A Holistic Heterogeneity-Aware Data Placement Scheme for Hybrid Parallel I/O Systems

Shuibing He, *Member, IEEE*, Zheng Li, Jiang Zhou, *Member, IEEE*, Yanlong Yin, Xiaohua Xu, Yong Chen, and Xian-He Sun, *Fellow, IEEE*

Abstract—We present H2DP, a holistic heterogeneity-aware data placement scheme for hybrid parallel I/O systems, which consist of HDD servers and SSD servers. Most of the existing approaches focus on server performance or application I/O pattern heterogeneity in data placement. H2DP considers three axes of heterogeneity: server performance, server space, and application I/O pattern. More specifically, H2DP determines the optimized stripe sizes on servers based on server performance, keeps only critical data on all hybrid servers and the rest data on HDD servers, and dynamically migrates data among different types of servers at run-time. This holistic heterogeneity-awareness enables H2DP to achieve high performance by alleviating server load imbalance, efficiently utilizing SSD space, and accommodating application pattern variation. We have implemented a prototype of H2DP under MPICH2 atop OrangeFS. Extensive experimental results demonstrate that H2DP significantly improve I/O system performance compared to existing data placement schemes.

Index Terms—Parallel I/O system, parallel file system, hybrid parallel file system, data placement, solid state drive

1 Introduction

During the past three decades, I/O performance has been a key bottleneck of high-performance computing (HPC) applications [1], [2], [3]. To alleviate the I/O bottleneck issue, many supercomputers deploy parallel file systems (PFSs), such as PVFS [4], OrangeFS [5], and Lustre [6] to provide I/O services. Unfortunately, although PFSs helps system performance, the various I/O access patterns and the ever-increasing data amounts of applications have driven alternative storage technologies to redesign I/O systems [7].

Flash-based solid state drives (SSDs), bring a new opportunity for I/O system evolution. Compared to hard disk drives (HDDs), SSDs have higher I/O bandwidth and lower latency [8], [9], [10]. However, SSDs often require high acquisition costs, making it impractical to completely replace HDDs in large-scale systems. Therefore, hybrid PFSs that consist of both HDD servers (HServers) and SSD servers (SServers) have attracted much attention in building cost-efficient I/O systems [11], [12], [13]. In a hybrid PFS, SServers can be used as a cache tier of HServers [7] or as a regular storage medium [11], [13], as discussed in Section 6. In this work, we focus on the latter case where HServers and SServers are combined into a single-level architecture.

PFS performance is heavily related to file data placement scheme, which determines how file data are distributed on multiple servers. For the simplicity of implementation, traditional schemes often place file data across servers with a fixed-size stripe in a round-robin fashion [4]. To maximize I/O system performance, other advanced approaches place file data considering application access characteristics [14], [15], [16]. However, most of these efforts focus on homogeneous PFSs with same HServers. When applied to hybrid PFSs, existing schemes are confronted with the following non-trivial challenges.

First, the *performance heterogeneity* between HServers and SServers may compromise I/O system performance. With the fixed-size striping (or other existing) approach, each server may be assigned with the same amount of data to process. However, as SServers have higher I/O performance than HServers, they will usually finish their I/O requests more quickly than HServers. As a result, severe load imbalance among heterogeneous servers occurs, wasting the potential of high-performance SServers. We will illustrate this in Section 2.

Second, the *space heterogeneity* between HServers and SServers can offset I/O system efficiency. Due to the high acquisition costs, SServers usually have relatively small space over HServers and thus may quickly run of their limited storage space. Consequently, some application data must be placed only on HServers. As all HServers and SServers (henceforth *hybrid servers*) working together may provide superior I/O performance than HServers alone [17],

- *S. He is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China. E-mail: heshuibing@zju.edu.cn.*
- *Z. Li is with the Computer Science Program, School of Business, Stockton University, Galloway, NJ 08205. E-mail: zheng.li@stockton.edu.*
- *J. Zhou is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100864, China. E-mail: zhoujiang@iie.ac.cn.*
- *Y. Yin is with the Intelligent Computing System Research Center, Institute of Artificial Intelligence, Zhejiang Lab, Hangzhou 311100, China. E-mail: yyin@zhejianglab.com.*
- *X. Xu is with the Department of Computer Science, Kennesaw State University, Kennesaw, GA 30144. E-mail: xxu6@kennesaw.edu.*
- *Y. Chen is with the Department of Computer Science, Texas Tech University, Lubbock, TX 79409. E-mail: yong.chen@ttu.edu.*
- *X.-H. Sun is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616. E-mail: sun@iit.edu.*

Manuscript received 13 Apr. 2019; revised 8 Oct. 2019; accepted 17 Oct. 2019. Date of publication 23 Oct. 2019; date of current version 10 Jan. 2020. (Corresponding author: Yanlong Yin.) Recommended for acceptance by W. Yu. Digital Object Identifier no. 10.1109/TPDS.2019.2948901

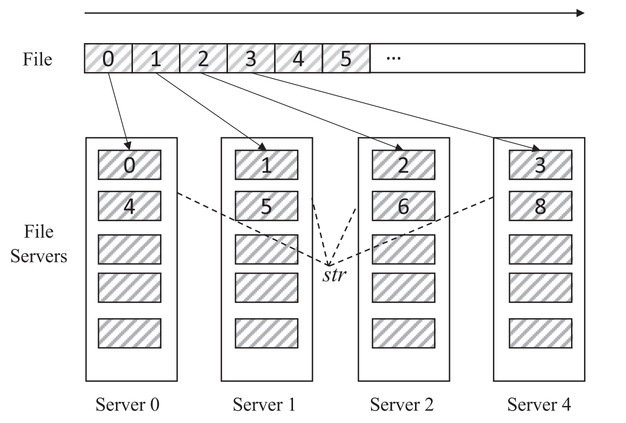

Fig. 1. Traditional fixed-size striping approach. The figure shows how a parallel file is placed on all servers with a *fixed-size* stripe (str) in a round-robin fashion.

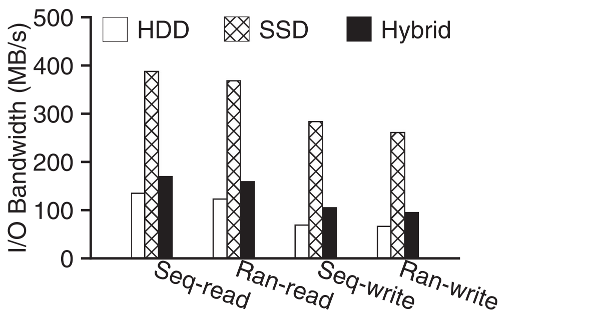

Fig. 2. IOR bandwidths with various I/O patterns. The performance is measured on an HDD, an SSD, and a Hybrid platform respectively, which denotes four HServers, four SServers, and eight hybrid servers.

the high I/O system performance may not last long once SServer space is exhausted.

Third, the *application I/O pattern heterogeneity* may lead to degraded system performance. Due to the specific interest of the application [18], [19], an application may access its data with nonuniform I/O patterns [14], [15], [16]: some data (hot data) are frequently accessed while others (cold data) are not. If the hot data are not properly placed on the high-performance servers, the overall system performance will be sub-optimal. Moreover, an application's access patterns may change at run-time [19], [20], [21]. Therefore, a *static* placement scheme will be inefficient as it is only efficient for specific access patterns but not for all of them.

Recent researches have made significant efforts to optimize file data placement for a heterogeneous PFS, such as adjusting file stripe sizes on servers [22], optimizing the number of servers (server group size) to place a file's data [13], and their combinations [17], [23], [24]. However, most of these studies focused on the single axis of heterogeneity of a hybrid FPS, and little work is devoted to exploring multiple axes of heterogeneity together, such as server performance, server space, and I/O patterns. Since each axis of heterogeneity can significantly impact the I/O system performance, existing schemes can't fully utilize the potential of the hybrid parallel I/O system.

To address above challenges, this paper proposes a *h*olistic *h*eterogeneity-aware *d*ata *p*lacement scheme (H2DP) for hybrid parallel I/O systems. The basic idea of H2DP is to place data on different servers considering three axes of heterogeneity: server performance, server space, and application I/O pattern. With this holistic heterogeneity-awareness, H2DP can achieve high system performance by alleviating load imbalance among servers, efficiently utilizing SSD space, and accommodating changing I/O patterns of applications. Specifically, we make the following contributions.

- We present a varied-size striping approach, which determines file stripe sizes on hybrid servers based on server performance, to alleviates load imbalance among servers.
- We propose a selective data placement policy, which only keeps performance-critical data on hybrid servers and the rest data on HServers, to efficiently utilize the limited SServer space.
- We devise a dynamic data migration scheme, which redistributes file data on different types of servers to accommodate changing I/O patterns.
- We implement H2DP under MPICH2 [25] atop OrangeFS. Experimental results with representative benchmarks show that H2DP can significantly improve I/O system performance compared to existing state-of-the-art data placement schemes.

The remainder of this paper is organized as follows. Section 2 presents the background and motivation. Section 3 and Section 4 illustrate the design and implementation of H2DP. Section 5 presents the performance evaluation with extensive experiments. Section 6 introduces the related work. Finally, Section 7 concludes this paper.

2 Background and Motivation

2.1 Existing Parallel File Data Placement Schemes

To meet the I/O demands of parallel applications, HPC computers rely on PFSs to store data. PFS often places a parallel file across multiple servers with a fixed-size stripe in a round-robin fashion, as shown in Fig. 1. By providing concurrent data accesses on multiple servers, this kind of methods can achieve even space utilization on servers and decent aggregated I/O bandwidth in many situations. These striping approaches are widely used in modern PFSs, such as OrangeFS, PVFS2, and Lustre. For example, they are the default placement policy called *simple striping* in OrangeFS and PVFS2. The default stripe size in OrangeFS is 64 KB, which can be configured by users via specific file system interfaces.

2.2 Heterogeneous Performance Impacts I/O Efficiency

Due to the differences of their inherit natures, HServers and SServers have heterogeneous storage performance. With this performance heterogeneity, the fixed-size striping approach may lead to sub-optimal I/O performance. To illustrate this issue, we run IOR [26] to access a 16 GB file on three platforms: four HServers (HDD), four SServers (SSD), and four HServers plus four SServers (Hybrid). The file is distributed on servers with the default data placement scheme at the stripe size of 64 KB. We limit the process number to 16, the request size to 512 KB, and the access pattern to sequential/random read/write requests.

Fig. 2 shows the I/O bandwidths of IOR under various access patterns. With the fixed-size striping strategy, the file system performance on eight hybrid servers (Hybrid) is slightly better than that on four homogeneous HServers. It is even worse than that on four homogeneous SServers. These results indicate that the hardware resources in a hybrid PFS are severely underutilized. By analyzing the I/O activities on servers, we find it is caused by the load imbalance among HServers and SServers. This result motivates us to devise varied-size striping methods based on server performance to enhance I/O system performance.

2.3 Heterogeneous Space Offsets I/O Performance

Besides performance disparity, SServers usually have a relatively smaller capacity than HServers, mainly because of their higher acquisition costs. With fixed-size striping methods, one concern is that SServers will quickly run out of their limited space. Even with the varied-size approach proposed in (Section 2.2), the situation is worse because SServers are tended to be assigned with a larger stripe size than HServers for load balance. As a result, some data of the application are placed on all hybrid servers and other data may be stored on HServers when the space of SServers is exhausted. Because hybrid servers can provide higher I/O performance than HServers due to more nodes contributing to I/O accesses [13], the system performance may degrade if the storage space of high-performance hybrid servers is not properly utilized.

Fortunately, several applications access data with non-uniform I/O patterns [14], [15], [18], which can be leveraged to optimize the data placement in a hybrid I/O system. For such applications, some data are frequently accessed while others are not. For example, in the Vector Particle-In-Cell (VPIC) simulation of magnetic re-connection phenomenon [27], applications are often more interested in the data that match a given property, such as "the energy of the particles > 1.1" [18]. Similar behaviors have been observed in adaptive mesh refinement (AMR) applications in 3D modeling [19]. *Such access skewness means that only partial data of applications are critical to the system performance.* This motivates the second design principle of H2DP: selectively placing critical data on high-performance hybrid servers while other data on HServers to efficiently utilize the limited SServer space.

2.4 Changing I/O Pattern Matters

Generally a given data placement policy may be suitable for a specific I/O access pattern but not for all of them [16]. However, applications may change their I/O access patterns at run-time [19], [20], [21]. For example, some data of the application can be frequently accessed at this time point, but they are not at other times. Therefore, a *static* data placement scheme is inefficient for applications with changing I/O patterns. An ideal scheme should adjust the data placement policy on servers over time according to the changes of I/O accesses. Although there are several previous studies can achieve this goal by creating multiple data replicas, each with a different data placement [13], [16], [18], [28], such approaches require additional storage space. In this study, we leverage *dynamic* data migration mechanism that occasionally redistributes file data on the proper types of servers to accommodate the changing I/O patterns of applications.

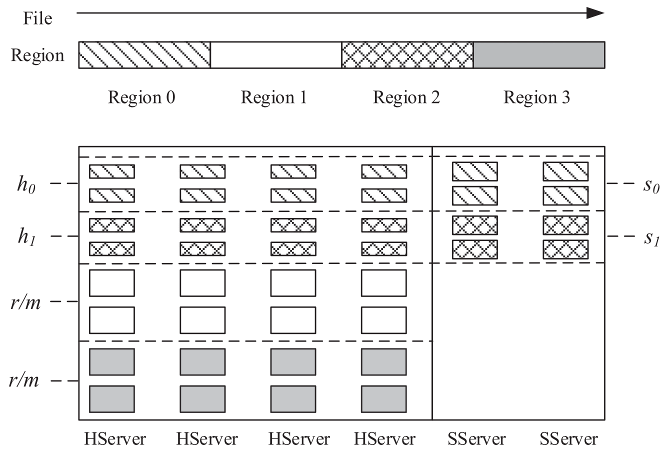

Fig. 3. H2DP data placement scheme. The figure shows how a file is placed on heterogeneous servers. A file is divided into multiple regions: critical region 0 and 2 are placed on hybrid servers and other regions are placed on HServers. The varied-size stripes on hybrid servers are determined by server performance. The data placement will also be tuned with dynamic data migration for changing I/O patterns (not shown in the figure).

3 Design of H2DP

We propose H2DP, a *h*olistic *h*eterogeneity-aware *d*ata *p*lacement scheme for hybrid PFSs. In this section, we first present the basic idea and system overview of H2DP (Section 3.1). We then elaborate the design details of its individual techniques (Section 3.2-Section 3.7) used to improve I/O system performance.

3.1 Idea of H2DP

The basic idea of H2DP is to place data on servers considering three axes of heterogeneity: server performance, server space, and application patterns. To achieve high performance, H2DP includes three critical techniques in data placement. First, to deal with the heterogeneous server performance, H2DP uses a *varied-size striping* (VSS) approach to place critical data on hybrid servers. The stripe sizes are determined based on server performance, so that all servers can complete their I/O requests almost simultaneously. Second, H2DP proposes a *selective data placement* (SDP) policy to distribute data on different types of servers. It groups file data into *critical data* and *non-critical data*, keeping only critical data on hybrid servers and the non-critical data on HServers, so that the limited SServer space can be efficiently utilized. Third, H2DP devises a *dynamic data migration* (DDM) strategy to redistribute file data on the proper types of servers over time, thus improving system performance for changing I/O patterns.

Fig. 3 illustrates the idea of the H2DP scheme. In this example, a file is divided into four fine-grained regions, each with a different data access frequency (hotness). Among all regions, region 0 and 2 have higher access hotness than others; thus they are more critical to the overall I/O system performance. Since the high-performance hybrid servers can only accommodate two regions, region 0 and 2 are placed on all hybrid servers while region 1 and 3 are on HServers. For region 0 and 2, their data are placed on HServers and SServers with the stripe pairs $< h0, s0 >$ and $< h1, s1 >$ respectively, which are determined by server

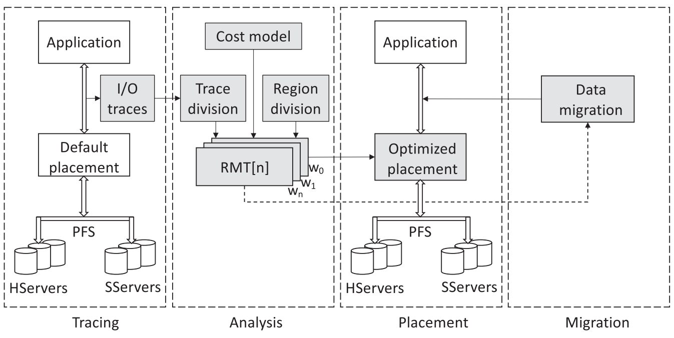

Fig. 4. The process of the H2DP scheme.

performance. Because SServers are usually faster than HServers, they are assigned with a large stripe size than HServers, i.e., $h0 < s0$ and $h1 < s1$. Furthermore, H2DP periodically tunes the region placements on servers to adapt to varying I/O workloads.

The proposed H2DP scheme requires prior knowledge of application's I/O access patterns. Fortunately, many HPC applications access their data with predictable behaviors [29], [30], [31], [32]. Taking the BTIO application [33] as an example, once its parameters, such as the size of the array, the number of the processes, etc., are given, the I/O accesses can be deduced. This is because the data access patterns of HPC applications mostly depend on their inherited numerical methods, not input data. As HPC applications often run multiple times to deal with various datasets, this feature facilitates us to enable H2DP through I/O trace analysis. Similar approaches have been widely used to optimize I/O system performance [29], [30], [31], [34].

Fig. 4 shows the procedure of the H2DP scheme, which includes four phases. In the *tracing* phase, file access statistics are collected into trace files in the MPI-IO layer during the application's first execution. In the *analysis* phase, the traces are divided into sub-traces in the order of time window (henceforth window), and the file is divided into fine-grained regions to identify critical data. Then for each window, the optimized stripe sizes for each region are determined based on the cost model and the accesses within that window. Such optimized stripe sizes are stored into a global region mapping table (*RMT*) indexed by the number of the window. In the *placing* phase, during the application's next run, each region is placed on the underlying servers with the optimized stripe sizes in *RMT[0]*. Finally, in the *migration* phase, the regions will be periodically redistributed on the proper servers with the stripe sizes in *RMT* indexed by $1, 2, \ldots,$ until the application ends. For the above four phases, the analysis phase is performed offline and the others are carried out online. In the following sections, we will elaborate the critical components of H2DP.

3.2 Trace Collection

Although several tools exist, we use IOSIG [35] to collect data access information due to its low overhead. IOSIG is designed as a pluggable library at the MPI-IO layer in the parallel I/O stack. It uses the Profiling MPI interface to record all file access activities in trace files. During execution, each process of the application linked to the trace collection library generates one trace file. Other than the linking step, there is no need for programmer intervention. Each record in the trace file includes process ID, MPI rank, file descriptor, type of operation (read/write), offset, request size, and time stamp information. In addition to the MPI-IO interface, IOSIG supports standard POSIX IO interfaces for portable deployment.

3.3 Trace and Region Division

The original traces are collected for the entire application duration. To enable timely data migration during the subsequent run of the application, H2DP first divides the original traces into sub-traces in the order of time window, then it analyzes the I/O accesses in each window, and finally determines the optimized stripe sizes on servers for all windows one by one.

A critical concern is how to determine the window size for the data migration. One can use coarse-grained intervals, such as the order of hours or days. But, as applications' I/O workloads may change most of the time [19], [36], this coarse-grained method may lead to sub-optimal performance. An alternative solution is to use short intervals, such as the order of minutes or hours, such that file data layout can quickly adapt to workload changes. Inspired by the methods adopted by previous work [20], [37], we empirically set the window size to 10 minutes, which ensures the following migration overhead does not overwhelm its performance benefit.

To enable the selective data placement on hybrid servers, SDP divides a file into fine-grained regions. One possible approach is to divide the entire file with varied-size chunks. This method relies on I/O workload analysis and is relatively complex. For simplicity and without loss of efficiency, SDP chooses the second approach: it divides a file with a fixed-size chunk/region, e.g., 64 MB or 128 MB, similar to previous work [14], [21]. The smaller the region size, the more efficient will be the data placement scheme. However, the region-level data placement incurs meta-data overhead, such as region mapping information and other statistics. This overhead is inversely proportional to the region size. In our current implementation, we empirically choose a region size of 64 MB, which brings decent system performance with a moderate overhead (Section 5.4).

3.4 Data Access Cost Model

To determine the proper stripe sizes on servers and to find the critical regions, we devise a cost model to evaluate the request time in a hybrid PFS. The related parameters are listed in Table 1. This model is inspired by the previous model in a *homogeneous* PFS [16], which has been verified in a practical I/O system. However, our model is designed for a *heterogeneous* environment, hence it considers the performance differences between HServers and SServers. First, SSDs often have a much shorter startup time and data transfer time than HDDs because they have superior performance. Second, reads and writes are treated differently on SSDs because writes usually involve slow garbage collection operations while reads do not.

The cost is defined as the overall completion time of a file request, which includes four parts: network establishing time, network transfer time, storage startup time, and storage read/write time. Since a file request may be served by

TABLE 1
Parameters in Cost Analysis Model

Symbol	Meaning
p	Number of client processes
c	Number of processes on one client node
m	Number of HServers
n	Number of SServers
h	Stripe size on HServer
s	Stripe size on SServer
r	Data size of one file request
e	Cost of single network connection establishing
t	Network transmission cost of one unit of data
α_h	Startup time on HServer
β_h	Transfer time per unit data on HServer
α_{sr}	Read startup time on SServer
β_{sr}	Read transfer time per unit data on SServer
α_{sw}	Write startup time on SServer
β_{sw}	Write transfer time per unit data on SServer

hybrid servers or homogeneous HServers due to the limited SServer space, we calculate their I/O costs respectively.

For the request served by hybrid servers, we assume it is distributed on all HServers and SServers, namely $m \times h + n \times s = r$. This assumption can make all servers to contribute the overall I/O performance. Table 2 lists the cost of a read request. The write cost will be similar except the startup and the data transfer time of SServers will change. For the request served by HServers, we assume it is distributed on all HServers with a stripe size of r/m. This assumption leads to good load balance among servers and thus brings optimized I/O performance. The corresponding cost is listed in Table 3. More details about constructing the model can be found in our previous research [22], [38].

Note that the model only considers limited parameters in characterizing I/O accesses. There are other factors that affect I/O access performance. However, the given parameters account for a large part of I/O access features. It can work as a reliable indicator to direct data placement, as shown in the evaluation section.

3.5 Region Stripe Size Decision

The proposed model shows that the cost of a file request is a function of the stripe size pair $< h, s >$ on hybrid servers. An optimized stripe size pair can efficiently reduce the overall I/O time for all requests in a time window. To obtain such optimized data placements, we propose an iterative algorithm (Algorithm 1) to determine the optimized stripe sizes on servers for all regions one by one.

For each window, the algorithm takes the request information in the region from the sub-traces as input and makes placement decisions by assuming that the region is placed on all hybrid servers. Beginning with $h = 0$, the second loop iterates h in 'step' increments while $h < \frac{r}{m+n}$. The extreme configuration we do consider is where $h = 0$, meaning that distributing data only on SServers is allowed as long as doing so benefits performance [17]. In the third loop, s starts with a size larger than h because such configuration can alleviate load imbalance among servers. For each potential stripe pair $< h, s >$, the algorithm calculates the region cost by accumulating the costs of all its requests according to the model in Table 2. Finally, the stripe pair leading to the lowest region cost is chosen. The 'step' value is 4 KB, which can be chosen by the user. Finer 'step' values result in more precise h and s, but with increased calculation overhead.

Algorithm 1. Stripe Size Determination Algorithm

```
1:  procedure StrDM $reg[k]$
2:    $step \leftarrow 4\ KB$
3:    $r \leftarrow$ request size
4:    for $i = 0; i < k; i++$ do                                  ▷ For all regions
5:      $opt\_cost \leftarrow \infty$
6:      $l_i \leftarrow$ total request number in $reg[i]$
7:      for $h \leftarrow 0; h \leq \frac{r}{m+n}; h \leftarrow h + step$ do
8:        for $s \leftarrow h + step; s \leq \frac{r}{n}; s \leftarrow s + step$ do
9:          $reg[i].cost \leftarrow 0$
10:         for $j \leftarrow 0; j < l_i; j \leftarrow j + 1$ do
11:           $T_i^j \leftarrow Cost(r, h, s)$ ▷ Call the model in Table 2
12:             $reg[i].cost \leftarrow reg[i].cost + T_i^j$
13:           end for
14:           if $reg[i].cost < opt\_cost$ then
15:             $opt\_cost \leftarrow reg[i].cost$
16:             $reg[i].strsizes \leftarrow < h, s >$
17:           end if
18:       end for
19:     end for
20:   end for
21: end procedure
```

When the application generates a large number of I/O traces, the computing overhead may be a concern. In this case, the calculation process can be easily converted into a parallel process with a simple divide-and-conquer algorithm. For the paper, we avoid to add the parallel processing part into the pseudo code to keep it simple and easy to read.

The algorithm only shows the procedure to determine the optimized stripe sizes for a given window. Such procedures are executed similarly for other windows. Once all these procedures are finished, the optimized stripe sizes for all windows are obtained and stored into the region mapping table in the order of window, which will be used to guide the following data selection and actual data placement process.

3.6 Selective Data Placement

The above algorithm determines the optimized stripe sizes for each region when they are placed on hybrid servers. However, not all regions can be placed there due to the

TABLE 2
Request Cost on Hybrid Servers

Condition	Network cost T_{NET}		Storage cost T_{STO}
	Establish T_E	Transfer T_X	Startup T_S+RD/WR T_T
$p \leq c(m+n)$	$c(m+n)e$	$\max\{crt, pst\}$	$p * \max\{\alpha_h + h\beta_h, \alpha_{sr} + s\beta_{sr}\}$
$p > c(m+n)$	pe	$\max\{crt, pst\}$	$p * \max\{\alpha_h + h\beta_h, \alpha_{sr} + s\beta_{sr}\}$

TABLE 3
Request Cost on Homogeneous HServers

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

limited space of SServers. To maximize system performance, SDP selectively places critical regions that can largely impact the I/O performance on hybrid servers and places the other regions on HServers.

To identify the critical regions, SDP calculate the performance benefit from placing the region on hybrid servers instead of HServers. In the previous subsection, we have obtained the optimized region cost when placing a region on hybrid servers. When placing the region on HServers, the stripe size is r/m and we can calculate the region cost using the model in Table 3. Let T_{H+S} and T_H denote such two region costs respectively, then the performance benefit of placing a region on hybrid servers is

$$B = T_H - T_{H+S}. \tag{1}$$

If B is larger than zero, then it is beneficial to place the region on hybrid servers. A region with frequent access requests will lead to high benefit, thus it is more critical to the overall system performance and should be placed on hybrid servers with a high priority.

To be cost-effective, SDP ranks all regions in the decent order of their performance benefits. From the region with the highest benefit, SDP begins to check whether the system has enough SServer space to accommodate each region. If yes, the region is deemed to a *critical region* and it will be placed on hybrid servers with the optimized stripe size. The free SServer space is also updated to check whether the system can hold the next region. Otherwise, the region is non-critical and it will be placed only on HServers with a fixed-stripe size of r/m. Once such space-aware decisions for all regions are made, SDP will update the optimized placement information in *RMT*.

In the actual data placement phase, SDP places all regions of a file either on hybrid servers or HServers based on the suggestion of *RMT*. As current PFSs do not support region-level data placement, we divide a logical file into multiple physical PFS files, each representing a region. For each region, we distribute the data on underlying servers with the optimized stripe sizes by specifying the file data layout via existing file system interfaces, as shown in Section 4.

3.7 Dynamic Data Migration

To enable the dynamic data migration at run-time, DDM makes data migration plans for all windows in the analysis phase. DDM will perform two types of data migrations. The first is to move cold regions from hybrid servers to HServers; these regions are no longer beneficial enough to stay there and need an "outgoing" migration. As opposed to this, the second is an "incoming" migration: it migrates regions from HServers to hybrid servers, which previously were not there but now have sufficient gains to be migrated. This ensures the high I/O performance in the future data accesses.

Algorithm 2 shows the process of constructing the plan for all windows. First, in each window it obtains the optimized data placements in the last window and the current window, respectively. Then the algorithm goes through each region handled in the last window, creating an entry in the outgoing list for each region that currently should be migrated out from hybrid servers. Similarly, the algorithm creates an entry in the incoming list for each region that should be migrated in. Accordingly, the entries in the *RMT* table are updated for the migrated regions, such that the following I/O requests can be sent to the proper locations based on the specifics of the requests.

Algorithm 2. The Migration Plan Algorithm

```
1:  procedure MigrationPlan ($w, k$)
2:    $outlist[0 : w-1] \leftarrow \emptyset$
3:    $inlist[0 : w-1] \leftarrow \emptyset$
4:    for $i = 1; i < w; i++$ do                ▷ For all windows
5:      $Reg\_last[k] \leftarrow Reg\_lookup(RMT, i-1)$
                  ▷ Obtain the data placement in the last window: i-1
6:      $Reg\_cur[k] \leftarrow Reg\_lookup(RMT, i)$
                  ▷ Obtain the data placement in the current window: i
7:      for each $reg \in Reg\_last[k]$ do ▷ Create the outgoing list
8:        $loc\_cur \leftarrow Reg\_lookup\_location(Reg\_cur, reg)$
9:        $loc\_last \leftarrow Reg\_lookup\_location(Reg\_last, reg)$
10:       if $loc\_last == "H+S"$ and $loc\_cur == "H"$ then
11:         $outlist[i] \bigcup \{reg\}$
12:       else ▷ Create the incoming list
13:         if $loc\_last == "H"$ and $loc\_cur == "H+S"$ then
14:           $inlist[i] \bigcup \{reg\}$
15:         end if
16:       end if
17:     end for
18:   end for
19: end procedure
```

During the later run of the application, DDM performs the actual migration operations at the end of each window based on the suggestion of the migration plan. For each entry in the "outgoing" list, DDM copies data from the hybrid servers to HServers. At the same time, the corresponding entries in *RMT* are updated to reflect the new region locations. For each entry in the "incoming" list, the regions are copied out based on the incoming list and *RMT* is updated accordingly. When a region is migrated to a new location, the data are placed on underlying servers with the optimized stripe sizes stored in the *RMT* table.

4 Implementation

We have implemented the H2DP scheme in the MPICH2 library on top of the OrangeFS (a successor of PVFS2) parallel file system.

4.1 Model Estimation

We use servers in the parallel file system to test the storage-related parameters, such as α and β for both HServers and SServers. Note that these parameters can vary for different I/O patterns. In addition, we use multiple pairs of nodes (client nodes and file servers) to estimate network parameters, such as e and t. We repeat the tests at different request sizes with thousands of times (the number is configurable), and use the average of multiple iterations for each parameter in the model.

4.2 Additional Data Structure

Existing PFSs do not support region-level data placement. To enable H2DP, we divide a logical file into multiple physical PFS files, each representing a file region. We place some regions on hybrid servers and others on HServers based on the placement benefit analysis. We use *RMT* to record region mapping and stripe size information. We leverage *Berkeley DB* [39] to implement this structure. Each record in *Berkeley DB* is a key-value pair. The key is the RegionID, which is encoded with application name, number of processes, original file name, and region sequence. The value is the target region file name and the optimized stripe size pair.

4.3 Optimized Data Placement

Currently OrangeFS can be accessed by two interfaces: the direct *PVFS2* interface and the *POSIX* interface. For compatibility, H2DP supports both of them with the optimized stripe sizes. With the PVFS2 interface, we utilize the "pvfs2-xattr" command to set the data distribution of directories where the application files are located. In addition, when a new file is created, we use the "pvfs2-touch" command with the "-l" option to specify the order of servers, so that the file stripe size h and s can be configured for the corresponding HServers and SServers accordingly. With the POSIX interface, we achieve the same goal by setting the optimized stripe sizes of a file via the "setfattr" command.

4.4 I/O Redirection in MPI-IO

As H2DP divides a logical file into fine-grained regions, the original I/O requests will be redirected to the underlying physical files, each representing a file region. To find the proper locations of file requests, we modify the MPI library so that *RMT* is loaded with `MPI_Init()` and unloaded with `MPI_Finalize()`. We also utilize an in-memory list to maintain the frequently accessed entries in *RMT* to speed up the lookup operations. Moreover, we revise `MPI_File_read/write()` (and other variants of read/write functions), so that I/O requests can be forwarded to the proper regions with the optimized placements based on the specifics of the requests.

4.5 Data Migration

Data migration operations may affect the activities of the normal I/Os. To alleviate this issue, we create a lightweight daemon program running in the background. It monitors a queue that contains all the regions that need to be migrated based on the migration plan. During the application's later run, when the last request finishes at the end of each window, the program will read regions from the original files and write them into the targeted files with the optimized data placements. To avoid serious interference with the foreground I/O activities of the application, the migration is performed asynchronously.

5 Performance Evaluation

5.1 Experimental Setup

We conducted the experiments on a 49-node SUN Fire Linux cluster, where there are 48 computing nodes and one head node. The head node is responsible for system management. Each computing node has two AMD Opteron(tm) processors, 8 GB memory, and a 250 GB HDD. Eight nodes are equipped with additional 100 GB SSD. All nodes are equipped with Gigabit Ethernet interconnection. The operating system is Ubuntu 14.04, the parallel file system is OrangeFS 2.9.6, and the MPI library is MPICH2-1.4.1 [25]. By default, we use 32 nodes as computing nodes running I/O clients, 12 nodes as HServers, four nodes as SServers. To mimic the scenarios where SServers run out of their space, we assume that SServers can accommodate 1/6 data of the tested files at most. Before running each test, we clean the operating system cache to ensure all data are read from storage devices. We also periodically flush the dirty data from the memory buffer to the storage devices, so that the write bandwidth is correctly measured. For each test, we measure the I/O bandwidth in the subsequent run of the application. We repeat each test five times and report the average bandwidth as the result. The I/O patterns of all five runs are the same. For each run, the execution time equals the total amount of data divided by the I/O bandwidth.

We compare H2DP with three other schemes: *DEF*, *PADP* [22], and *PSA* [40]. DEF is the default approach that simply places file data on servers with a fixed-sized stripe of 64 KB. PADP only considers performance heterogeneity: it uses varied-size striping to distribute data on hybrid servers, but the placement on hybrid servers are randomly chosen. PSA considers both performance and space heterogeneity: it assigns a smaller stripe size to SServers and a large size to HServers, so that the load imbalance can be alleviated and more requests can be placed on hybrid servers. However, PSA ignores application heterogeneity: it randomly chooses the data placed on hybrid servers and keeps a static data layout even for changing I/O patterns. Moreover, the stripe size decision policy may compromise the performance of requests on hybrid servers. As opposed to the three counterparts, H2DP is a holistic scheme which fully considers performance, space, and application heterogeneity. A We use two popular micro-benchmarks (IOR and HPIO) and one macro-benchmark(BTIO) to evaluate the I/O system performance.

- *IOR* is a parallel file system benchmark developed by Lawrence Livermore National Laboratory [26]. It provides three APIs: MPI-IO, POSIX, and HDF5. In the experiments, we only use the MPI-IO interface.
- *HPIO* is a popular I/O testing program developed by Northwestern University and Sandia National Laboratories [41]. This benchmark can generate various data access patterns by changing three parameters: region count, region spacing, and region size.

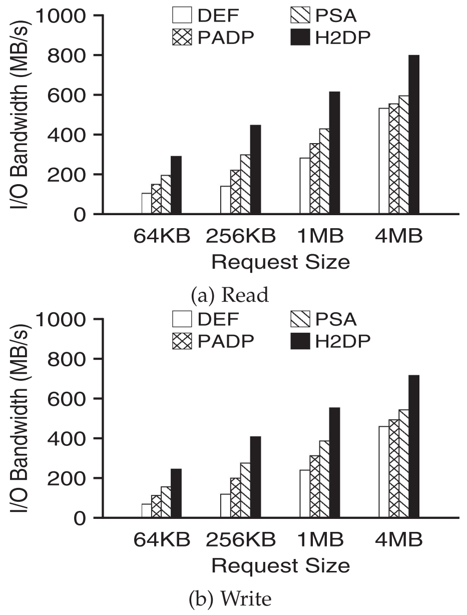

Fig. 5. IOR bandwidths with various request sizes.

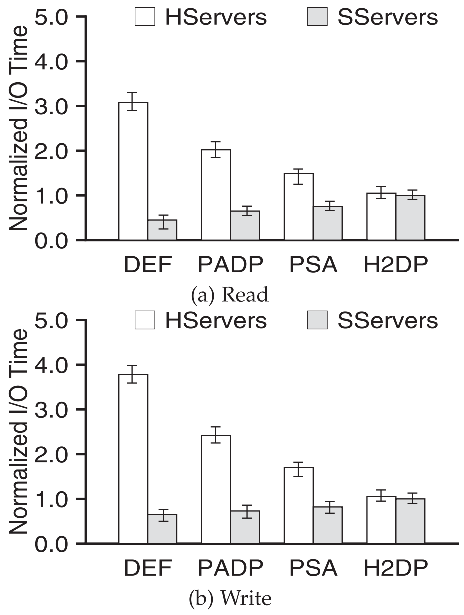

Fig. 6. I/O times on different servers.

- *BTIO* is from the NAS Parallel Benchmark (NPB3.3.1) suite [33]. It is a typical scientific application with interleaved computation and I/O phases. It uses a Block-Tridiagonal (BT) partitioning pattern to solve the three-dimensional compressible Navier-Stokes equations.

The two micro-benchmarks can generate various I/O access patterns. The macro-benchmark represents the data access behavior of an application. We use their combination to test I/O system performance. This evaluation mechanism is also widely used in many other studies.

5.2 Major Results

5.2.1 The IOR Benchmark

The original IOR can generate uniformly random workloads. To generate non-uniform I/O access patterns, we modified IOR to generate I/O requests following Zipf random distribution ($\alpha = 0.8$ where α is the Zipfian parameter). IOR issues I/O requests to various parts of a file with different frequencies. Moreover, the access frequencies change at run-time. Unless otherwise specified, IOR runs with 32 processes, each with individual I/Os to access a 128 GB shared file, and the request size is 512 KB.

Various Request Sizes. Fig. 5 plots the I/O bandwidths of IOR with different request sizes of 64 KB, 256 KB, 1 MB, and 4 MB. As shown in the figure, H2DP achieves the best performance among the four schemes. Compared to DEF, PADP, and PSA, the read improvement is up to 219, 103, and 49 percent, respectively, and the write improvement is up to 253, 115, and 56 percent. As expected, DEF has the worst performance because it ignores the heterogeneity of server performance, server space, and I/O access pattern. Accordingly, its fixed stripe size for all servers will lead to severe load imbalance and its random data placement on hybrid servers will lead to low utilization of the scarce SServer space. In contrast, the other three schemes can alleviate these issues by leveraging the varied-size striping and the selective data placement.

To give a detailed explanation for the performance behaviors of various schemes, Fig. 6 plots the I/O time of all servers during the IOR execution with the request size of 256 KB. For each scheme, the figure shows two bars grouped together. In each group, the left bar shows all 12 HServers' average I/O time, the right one shows all four SServers' average I/O time. Each bar has an interval on top to show the range of all I/O times represented in that bar. Also, each group's two bars are normalized to the average I/O time of all SServers with the H2DP scheme. As shown in Fig. 6, the I/O workloads on HServers and SServers are severely skewed under the DEF scheme. In contrast, PADP, PSA, and H2DP achieve more even load distribution among heterogeneous servers by assigning the strip sizes to different types of servers based on server performance. We also note that the I/O times of the SServers in PADP and PSA are lower than the one in H2DP. This is because in H2DP more I/O requests are served by the hybrid servers due to its selective data placement.

Another observation from Fig. 5 is that PSA is faster than PADP. This is because PSA takes space heterogeneity into account while PADP does not. However, PSA is still worse than H2DP due to following three reasons. First, H2DP only keeps frequently accessed data on hybrid servers while PSA randomly chooses the data on them, thus H2DP can increase the requests served by high-performance hybrid servers than PSA. Second, reverse to PSA, H2DP assigns large stripes to SServers and small stripes to HServers, which can increase the performance of requests on hybrid servers. Third, PSA is a static approach which can't tune its data placement for changing I/O patterns while H2DP can do this through dynamic data migration.

Various Numbers of Processes. Fig. 7 shows the IOR bandwidths with various number of processes. The request size is set as 512 KB. Similar to the previous tests, H2DP achieves the best I/O performance among the four schemes. Compared to DEF, H2DP obtains up to 188 percent performance improvement for reads and up to 204 percent improvement

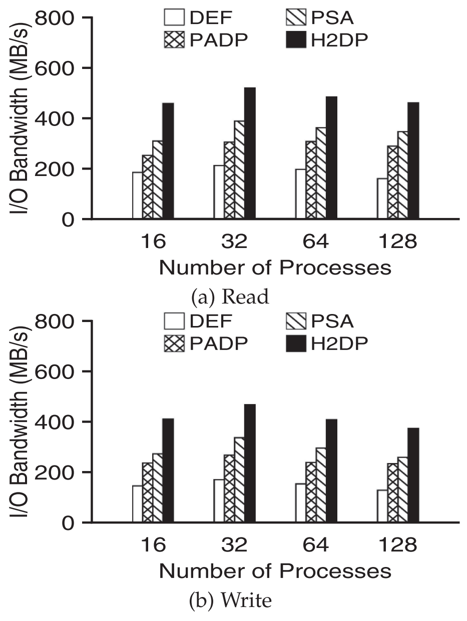

Fig. 7. I/O bandwidths with various numbers of processes.

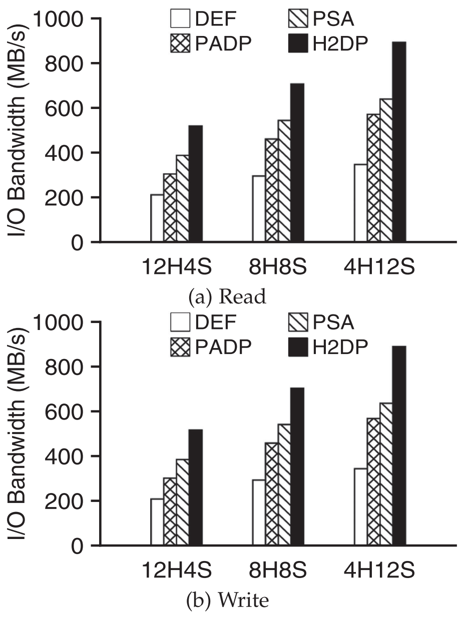

Fig. 8. IOR bandwidths with various server ratios.

for writes. In contrast to PADP, the read performance is increased by up to 82 percent and the write performance is increased by up to 75 percent. When compared to PSA, H2DP obtains up to 49 and 51 percent improvements for reads and writes, respectively. These improvements again show the necessity of the heterogeneity-awareness in terms of server performance, server space, and application patterns in the data placement of a hybrid parallel I/O system. We also note that with the increasing of process number, all the schemes lead to degraded system performance because of the enhanced I/O interference on servers. However, the gap of H2DP is less pronounced than other schemes because it can more efficiently utilize the performance and space features of SServers. This result shows that H2DP has good scalability in terms of various numbers of processes.

Various Server Ratios. Fig. 8 shows the IOR bandwidths with various server ratios. The label "x H y S" means the I/O system consists of x HServers and y SServers. As can be seen from the results, H2DP can improve I/O throughputs for both read and write requests. With the ratio of 12H4S, H2DP outperforms DEF, PADP, and PSA by 143, 71, and 34 percent for reads, and 173, 74, and 38 percent for writes. When the ratio is 4H12S, the improvements become up to 197,62, and 40 percent, respectively. This result indicates that H2DP has performance advantages over existing placement schemes. Moreover, we find that the system performance increases with the increase of the number of SServers. This is because more SServers contribute to the overall system performance and H2DP can make full utilization of them.

5.2.2 The HPIO Benchmark

In our experiments, we set the number of process to 16, the region count to 40960, the region spacing to 0, and vary the region size from 64 KB to 512 KB. The file system is built on 12 HServers and four SServers. We run the benchmark to measure the throughput of the storage system for relatively uniform and contiguous data accesses.

As shown in Fig. 9, H2DP has superior I/O performance over existing schemes. It outperforms DEF by 192, 136, and 111 percent for reads at the region size of 64 KB, 256 KB, and 1 MB, respectively. For writes, the improvements are 166, 156, and 113 percent. In contrast to PADP, H2DP obtains read improvements of 91, 72, and 62 percent, respectively, and write improvements of 71, 70, and 67 percent. Compared to PSA, H2DP can increase the I/O bandwidth by up to 41, 21, and 17 percent at different region sizes. These results mean that H2DP is effective for the HPIO benchmark. However, such performance improvements over PSA in HPIO are not as significant as those in IOR because the workload exhibits weaker locality than IOR. Similar to the previous tests, H2DP provides higher I/O performance for large region sizes (request sizes) because large requests benefit more for both HServers and SServers.

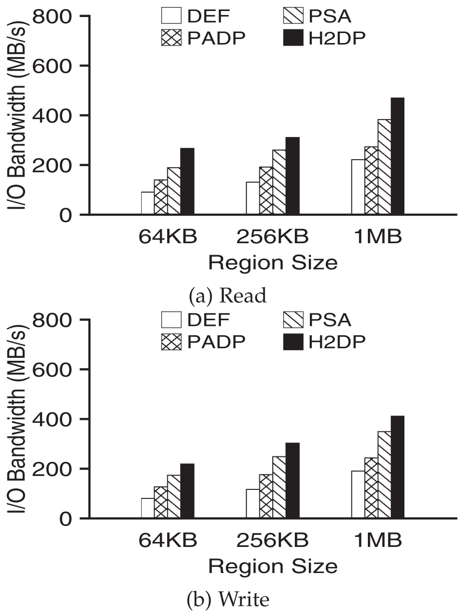

Fig. 9. HPIO bandwidths with various region sizes.

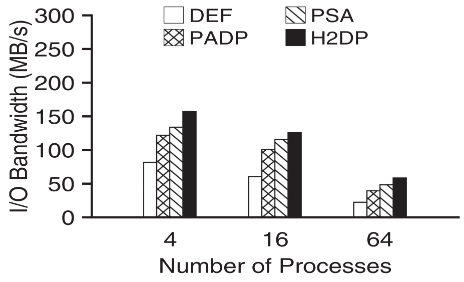

Fig. 10. BTIO bandwidths with various process numbers.

5.2.3 The BTIO Benchmark

BTIO partitions a three-dimensional array across a square number of processes, each process processing multiple Cartesian subsets. All processes write their data into a shared file and then read back into their memory. We consider the Class D and simple subtype BTIO workload in the experiments. In other words, BTIO writes and reads a total size of 135.8 GB data with non-collective I/O operations. We use 4, 16, and 64 processes as BTIO requires a square number of processes. We run this benchmark to measure the system performance with relatively uniform and non-contiguous I/O requests. The output file is stored in an OrangeFS file striped across 12 HServers and four SServers.

Fig. 10 plots the results of BTIO. As shown in the figure, H2DP achieves better throughput than DEF, PADP, and PSA. In contrast to DEF, H2DP obtains the improvements by 137, 105, and 92 percent for 4, 16, and 64 processes, respectively. Compared to PADP, the improvements are 48, 28, and 25 percent, respectively. In comparison with PSA, H2DP gets the improvement by up to 19 percent. Similar to the HPIO benchmark, the performance improvements obtained by H2DP are not as significant as those of IOR. This is because the workload of BTIO exhibits weaker skewness than IOR so that the selective and dynamic data placement brings less performance benefits. We also note that the performance of all schemes decreases as the number of processes increases. This is due to two reasons. First, the I/O contention in each file server becomes more serious because each file server needs to concurrently serves more processes. Second, the I/O requests size decreases, which leads to lower I/O efficiency on disks.

5.3 Effectiveness of Dynamic Data Migration

The dynamic data migration can reduce the number of requests on HServers by moving the frequently accessed (hot) regions to hybrid servers, therefore improving the overall system performance. We study the impact of the dynamic data migration on the effectiveness of H2DP in this section. We run IOR with the same experimental setup as the "various request sizes" test in Section 5.2.1. IOR issues random I/O requests following Zipf random distribution, the request size is varied from 64 KB, 256 KB, 1 MB, to 4 MB. The number of processes is fixed to 32. The file system is across 12 HServers and four SServers.

Fig. 11 shows the IOR bandwidths without and with the DDM module. In the figure, the "W/DDM" approach (H2DP) means the initial placement+SDP+VSS+DDM and the "W/O" approach=Initial placement+SDP+VSS. We can

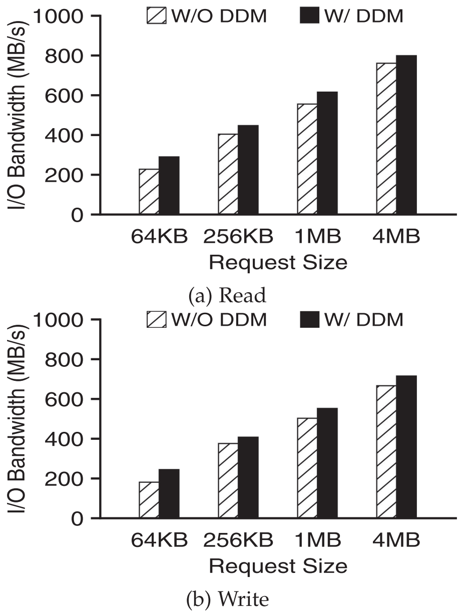

Fig. 11. IOR bandwidths without and with dynamic data migration with various request sizes.

see that DDM can improve system performance for both reads and writes at various request sizes. When the request size is smaller than 1MB, DDM can offer significant performance improvements. For instance, the read improvement reaches 26 percent at the size of 64 KB. In this case, DDM can migrate 17 percent of the total data and increase the number of requests to hybrid servers by 32 percent. However, when the request size is increased to 4 MB, the improvement is 5 percent for reads. The enhancement reduces because SSDs exhibit much better performance over HDDs for small requests, thus migrating critical data to hybrid servers offers less pronounced performance benefits for large requests.

When DDM is enabled, it uses data access features to determine the hot critical regions and then replaces the non-critical regions on hybrid servers with the critical ones. The data migration operation does incur performance overhead. However, by empirically setting the proper migration period and using the asynchronous migration policy, such overhead is well amortized by the benefits of reducing I/O operations to the slow HServers. As a result, the DDM scheme can effectively improve the I/O system performance.

5.4 Overhead Analysis

H2DP incurs some overheads on system resource utilization. We discuss them as follows.

Meta-Data Space Overhead. In H2DP, the region mapping table is maintained to track the region location and region stripe configuration information on underlying servers. These two tables bring additional space overheads. In our implementation, we use a fixed region size of 64 MB, hence for a 100 GB file, there are up to 1600 regions. We use 128 bytes for each entry in *RMT*, hence the total size of *RMT* will be 0.4 MB, which is less than 0.001 percent and negligible in terms of the original file space.

Performance Overhead. H2DP incurs performance overhead due to additional activities, such as I/O access collection, determine the optimized stripe sizes, and constructing the migration plan.

We use IOSIG [31] to collect the traces during the first execution of the application. It runs online. Previous work has shown that the run-time overhead of IOSIG is very low (below 6 percent) [35], which also applies to this study.

For the stripe size determination, the calculation of Algorithm 1 only involves simple arithmetic operations. Its time complexity is $O(N)$, $N = \sum_{i=0}^{k-1} l_i$, where N is the number of I/O requests, k is the number of regions, and l_i is the number of requests in region i. For a relatively small number of I/O requests, the time and space overheads to obtain the optimized stripe sizes are not a concern. When there are a large number of I/O traces to handle, namely N is very large, we can easily convert Algorithm 1 into a parallel process to reduce the calculation overhead.

Constructing a migration plan also incurs additional overhead. For Algorithm 2, the time complexity is $O(wk)$, where w and k are the number of time windows and the number of regions. These two parameters are usually limited for a given application. Furthermore, the migration plan is constructed once and offline. Thus, the overhead is acceptable and it does not affect the execution of the application.

6 Related Work

Data Placement in Homogeneous File Systems. Existing parallel file systems provide several data placement policies [16], such as simple striping, two-dimensional striping, to distribute data on servers for relatively regular data access patterns. For non-uniform I/O requests, researchers have proposed file stripe resizing approaches [14], [15]. Facing more complex access patterns, there are other data placement techniques, such as data partition [34], data replication [28], [31], data reorganization [42] to optimize file system performance. All these studies are designed for homogeneous I/O systems where same HDD servers (HServers) are deployed; thus server heterogeneity is not a concern. Our work differs from them in that we focus on hybrid I/O systems where both HServers and SServers are deployed.

Data Placement in Heterogeneous File Systems. Owing to the outstanding performance, SSD servers (SServers) have been integrated into parallel file systems. One approach is to use SServers as a cache of HServers [7], [43]. This approach has an assumption that the aggregated I/O bandwidth of SServers is far higher than that of HServers. However, this is not always true for all access patterns and system configurations. For example, when the request size is very large and the system has a much large number of HServers than SServers. In this case, an alternative approach is to use SServers as a storage layer, such as CARL [12], [37], PADP [22], HAS [13], [23], and HARL [17], [44]. All these schemes mostly focus on performance heterogeneity among servers, but they ignore the space and application heterogeneity in the data placement.

A highly related work is PSA [38], [40], which resizes file stripes considering both performance and space features of heterogeneous file servers. PSA assigns SServers with a small stripe size and HServers with a large size. This policy may compromise the request performance on hybrid servers. As opposed to this, H2DP assigns SServers with a large stripe size and HServers with a small size to overcome this issue. Furthermore, PSA does not consider application pattern characteristics: it randomly places critical data on hybrid servers and applies a static data placement policy even for changing I/O patterns. In contrast, this study comprehensively considers performance, space, and application heterogeneity in the data placement of hybrid PFSs.

Data Placement in SSD-Based Single-Node System. SSDs are also widely used in a single-node or local storage system. Some studies use an SSD as a cache of a traditional HDD [1], [45], [46], [47]. Other approaches integrate an SSD and an HDD into a unified block device [9], [48], [49], [50]. Most of them place the critical data on an SSD *in a single node* with a fixed-size block. Different from the prior efforts, H2DP focuses on *a parallel I/O system* with multiple heterogeneous devices, namely HDDs and SSDs. At the same time, the critical data can be simultaneously placed on all devices rather than SSDs alone, based on the specifics of data access patterns and system configurations. Therefore, the potential of all devices can be utilized. However, determining what data should be placed on heterogeneous devices and how to place them are challenging. This study addresses these issues.

7 Conclusion

Hybrid parallel file systems have attracted much attention in building large-scale I/O systems due to the high cost-effectiveness. In this study, we propose a holistic heterogeneity-aware data placement scheme, which distributes data across HDD and SSD servers based on server performance, server space, and application patterns. We have implemented H2DP within MPICH2 on top of OrangeFS. Experimental results show that H2DP is effective to improve the hybrid I/O system performance compared to existing data placement approaches. In the future, we plan to evaluate H2DP in a large-scale HPC cluster that is not currently available to us. We also intend to extend H2DP in practical environments.

Acknowledgments

This work was supported in part by the National Science Foundation of China No. 61572377, the Natural Science Foundation of Hubei Province of China No. 2017CFC889, the Fundamental Research Funds for the Central Universities No. 2018QNA5015, and the Zhejiang Lab Research Project No. 2019KC0AC01.

References

[1] Y. Kim, S. Atchley, G. R. Valle, and G. M. Shipman, "LADS: Optimizing data transfers using layout-aware data scheduling," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 67–80.
[2] R. Latham, R. Ross, B. Welch, and K. Antypas, "Parallel I/O in practice," in *Proc. Tut. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2015.
[3] P. Carns et al., "Understanding and improving computational science storage access through continuous characterization," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol.*, May 23–27, 2011, pp. 1–14.
[4] P. H. Carns, I. Walter, B. Ligon, R. B. Ross, and R. Thakur, "PVFS: A parallel virtual file system for Linux clusters," in *Proc. 4th Annu. Linux Showcase Conf.*, 2000, pp. 317–327.

[5] "Orange file system," 2017. [Online]. Available: http://www.orangefs.org/
[6] P. Schwan, "Lustre: Building a file system for 1000-Node Clusters," in *Proc. 2003 Linux Symp.*, 2007, pp. 380–386.
[7] S. He, X.-H. Sun, and B. Feng, "S4D-Cache: Smart selective SSD cache for parallel I/O systems," in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2014, pp. 514–523.
[8] K. Zhou, S. Hu, P. Huang, and Y. Zhao, "LX-SSD: Enhancing the lifespan of NAND flash-based memory via recycling invalid pages," in *Proc. 33rd Int. Conf. Massive Storage Syst. Technol.*, 2017.
[9] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the best use of solid state drives in high performance storage systems," in *Proc. Int. Conf. Supercomputing*, 2011, pp. 22–32.
[10] A. Caulfield, L. Grupp, and S. Swanson, "Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications," in *Proc. 14th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2009, pp. 217–228.
[11] M. Zhu, G. Li, L. Ruan, K. Xie, and L. Xiao, "HySF: A striped file assignment strategy for parallel file system with hybrid storage," in *Proc. IEEE Int. Conf. Embedded Ubiquitous Comput.*, 2013, pp. 511–517.
[12] 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 *Proc. IEEE Int. Conf. Cluster Comput.*, 2013, pp. 1–8.
[13] S. He, X.-H. Sun, and A. Haider, "HAS: Heterogeneity-aware selective data layout scheme for parallel file systems on hybrid servers," in *Proc. 29th IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 613–622.
[14] 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 *Proc. 11th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2011, pp. 414–423.
[15] H. Song, H. Jin, J. He, X.-H. Sun, and R. Thakur, "A server-level adaptive data layout strategy for parallel file systems," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp. Workshops PhD Forum*, 2012, pp. 2095–2103.
[16] H. Song, Y. Yin, Y. Chen, and X.-H. Sun, "A cost-intelligent application-specific data layout scheme for parallel file systems," in *Proc. 20th Int. Symp. High Perform. Distrib. Comput.*, 2011, pp. 37–48.
[17] S. He, X.-H. Sun, Y. Wang, A. Kougkas, and A. Haider, "A heterogeneity-aware region-level data layout scheme for hybrid parallel file systems," in *Proc. 44th Int. Conf. Parallel Process.*, 2015, pp. 340–349.
[18] H. Tang et al., "Usage pattern-driven dynamic data layout reorganization," in *Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 16–19, 2016, pp. 356–365.
[19] B. Nguyen, H. Tan, and X. Zhang, "Large-scale adaptive mesh simulations through non-volatile byte-addressable memory," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2017, pp. 1–12.
[20] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami, "Cost effective storage using extent based dynamic tiering," in *Proc. 9th Conf. File Storage Technol.*, 2011, pp. 273–286.
[21] 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 *Proc. 12th USENIX Conf. File Storage Technol.*, 2014, pp. 33–45.
[22] S. He, X.-H. Sun, B. Feng, and F. Kun, "Performance-aware data placement in hybrid parallel file systems," in *Proc. 14th Int. Conf. Algorithms Architectures Parallel Process.*, 2014, pp. 563–576.
[23] S. He, Y. Wang, and X.-H. Sun, "Boosting parallel file system performance via heterogeneity-aware selective data layout," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2492–2505, Sep. 2016.
[24] S. He, X.-H. Sun, Y. Wang, and C. Xu, "A migratory heterogeneity-aware data layout scheme for parallel file systems," in *Proc. 32nd IEEE Int. Parallel Distrib. Process. Symp.*, 2018, pp. 1133–1142.
[25] A. N. Lab, "MPICH2 : A high performance and widely portable implementation of MPI," 2018. [Online]. Available: http://www.mcs.anl.gov/research/project-detail.php?id=2
[26] "Interleaved or random (IOR) benchmarks," 2017. [Online]. Available: http://sourceforge.net/projects/ior-sio/
[27] S. Byna et al., "Parallel I/O, analysis, and visualization of a trillion particle simulation," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2012, pp. 1–12.
[28] J. Jenkins, X. Zou, H. Tang, D. Kimpe, R. Ross, and N. F. Samatova, "RADAR: Runtime asymmetric data-access driven scientific data replication," in *Proc. Int. Supercomputing Conf.*, 2014, pp. 296–313.
[29] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp, "Parallel I/O prefetching using MPI file caching and I/O signatures," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2008, pp. 1–12.
[30] X. Zhang and S. Jiang, "InterferenceRemoval: Removing interference of disk access for MPI programs through data replication," in *Proc. 24th ACM Int. Conf. Supercomputing*, 2010, pp. 223–232.
[31] Y. Yin, J. Li, J. He, X.-H. Sun, and R. Thakur, "Pattern-direct and layout-aware replication scheme for parallel I/O systems," in *Proc. 27th IEEE Int. Parallel Distrib. Process. Symp.*, 2013, pp. 345–356.
[32] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai, "Automatic identification of application I/O signatures from noisy server-side traces," in *Proc. 12th USENIX Conf. File Storage Technol.*, 2014, pp. 213–228.
[33] "The NAS parallel benchmarks," 2017. [Online]. Available: www.nas.nasa.gov/publications/npb.html
[34] Y. Wang and D. Kaeli, "Profile-guided I/O partitioning," in *Proc. 17th Annu. Int. Conf. Supercomputing*, 2003, pp. 252–260.
[35] Y. Yin, S. Byna, H. Song, X.-H. Sun, and R. Thakur, "Boosting application-specific parallel I/O optimization using IOSIG," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2012, pp. 196–203.
[36] A. W. Leung, S. Pasupathy, G. R. Goodson, and E. L. Miller, "Measurement and analysis of large-scale network file system workloads," in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 213–226.
[37] S. He, Y. Wang, Z. Li, X.-H. Sun, and C. Xu, "Cost-aware region-level data placement in multi-tiered parallel I/O systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 1853–1865, Jul. 2017.
[38] S. He, Y. Liu, Y. Wang, X.-H. Sun, and C. Huang, "Enhancing hybrid parallel file system through performance and space-aware data layout," *Int. J. High Perform. Comput. Appl.*, vol. 30, no. 4, pp. 396–410, 2016.
[39] M. A. Olson, K. Bostic, and M. I. Seltzer, "Berkeley DB," in *Proc. USENIX Annu. Tech. Conf.*, 1999, pp. 183–191.
[40] S. He, Y. Liu, and X.-H. Sun, "A performance and space-aware data layout scheme for hybrid parallel file systems," in *Proc. Data Intensive Scalable Comput. Syst. Workshop*, 2014, pp. 41–48.
[41] A. Ching, A. Choudhary, W.-k. Liao, L. Ward, and N. Pundit, "Evaluating I/O characteristics and methods for storing structured scientific data," in *Proc. 20th Int. Parallel Distrib. Process. Symp.*, 2006.
[42] W. Tantisiriroj, S. Patil, G. Gibson, S. Seung Woo, S. J. Lang, and R. B. Ross, "On the duality of data-intensive file system design: Reconciling HDFS and PVFS," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 1–12.
[43] S. He, X.-H. Sun, and Y. Wang, "Improving performance of parallel I/O systems through selective and layout-aware SSD cache," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2940–2952, Oct. 2016.
[44] S. He, Y. Wang, X.-H. Sun, and C. Xu, "HARL: Optimizing parallel file systems with heterogeneity-aware region-level data layout," *IEEE Trans. Comput.*, vol. 66. no. 6. pp. 1048–1060, Jun. 2016.
[45] M. Srinivasan and P. Saab, "Flashcache: A general purpose write-back block cache for Linux," 2013. [Online]. Available: https://github.com/facebook/flashcache
[46] A.-I. A. Wang, P. Reiher, G. J. Popek, and G. H. Kuenning, "Conquest: Better performance through a disk/persistent-RAM hybrid file system," in *Proc. USENIX Annu. Tech. Conf.*, 2002, pp. 15–28.
[47] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2010, pp. 1–11.
[48] H. Payer, M. Sanvido, Z. Bandic, and C. Kirsch, "Combo drive: Optimizing cost and performance in a heterogeneous storage device," in *Proc. 1st Workshop Integrating Solid-State Memory Storage Hierarchy*, 2009, vol. 1, pp. 1–8.
[49] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "HybridStore: A cost-dfficient, high-performance storage system combining SSDs and HDDs," in *Proc. IEEE 19th Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst.*, 2011, pp. 227–236.
[50] Q. Yang and J. Ren, "I-CASH: Intelligently coupled array of SSD and HDD," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Architecture*, 2011, pp. 278–289.

Shuibing He received the PhD degree in computer science and technology from the Huazhong University of Science and Technology, in 2009. He is now a ZJU100 young professor with the College of Computer Science and Technology, Zhejiang University. His research interests include parallel I/O systems, file and storage systems, high-performance and distributed computing. He is a member of the IEEE and ACM.

Zheng Li received the PhD degree in computer science from the Illinois Institute of Technology. He is currently an assistant professor with the Computer Science Program, School of Business, Stockton University. His research interests include distributed computing, real-time computing, many-core computing, and reconfigurable computing.

Jiang Zhou received the PhD degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2014. He is currently an assistant researcher with the Institute of Information Engineering, Chinese Academy of Sciences. His research interests include distributed file systems, big data, high-performance and cloud computing. He is a member of the IEEE.

Yanlong Yin received the BE degree in computer science and MS degree in computer architecture from the Huazhong University of Science and Technology, Wuhan, China in 2006 and 2008, respectively, and the PhD degree in computer science from the Illinois Institute of Technology, in 2014. He is currently a research scientist with Zhejiang Lab. His research interests include parallel computing systems, parallel I/O systems, and parallel file systems.

Xiaohua Xu received the PhD degree in computer science from the Illinois Institute of Technology, Chicago, in 2012. He is an assistant professor with the Department of Computer Science with Kennesaw State University, Georgia. His teaching and research interests include network security, machine learning, wireless networking, and cloud computing.

Yong Chen is an associate professor and director of Data-Intensive Scalable Computing Laboratory, Computer Science Department, Texas Tech University. He is also the site director of the NSF Cloud and Autonomic Computing Center, Texas Tech University. His research interests include data-intensive computing, parallel and distributed computing, high-performance computing, and cloud computing.

Xian-He Sun received the BS degree in mathematics from Beijing Normal University, China, in 1982 and the MS and PhD degrees in computer science from Michigan State University, in 1987 and 1990, respectively. He is a distinguished professor of the Department of Computer Science, Illinois Institute of Technology (IIT), Chicago. His research interests include parallel and distributed processing, memory, and I/O systems. He is a fellow of the IEEE.