The IOR is a PFS 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 16 GB shared file with request size of 512 KB. To simulate the situation that SServers have relatively smaller space than HServers, we limit the storage space of each SServer to 1 GB.

(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 $\langle 28\,\text{KB}, 100\,\text{KB}\rangle$,$\langle 20\,\text{KB}, 108\,\text{KB}\rangle$,$\langle 24\,KB, 104\,KB\rangle$, and $\langle 36\,\text{KB}, 92\,\text{KB}\rangle$, respectively. For PSA, the optimal stripe sizes for sequential and random read and write, are $\langle 120\,\text{KB}, 8\,\text{KB}\rangle$, $\langle 120\,\text{KB}, 8\,\text{KB}\rangle$, $\langle 116\,\text{KB}, 12\,\text{KB}\rangle>$, and $\langle 120\,\text{KB}, 8\,\text{KB}\rangle$, 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 make better trade-off between SSD's performance and space to improve the overall I/O performance. PSA's read

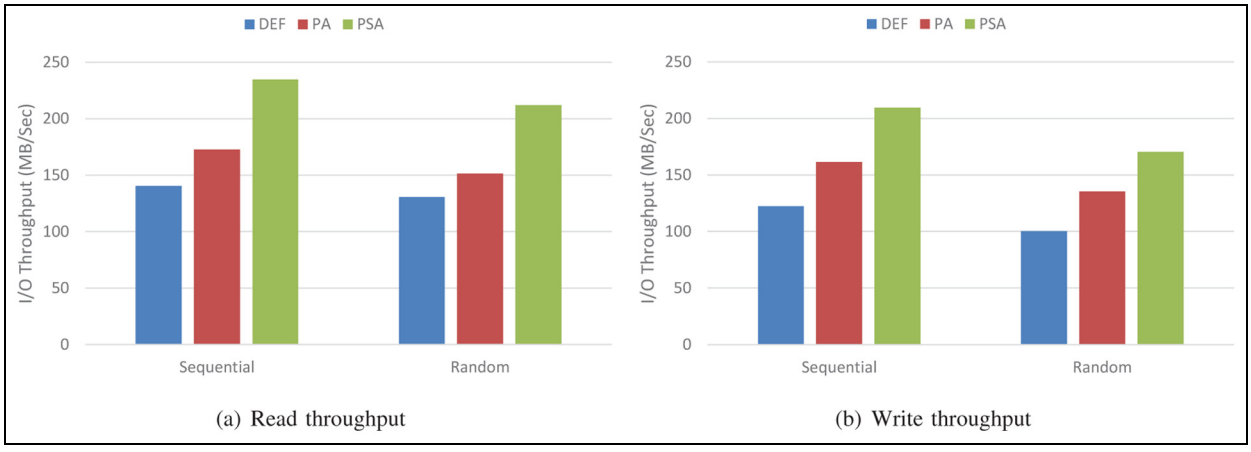


**Figure 7.** Throughputs of IOR under different layout schemes with different I/O modes. I/O: input/output.

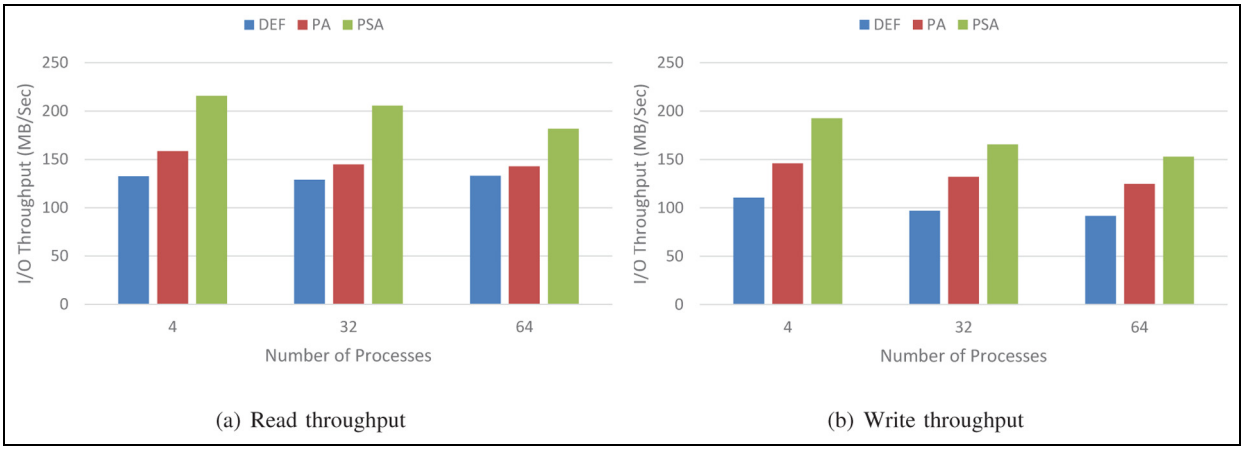


**Figure 8.** Throughputs of IOR with varied number of processes.

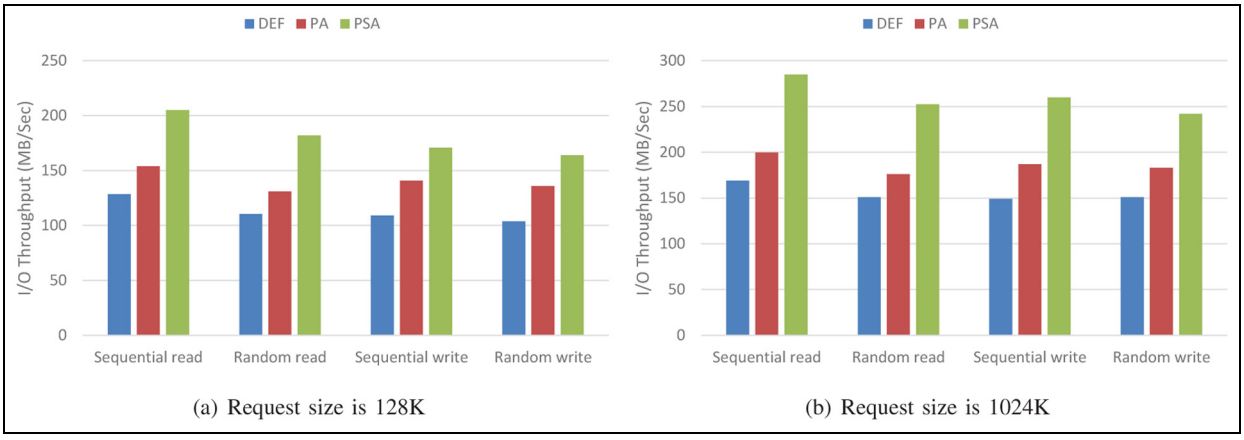


**Figure 9.** Throughputs of IOR with varied request sizes.

performance exceeds its write performance because SSDs perform 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.

(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 128 KB and 1024 KB 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

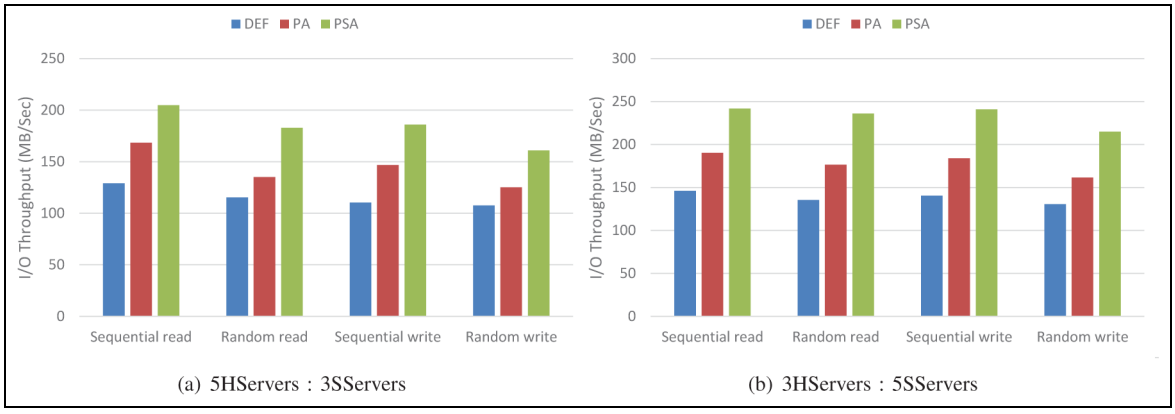


**Figure 10.** Throughputs of IOR with varied file server configurations.

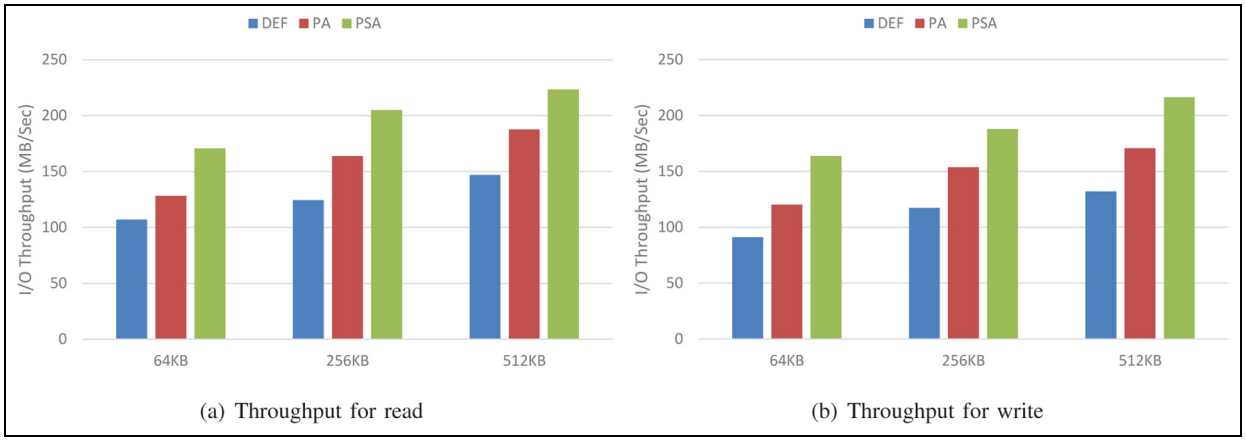


**Figure 11.** Throughputs of HPIO with varied region sizes.

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 IOR with 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. Using the optimal stripe sizes determined by the PSA layout method in this article, PSA can significantly improve the hybrid file system performance with every file server configuration.

### 4.3. The HPIO benchmark

HPIO is a program developed by Northwestern University and Sandia National Laboratories to evaluate I/O system performance (Ching et al., 2006). 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 64 KB to 512 KB. 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 64 KB, 256 KB, and 512 KB,

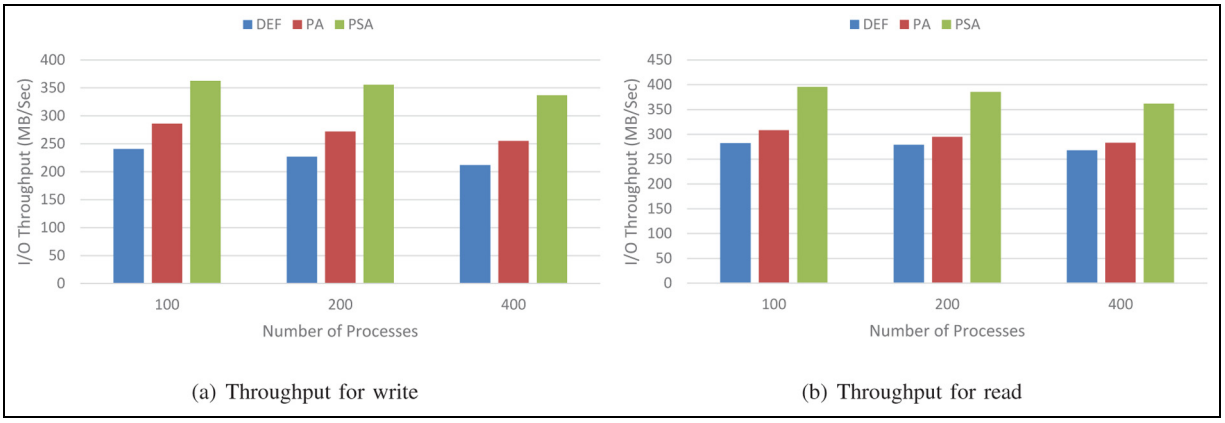


**Figure 12.** Throughputs of MPI-Tile-IO with varied number of processes.

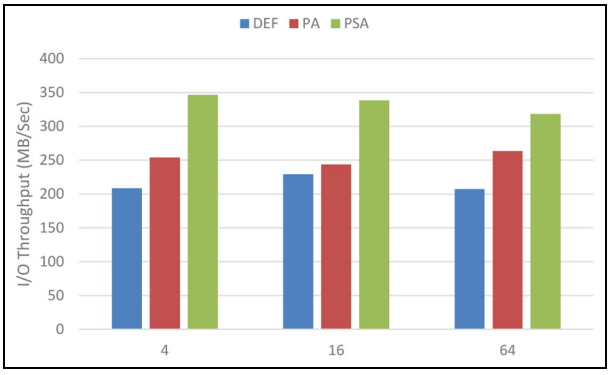


**Figure 13.** Throughput of BTIO under varied number of processes.

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.

### 4.4. The MPI-tile-IO benchmark

MPI-Tile-IO is an MPI I/O test program designed by the Parallel I/O Benchmarking Consortium (MPI-TILE-IO, 2014). 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 32 KB, 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.

### 4.5. The BTIO benchmark

In addition to the benchmarks above, we use the BTIO benchmark (The NAS parallel benchmarks, 2016) 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 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 6.64 GB 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 0.5 GB file data.

As shown in Figure 13, 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, and 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.

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 PFS provide high-performance I/O service to meet the growing data demands of many HPC applications.

## 5. Related work

### 5.1. Data layout in HDD-based file systems

Data layout optimization is an effective approach to improve the performance of file systems. PFSs generally provide several data layout strategies for different I/O workloads (Song et al., 2011a), including simple stripe, two-dimensional stripe, and variable stripe. Widely adopted techniques for data partition (Rubin et al., 2002; Wang and Kaeli, 2003) and replication (Huang et al., 2005; Jenkins et al., 2014; Song et al., 2011a) 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 nonuniform access patterns (Song et al., 2011b). Server-level adaptive layout strategies adopt various stripe sizes on different file servers to improve the overall I/O performance of PFSs (Song et al., 2012). PARLO utilizes various data layout polices to accelerate scientific applications with heterogeneous access patterns at I/O middleware layer (Gong et al., 2013). However, these efforts are suitable for heterogeneous file servers. AdaptRaid addresses the load imbalance issue in heterogeneous disk arrays (Cortes and Labarta, 2003) with adaptive number of blocks, which cannot be obtained in PFSs. Tantisiriroj et al. (2011) use HDFS-specific layout optimizations (Shvachko et al., 2010) 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.

### 5.2. Data layout in SSD-based file systems

SSDs, which exhibit noticeable performance benefits over traditional HDDs, are commonly integrated into PFSs to improve I/O performance. Currently, most SSDs are used as a cache to traditional HDDs, for example, Sievestore (Pritchett and Thottethodi, 2010), iTransformer (Zhang et al., 2012), and iBridge (Zhang et al., 2013). SSD-based hybrid storage is another popular method that utilizes the full potential of SSDs, such as I-CASH (Yang and Ren, 2011) and Hystor (Chen et al., 2011). Yet, the vast majority of these techniques are done on a single file server.

For hybrid parallel I/O systems, our earlier work CARL (He et al., 2013) selects and places file regions with high access costs onto the SSD-based file servers at the I/O middleware layer. S4D-Cache (He et al., 2014b) and SLA-Cache (He et al., 2016) use all SSD-based file servers as a cache and selectively caches performance-critical data on these high-performance servers. HyCache + (Zhao et al., 2014) uses fast storage media, such as memory or SSD to provide a scalable high-performance caching middleware to improve the I/O performance of PFSs. Welch et al. place small files and file metadata onto SSDs, and large file extents onto HDDs (Welch and Noer, 2013). 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 (He et al., 2014c) uses varied-size stripes to improve the performance of hybrid PFSs, but the stripe sizes are only optimized for server storage performance. HAS (He et al., 2015a, 2015c) 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. HARL (He et al., 2015b) applies varied stripe sizes on file servers for different regions of a file. 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.

## 6. Conclusions

With the availability of SSDs, hybrid PFSs 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 PSA data layout scheme, which distributes data across HServers and SServers 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 hybrid PFSs with two or more file server performance profiles and investigate online approaches to adapt to the changes of workloads.

## Acknowledgement

We would like to thank the anonymous reviewers for their valuable feedback and comments, which substantially improved the quality of this article.

## Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

## Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research is supported in part by the China National Basic Research Program (973 Program, No. 2015CB352400), NSFC under grant U1401258, No. 61572377, No. 61572370, and No. 61373040, the PhD Programs Foundation of Ministry of Education of China under Grant No. 20120141110073, the Natural Science Foundation of Hubei Province of China under Grant No. 2014CFB239, the Open Fund from HPCL under Grant No. 201512-02, the Open Fund from SKLSE under Grant No. 2015-A-06, and the US National Science Foundation under Grant CNS-1162540.

## References

Carns PH, Walter I, Ligon B, et al. (2000) PVFS: a parallel virtual file system for linux clusters. In: *Proceedings of the 4th Annual Linux Showcase and Conference*, Houston, TX, October 2000, pp. 317–327. Belltown Media.

Chen F, Koufaty DA and Zhang X (2009) 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*, pp. 181–192.

Chen F, Koufaty DA and Zhang X (2011) Hystor: making the best use of solid state drives in high performance storage systems. In: *Proceedings of the international conference on Supercomputing*, pp. 22–32.

Ching A, Choudhary A, Liao W-K, et al. (2006) Evaluating I/O characteristics and methods for storing structured scientific data. In: *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, Rhodes Island, April, 2006. IEEE.

Cortes T and Labarta J (2003) Taking advantage of heterogeneity in disk arrays. *Journal of Parallel and Distributed Computing* 63(4): 448–464.

Gong Z, Boyuka DA II, Zou X, et al. (2013) 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*.

He S, Liu Y and Sun X-H (2014a) PSA: a performance and space-aware data layout scheme for hybrid parallel file systems. In: *Proceedings of the Data Intensive Scalable Computing Systems Workshop*, pp. 563–576.

He S, Sun X-H and Feng B (2014b) S4D-cache: smart selective SSD cache for parallel I/O systems. In: *Proceedings of the International Conference on Distributed Computing Systems*.

He S, Sun X-H, Feng B, et al. (2013) A cost-aware region-level data placement scheme for hybrid parallel I/O systems. In: *Proceedings of the IEEE International Conference on Cluster Computing*.

He S, Sun X-H, Feng B, et al. (2014c) Performance-aware data placement in hybrid parallel file systems. In: *Proceedings of the 14th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*.

He S, Sun X-H and Haider A (2015a) 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*.

He S, Sun X-H and Wang Y (2016) Improving performance of parallel I/O systems through selective and layout-aware SSD cache. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 99: 1.

He S, Sun X-H, Wang Y, et al. (2015b) A heterogeneity-aware region-level data layout scheme for hybrid parallel file systems. In: *Proceedings of the 44th International Conference on Parallel Processing*.

He S, Wang Y and Sun X (2015c) Boosting parallel file system performance via heterogeneity-aware selective data layout. *IEEE Transactions on Parallel and Distributed Systems* 99: 1–1.

Hennessy JL and Patterson DA (2011) *Computer Architecture: A Quantitative Approach*. USA: Morgan Kaufmann, Elsevier.

Huang H, Hung W and Shin KG (2005) 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*, pp. 263–276.

Interleaved Or Random (IOR) Benchmarks (2016) Available at: http://sourceforge.net/projects/ior-sio/

Jenkins J, Zou X, Tang H, et al. (2014) RADAR: runtime asymmetric data-access driven scientific data replication. In: *Proceedings of the International Supercomputing Conference*. Springer, pp. 296–313.

Kandemir M, Son SW and Karakoy M (2008) Improving I/O performance of applications through compiler-directed code restructuring. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, 2008, pp. 159–174.

Kim H, Seshadri S, Dickey CL, et al. (2014) 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*, pp. 33–45.

Latham R, Ross R, Welch B, et al. (2013) *Parallel I/O in practice*. Technical Report. Tutorial of the International Conference for High Performance Computing, Networking, Storage and Analysis.

Leung AW, Pasupathy S, Goodson GR, et al. (2008) Measurement and analysis of large-scale network file system workloads. In: *USENIX Annual Technical Conference*.

Liu Y, Gunasekaran R, Ma X, et al. (2014) Automatic identification of application I/O signatures from noisy server-side traces. In: *Proceedings of the 12th USENIX conference on File and Storage Technologies*, pp. 213–228.

Microsystems S (2007) *Lustre File System: High-performance Storage Architecture and Scalable Cluster File System*. Technical Report Lustre File System White Paper.

MPI-Tile-IO Benchmark (2016) Available at: http://www.mcs.anl.gov/research/projects/pio-benchmark/

Orange File System (2016). Available at: http://www.orangefs.org/

Ou J, Shu J, Lu Y, et al. (2014) 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*.

Pritchett T and Thottethodi M (2010) SieveStore: a highly-selective, ensemble-level disk cache for cost-performance. In: *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pp. 163–174.

Rubin S, Bodik R and Chilimbi T (2002) An efficient profile-analysis framework for data-layout optimizations. *ACM SIGPLAN Notices* 37(1): 140–153.

Schmuck F and Haskin R (2002) GPFS: a shared-disk file system for large computing clusters. In: *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, pp. 231–244.

Shvachko K, Kuang H, Radia S, et al. (2010) The hadoop distributed file system. In: *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies*, pp. 1–10.

Song H, Jin H, He J, et al. (2012) 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*, pp. 2095–2103.

Song H, Yin Y, Chen Y, et al. (2011) A cost-intelligent application-specific data layout scheme for parallel file systems. In: *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, pp. 37–48.

Song H, Yin Y, Sun X-H, et al. (2011) 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)*, pp. 414–423.

Tantisiriroj W, Patil S, Gibson G, et al. (2011) 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)*, pp. 1–12.

The NAS parallel benchmarks (2016) Available at: www.nas.nasa.gov/publications/npb.html

Wang Y and Kaeli D (2003) Profile-guided I/O partitioning. In: *Proceedings of the 17th Annual International Conference on Supercomputing*, pp. 252–260.

Zhang X and Jiang S (2010) Interference removal: removing interference of disk access for MPI programs through data replication. In: *Proceedings of the 24th ACM International Conference on Supercomputing*, pp. 223–232.

Zhang X, Davis K and Jiang S (2012) iTransformer: using SSD to improve disk scheduling for high-performance I/O. In: *Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium*, pp. 715–726.

Zhang X, Liu K, Davis K, et al. (2013) iBridge: improving unaligned parallel file access with solid-state drives. In: *Proceedings of 27th IEEE International Parallel and Distributed Processing Symposium*.

Zhao D, Qiao K and Ioan R (2014) 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, pp. 267–276.

Zhu M, Li G, Ruan L, et al. (2013) 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*, pp. 511–517.

Yang Q and Ren J (2011) I-CASH: intelligently coupled array of SSD and HDD. In: *Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 278–289.

Welch B and Noer G (2013) 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, pp. 1–12.

## Author Biographies

**Shuibing He** received the PhD degree in computer science and technology from Huazhong University of Science and Technology, China, in 2009. He is now an assistant professor at the Computer School of Wuhan University, China. His current research areas include parallel I/O systems, file and storage systems, high-performance computing, and distributed computing.

**Yan Liu** received the PhD degree in computer science and technology from Hunan University, China, 2010. He is an assistant professor at the College of Computer Science and Electronic Engineering of Hunan University, China. His research areas include parallel and distributed file system and high-performance computing.

**Yang Wang** received the BSc degree in applied mathematics from Ocean University of China (1989) and the MSc and PhD degrees in computer science from Carleton University (2001) and University of Alberta, Canada (2008), respectively. He is currently with Shenzhen Institute of Advanced Technology, Chinese Academy of Science, as a professor. His research interests include cloud computing, big data analytics, and Java Virtual Machine on multicores.

**Xian-He Sun** received his BS in Mathematics in 1982 from Beijing Normal University, China, and completed his MS and PhD degrees in Computer Science in 1987 and 1990 from Michigan State University. He is a

distinguished professor of the Department of Computer Science, the Illinois Institute of Technology (IIT), Chicago. He is the director of the Scalable Computing Software laboratory at IIT, and is a guest faculty in the Mathematics and Computer Science Division at the Argonne National Laboratory. He is an IEEE fellow. His research interests include parallel and distributed processing, memory and I/O systems, software systems, and performance evaluation and optimization.

**Chuanhe Huang** received his PhD degree in Computer Science from Wuhan University, China, in 2002. He is now a professor at the Computer School of Wuhan University, China. His current research areas include high-performance computing, distributed computing, and computer networks.