Model-driven Data Layout Selection for Improving Read Performance

Jialin Liu
Department of Computer Science
Texas Tech University
Lubbock, Texas, USA
Email: jaln.liu@ttu.edu

Surendra Byna, Bin Dong, Kesheng Wu
Computational Research Division
Lawrence Berkeley Laboratory
Berkeley, California, USA
Email: {SByna, DBin, KWu}@lbl.gov

Yong Chen
Department of Computer Science
Texas Tech University
Lubbock, Texas, USA
Email: yong.chen@ttu.edu

***Abstract*—Performance of reading scientific data from a parallel file system depends on the organization of data on physical storage devices. Data is often immutable after producers of data, such as large-scale simulations, experiments, and observations, write the data to the parallel file system. As a result, read performance of data analysis tasks is often slow when the read pattern does not conform with the original organization of the data. For example, reading small non-contiguous chunks of data from a large array is many times slower than reading the same size of contiguous chunks of data. Towards improving the data read performance during analysis phase, we are developing the Scientific Data Services (SDS) framework for automatically reorganizing previously written data to conform with the known read patterns. In this paper, we introduce a model-driven strategy for selecting the data layouts that benefit the performance of different read patterns. We have developed a parallel I/O model based on the striping parameters on Lustre file system and the block-level striping on RAID-based disks within an Object Storage Target (OST) of Lustre. We have applied the model to reorganize large 3D array datasets on a Cray XE6 platform and achieved 9X to 128X improvement in accessing the reorganized data compared to reading the data in its original layout.**

***Keywords*-Scientific Data Management, Scientific Data Services (SDS), I/O Performance Model, Big Data, high performance computing**

I. Introduction

Discoveries in all fields of science depend on the collection and analysis of large quantities of data [24]. For example, the recent discovery in confirming the existence of the Higgs boson that led to winning the Nobel prize in physics involved analyzing petabytes of high-energy collision data generated by the Large Hadron Collider [18]. Similarly, applications across various science domains, including plasma physics [6], astrophysics [11], astronomy [12, 14], and climate [1], produce or are expected to produce petabytes of data each hour or each day. Rapid analysis of these enormous datasets is vital to enable fast scientific discoveries.

Scientific data analysis tasks often use large cluster computers, where the tasks read the data to be analyzed from highly concurrent parallel file systems. However, in many analyses, the time to read data from the storage is significantly longer than the time to write. When data is written, data producers such as simulations organize data on the storage system in a layout[1] that gives the best performance to write. Reading that data in a different pattern is typically slow. For example, a plasma physics application writing particle data organizes the data in large contiguous chunks on all available storage targets. An analysis task studying highly energetic particles needs to read and scan the whole data even when only a small fraction of the data is needed for analysis. Similarly, accessing a small sub-plane or sub-cube of a large 3-D dataset results in numerous non-contiguous I/O requests causing poor read performance. The read performance must be improved to speed up the scientific data analysis process.

An effective strategy to accelerate data read speeds is reorganizing previously written datasets according to specific read patterns and redirecting data read calls transparently to the reorganized data. Our recent effort in reorganizing original data and accessing it achieves a performance improvement of greater than 50X compared to accessing the original data [9]. Many research efforts [2, 17, 20, 23, 25] also explored optimizing data organization by deriving an optimal file layout strategy for a certain access pattern.

While it is proven that reorganizing data is beneficial, there are three main steps towards automating the process of data reorganization. They are: identifying an organization that improves the performance of read patterns, reorganizing the data automatically, and redirecting the I/O read calls to the reorganized data. Read patterns represent the logical view of data in terms of arrays and indices. The physical view of data varies based on the parallel file system and the storage hardware. In the case of Lustre file system, data is stored on Object Storage Targets (OSTs) and the data can be written to or read from multiple OSTs simultaneously. If the data accesses are distributed uniformly on the OSTs, parallelism may be beneficial. Accessing one or a small fraction of the available OSTs wastes concurrency and may result in poor performance. Finding a balance between parallelism, locality, and the number of requests holds key to better performance. Performing reorganization of data

[1]We use $layout$ and $organization$ interchangeably to refer to the way data is distributed on a parallel file system.

automatically requires a technique to find the computation and memory resources to read data from the original dataset, perform any transformation such as data transposition or sorting, and write the data to a new layout. Redirecting I/O calls based on the read patterns needs analyzing the read calls and transparently reading the reorganized data when they can provide better performance.

Towards solving the above steps of automating reorganization, we are developing the Scientific Data Services (SDS) framework [9, 10] that performs reorganization of data and redirection of I/O read calls to the reorganized data transparently. This paper introduces a strategy for identifying an optimal layout for known read patterns that appear in analysis tasks. We have developed a parallel I/O performance model based on the OSTs of the Lustre file system and the disk blocks of a RAID storage within an OST. We have also developed a prediction mechanism based on an empirical analysis of measured I/O times in reading data from the datasets organized in different layouts. Using these measured time and read pattern information as a training set, we predict read performance for a new read request with various data layouts and select a layout that gives the best performance. We have applied the data layout selection strategy in the SDS framework for organizing a 3D array dataset and evaluated performance benefits in reading data from the reorganized dataset.

- We identify and show that maintaining multiple layouts plays an important role in scientific discovery and can have significant performance benefits.
- We design a new parallel I/O performance model to select optimal data organizations for different I/O request patterns. We design a new disk-level I/O representation that reduces the complex parameter space of the hierarchical parallel I/O stacks into minimum key factors.
- We apply the automatic layout selection strategy in SDS and show it is easy to implement.

The rest of this paper is organized as follows. Section II discusses related work and compares them with our proposed approach. Section III introduces the SDS framework and describes a scientific dataset as a use case. Section IV introduces parallel I/O subsystem and our I/O formalization. Section V presents the design of our parallel I/O performance model. We present our system setup for performance evaluation and experimental results in Section VI. We conclude the discussion with a brief discussion of future work in Section VII.

II. Related Work

Several research efforts studied organization of data on parallel file systems and prediction of I/O performance. In this section we briefly discuss existing research.

A. Data Organization

In high performance parallel systems, various projects explored improving performance by organizing data efficiently. For example, the CHARISMA [19] project characterized and explored the importance of access patterns and data layouts of different user cases. In current scientific data optimization, applying space-filling curves in placing data improves the locality. In [15], the authors mapped multi-dimensional data to the space-filling curves and indexed the data using a B+ tree to facilitate the query. The Elastic Data Organization (EDO) addresses the issue of accessing slow dimensions of datasets via providing various data organization schemes [25]. A Pattern-direct Layout-aware (PDLA) runtime replication was proposed in [28]. The PLDA system detects the read pattern of an application, and selects one of three layouts in PVFS, i.e., 1D, 2D-V and 2D-H, to replicate the accessed data. 'Smart-IO' [26] applies an optimized chunking model, hierarchical spatial aggregation, and space filling curve reordering to speedup data analytics. These efforts demonstrate that data reorganization is beneficial. But to the best of our knowledge, the existing works try to search the best layout for specific access pattern, although the I/O requests in the same pattern may desire different layouts and one application may have multiple access patterns. The complexity of parallel system and the multiple I/O stacks make it impossible to provide a single optimal layout for all I/O requests, even for one access pattern. We identify the importance of maintaining multiple optimal layouts and develop a model to automatically select one.

B. I/O Performance Model

Several efforts studied predicting the I/O performance of parallel applications as I/O is a major bottleneck in high performance computing (HPC) applications. There has been a lot of related work starting from the single disk era to today's parallel file systems. For example, in [21], the authors demonstrate a procedure for selecting IOR benchmark parameters to match the I/O patterns of an application and to predict the I/O performance of the application. A queuing network model was proposed in [22] to predict the performance of I/O as a function of I/O hardware configuration (e.g., 16 RAID-3 disk arrays). In [23], a cost model was developed to predict the I/O cost under different layouts (1DV, 1DH and 2D) on PVFS. In [29], the author designs an auto tuning I/O framework based on the queuing model on Cray XT5 system. To predict the I/O performance, researchers usually represent the I/O workload as vectors [27, 30], and apply different models to fit the data. The representation of workloads, the regression tools, and the training I/O traces are three important factors in model quality[27]. Our work differs from the previous modeling efforts in workload representation. By looking into the parallel file storage hierarchy, we logically map the I/O from the HDF5 layer down to the disk level. There

are several efforts focusing on I/O at the disk level, e.g., DiskSim [4]. In contrast, our disk-level I/O is a top-down logical view by computing and mapping the logical I/O in HDF5 layer, instead of tracing or computing the physical blocks accessing details at the disk level. The goal of our prediction model is to estimate I/O performance accurate enough to make a decision on selecting a data layout.

III. Scientific Data Services Framework

The goal of the Scientific Data Services (SDS) framework is to apply data management optimizations transparently without placing burden on scientific application developers. SDS is targeted to reorganize and replicate data to improve data access performance on large scale parallel computers, where data is generally stored in parallel file systems.

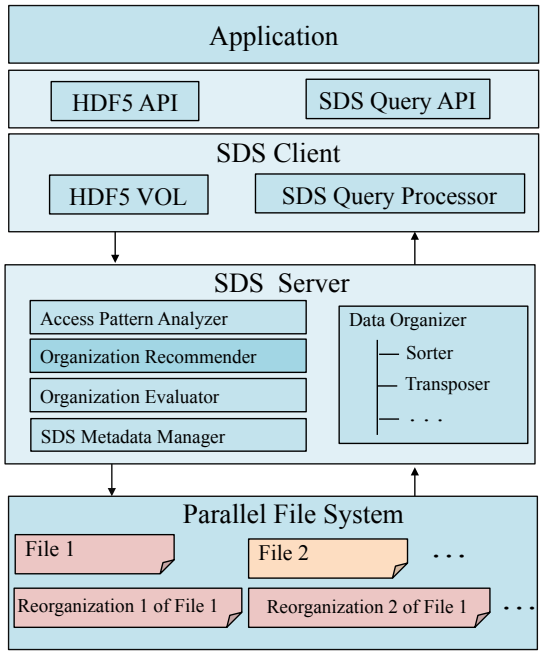

Figure 1: Overview of Scientific Data Services Framework

Figure 1 shows an overview of the SDS framework. We design SDS as a client-server architecture. The SDS Server analyzes the access patterns of I/O read calls, identifies the data layouts that benefit the read patterns, performs data reorganizations, and manages the metadata of the reorganized datasets. The Access Pattern Analyzer (APA) traces frequently accessed files, variables, and the offsets (data locations) of the read accesses and identifies the read patterns. We define the read patterns as a plane in a multi-dimensional array, a sub-cube of an array, a sub-plane of an array, contiguous chunks of data, and non-contiguous chunks of data. These patterns are well-known in scientific data accesses [5, 17]. Advanced users can pass this information of the read patterns of applications to APA without the need for tracing. APA stores the frequently accessed pattern information as metadata. The SDS Metadata Manager, implemented using Berkeley DB, manages the metadata. The Organization Evaluator of the Server periodically reads the frequently read patterns and contacts the Organization Recommender to identify data layouts that will achieve better performance. After identifying the optimal layouts, the Data Organizer component invokes the functions such as Sorter and Transposer to organize partial or full replicas of the data on the parallel file system. The Sorter sorts data according to a given variable and the transposer transforms 3D array data to improve locality. These reorganization functions are pluggable to the SDS framework. We have described the functionality of the SDS design and implementation in our previous papers [9, 10]. In this paper, we focus our discussion on the performance prediction models that the Organization Recommender uses to detect optimal layouts.

IV. Parallel I/O Subsystem and I/O Model Formation

In this section we introduce the parallel I/O subsystem that we consider to model. We will explain the logical and physical views of data access patterns and the mapping of a read request to physical view. Based on that, we introduce our proposed I/O model.

A. Two-level Data Striping in the I/O Subsystem

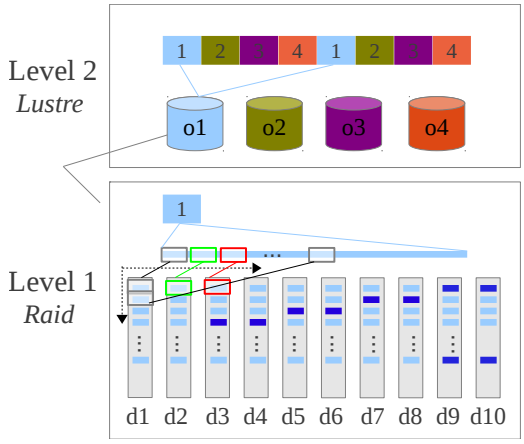

Figure 2: Two Level Data Striping in Lustre File Systems

The parallel I/O subsystem in most current supercomputers contains multiple layers of software libraries including a high-level I/O library (e.g. HDF5, PNetCDF, and ADIOS), middleware layer (MPI-IO), and parallel file system (e.g. Lustre, GPFS, and PVFS). In this study, we develop a model for the Lustre parallel file system with MPI-IO as middleware and HDF5 as data format. The Lustre file system uses Object Storage Servers (OSS) for managing data to be stored on Object Storage Targets (OSTs). Lustre allows parallel applications to read and write data concurrently to the file system. To support this mechanism, Lustre splits the

data into "stripes" and stores each stripe on one OST. If the stripes are more than the number of available OSTs, Lustre distributes the stripes on OSTs in a round-robin fashion. As shown in Figure 2, in a Lustre file system with 4 OSTs, stripes 1 and 5 are stored on $OST1$, and stripes 2 and 6 are stored on $OST2$, and so on. Applications and users can choose the size of a stripe and the number of OSTs to write data on a Lustre file system. An OST typically writes data to a RAID for achieving fault tolerance and speed. A RAID also splits an OST stripe into smaller equal sized chunks of data and stores it on multiple disks. For example, on Hopper supercomputer at National Energy Research Scientific Computing Center (NERSC), the $/scratch2$ Lustre file system contains 156 OSTs and each OST manages a RAID6 with 8+2 (8 disks for storing data and 2 disks for storing parity) organization. The stripe size of OST is configurable and the block size of RAID6 hardware is set to 512-bytes. As Figure 2 shows, data is distributed to the disks (labeled from d1 to d10) in a round-robin fashion. The minimum size of a read or write request to the RAID is 4 KB referring to eight 512-byte blocks. We consider this two-level striping model as the basis for our I/O performance model and prediction.

B. Logical and Physical I/O Patterns

Mapping the logical view of reading data to the physical organization of the data holds the key to improved read performance. The logical view of accessing large datasets is often expressed in terms of a multi-dimensional arrays. The physical organization of the data is flattened array stored on multiple storage targets and on numerous disks at a small 512-byte block granularity. Mapping these two views by reducing the number of small data requests and by improving parallelism increases the read performance. Towards that mapping, we define simple logical and physical representations of read patterns.

Many large-scale scientific applications use high-level I/O libraries, such as HDF5, ADIOS, and PNetCDF, for writing and reading data from/to parallel file systems. Analysis applications often read data using starting positions and length in each dimension of a multi-dimensional array. We represent this logical view using Equation 1.

$$I/O_{logical} = (Start[n], Length[n]) \quad (1)$$

Since multi-dimensional arrays are flattened and stored on multiple physical storage devices, typically disks, the read patterns become non-contiguous. We express the physical view of reading data using the blocks stored on RAID disks.

$$I/O_{physical} = (Block\ Depth, Block\ Gap) \quad (2)$$

We define the $Block\ Depth$ (BD) as the number of blocks to be accessed on one disk to satisfy a read request. The read time is dependent on the disk with the most number of blocks to be accessed. Hence, we use maximum BD in Equation 2, which refers to the maximum block depth among all the participating RAID disks on all OSTs that contain the data to satisfy a read request. For example, if a read request needs to touch 36 blocks of data distributed among 8 disks, the BD of 4 disks is 4 and that of the remaining 4 disks is 5. Then, the $Maximum\ Block\ Depth$ is 5. In Equation 2, $Block\ Gap$(BG) refers to the gaps among different blocks being accessed in a disk. BG is incremented when two blocks are neither logically contiguous nor physically contiguous.

Note that, the precise block distribution on storage hardware depends on various factors of the file system. Our representation of the physical view of data accesses is a simple model of a complex system. With the goal of predicting I/O performance trends quickly based on the read patterns, we kept the model simple.

C. Determining Maximum Block Depth and Block Gap

Pseudocode 1: I/O Logical to Physiscal Mapping

```
input : Dataset Dimension: n, Stripe Size: sts, Stripe
        Count: stc, I/O (s,l), Block Size: bks
output: Block Depth and Block Gap

Step 1: map I/O to each stripe
for i ← s_0 to s_0 + l_0 do
    ...
    for n... do
        index = i * ∏_{d=1}^{n} l_d + j * ∏_{d=2}^{n} l_d... + n;
        istripe = index/sts;
        iblock = (index mod sts)/(bks * 8);
    ...
Step 2: Compute BD
for i ← 0 to num(stripe) do
    for j ← 0 to num(block)/stripe do
        if iblock[i, j] > 0 then
            BD[i mod num(ost)] += 1;
Step 3: Compute BG
for i ← 0 to n do
    if fast dim then
        BG[i] = l[i]
    else
        BG[i] = (dim[i] − l[i])
```

We estimate the maximum BD and BG values for a given read request to a file stored on Lustre parallel file system with specific striping information using the I/O logical to physical mapping (IOLPM) function shown in Pseudocode 1. The IOLPM function takes the stripe count and stripe size, and the read pattern information including the starting offsets and lengths in different dimensions as input. The mapping function returns the BD and BG values as output. To calculate BD, the function uses the two-level striping

model to identify the number of blocks that would be accessed from each OST and then from each disk. The algorithm finds the number of I/O requests that go to each OST and the number of requests that go to each disk on RAID. We assume the the RAID6 model as we ran all our experiments on Hopper at NERSC. To calculate the Block Gap (BG), a tedious way is to use the intermediate result when calculating $Block\ Depth$, i.e., $index\_block$ and then accumulate the distance between two close blocks. Instead, we compute the difference between the full length of the dataset in an array dimension and the length of the read pattern in the same dimension. For example, a read request has 100 on one dimension (not the fast dimension), and this dimension of the full dataset has size of 1000, then the block gap on this dimension is 900.

V. I/O Performance Model

A. Model Definition

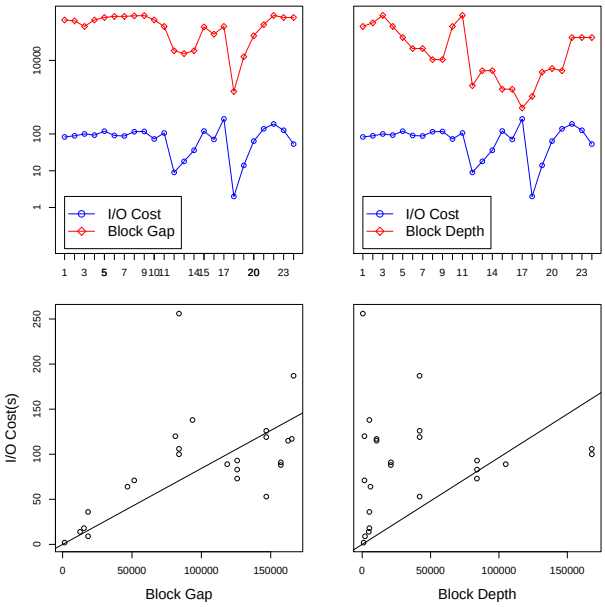

Figure 3: Linear Relations among Factors

We used linear regression analysis based on BD and BG to predict I/O time of a read request. We ran a benchmark to access data in different read patterns from a 3D array dataset with 1024 x 1024 x 65536 dimensions. The total size of the dataset is 128 GB and stored on a Lustre file system using 100 OSTs and with a stripe size of 1 MB. We use MPI-IO's collective I/O to access different 24 sub-cube patterns of data, e.g., 1024*1024*1024 or 100*1024*65536, etc. For these tests, we used 1024 MPI processes and 16 aggregators for MPI-IO collective buffering with a buffer size of 16 MB.

We plot the 'I/O Cost vs. Block Gap' and 'I/O Cost vs. Block Depth' to understand the relationships of BD and BG with the I/O cost and fitted regression lines. As shown in Figure 3, the BG fits into linear relations with the I/O Cost, whereas the BD does not fit into the linear model very well. The coefficient of determination (R^2), which indicates how well the I/O cost and BD, and the I/O cost and BG data points fit on a line, are 0.7 and 0.3, respectively. Typically, a value of $R^2 > 0.5$ indicates that the data fits the linear model well [8]. We also analyzed the statistical significance of BD and BG using $p - value$ calculation. The p-values of BG and BD are 0.000435 and 0.169462 separately, which means the BG is more significant than BD. This is reasonable since the disk seek time (related with BG) is usually 1000 times larger than the disk read time (related with BD). Since the leading factor, i.e., Block Gap, demonstrates a linear relation with the I/O cost, we choose the linear regression model, as shown in Equation 3.

$$F = \alpha \times BD + \beta \times BG + \varepsilon \quad (3)$$

In Equation 3, we have not considered the number of processes, the number of MPI-IO collective buffering aggregators, and the Lustre striping parameters, such as the number of OSTs and the stripe size. We refine this basic model further to consider various other parameters. In the first refinement, we use independent I/O without MPI-IO collective buffering, and in the second refinement we consider the collective buffering.

B. Model Refinement

Independent I/O. For independent I/O, each process carries out its own I/O, and different processes compete with each other when they access the same OST. In prior research, a queuing model is usually deployed to simulate the contention of concurrent processes on the server side. In our model, we select two cases to simplify the contention, i.e., 'best case' when processes come in the same direction with the disk head movement, and 'worst case' when the processes come in the reverse direction of the disk head movement. For the 'best case', concurrent processes act as a single I/O, which means the contention is absent. In the 'worst case', the number of concurrent processes on the same OST matters, and the increased latency leads to increased I/O cost. The Equation 3 is now rewritten as,

$$F = \alpha \times BD + \beta \times BG + \gamma \times Nproc/OST + \varepsilon \quad (4)$$

Collective I/O. MPI-IO collective buffering reduces the number of concurrent accesses to file system by individual MPI processes. This feature is targeted to reduce the number of metadata reads, which can become a performance bottleneck. Collective buffering implements reading data in two phases.

- Phase 1. The aggregators collect read request information from a set of MPI processes and perform the I/O in an aggregated way on the server side, i.e., I/O phase.

- Phase 2. Aggregators shuffle and send the data to the MPI processes on the client side, i.e., shuffle phase.

Both phases of collective buffering contribute the I/O cost. For the I/O phase, the cost includes BD, BG, the number of OSTs, and the number of aggregators. For the communication between aggregators and MPI processes, the communication cost is dependent on the number of communications and the data size. For the communication cost, we use the model similar to [7, 16].

To rewrite Equation 3, the two separate equations for the two phases of collective I/O are:

$$F_1 = \alpha \times BD + \beta \times BG + \gamma \times Nagg/OST + \varepsilon_1$$
$$F_2 = \lambda \times Nproc + \mu \times Nagg + \omega \times Size/Nagg + \varepsilon_2 \quad (5)$$

In Equation 5, F_1 represents the I/O phase on the server side, and F_2 represents the shuffle phase on the client side. The linear regression model for the total I/O cost is $F = F_1 + F_2$.

VI. Experimental Results and Analyses

We have evaluated the accuracy of the I/O model and the selection of data layouts including data transpositions and Lustre stripe settings. Based on our initial results of the accuracy, we refine the prediction model further. As mentioned earlier, we ran all the experiments on the Hopper, a Cray XE6 supercomputer with 6384 nodes. Each node has two 12-core AMD 'MagnyCours' 2.1 GHz processors and at least 32 GB memory per node. The Lustre file system we used has 156 OSTs with a 35 GB/s peak I/O bandwidth.

A. Layouts Selection with Common I/O Pattern

We have compared the predicted I/O performance with the measured performance for different read patterns accessing data from different data organizations. Our goal of this evaluation is to be able to predict the data organization that gives the best performance for a given read pattern. The read patterns include accessing a 1D array, a 2D sub-plane, and a 3D sub-cube. These patterns are similar to the common read patterns we have discussed in Section III. We used a synthetic benchmark for reading these patterns from a large 3D array with [1024, 1024, 65536] size in [x, y, z] dimensions. We stored data in [x, y, z], [x, z, y], and [z, x, y] dimensions that represent three different data layouts. We label these layouts as 1, 2, and 3, respectively.

IO	Measured Best	Predicted Best	Speedup
1D	(1,1,3,3,2,2,1,3)	(1,1,3,3,2,2,3,3)	128.42
2D	(1,2,3,3,2,1,1,2)	(1,2,3,3,3,1,1,2)	68.46
3D	(3,1,3,2,2,1,3,2)	(3,3,3,2,2,1,3,2)	9.26

Table I: Organization Selection based on Models

For each I/O pattern, we conducted 8 tests with 512 processes, with varying starting offsets and the lengths of read requests. We used independent I/O where each process accesses the file system. For example, in the tests with 1D pattern accesses, each process accesses a 1D line with varying lengths. We measured the I/O cost in accessing data stored in three different organizations: (1024,1024,65536), (65536,1024,1024), and (1024,65536,1024). We compared the best predicted I/O cost using Equation 4 for each test in accessing different organizations. We have also measured the best I/O cost with each test. Table I compares the measured best costs for each test in the three I/O patterns with the predicted best costs. For example, in the first row under "Measured Best" column, (1,1,3,3,2,2,1,3) refers to the list of data organizations that give the best performance for the 1D pattern tests. The "Predicted Best" list matches 7 out of the 8 tests with the measured best organizations. We show the predicted best cases that mismatch with the measured best cases in "red" color. We also show the average speedup achieved when accessing the selected layouts compared to the original layout, which is organization 1. The average speedup for 1D patterns is 128.42X , that for 2D patterns is 68.46X, and that for 3D patterns is 9.26X.

B. Improving the Accuracy of Prediction

In this section, we explore the mismatched "Predicted Best" and "Measured Best" cases further and refine our linear model to avoid such mismatches. We discuss the 3D pattern accesses. Figure 6 compares the predicted I/O costs (using Equation 4) and the measured I/O costs for the eight 3D access patterns to the three datasets (showing all 24 cases). We can observe that the error of predicted costs is higher in 8 of the 24 cases. The average residual error in these 8 mismatching predicted times is 126.8. The reasons for this error may be due to the contention in the system from other users accessing the I/O system simultaneously.

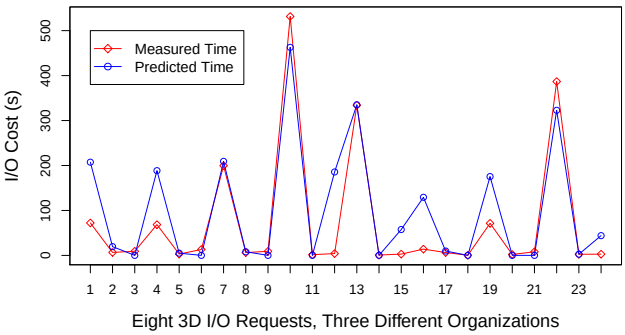

Figure 4: Empirical Model at Larger Scales

To improve the prediction accuracy, we applied "multiplicative linear regression model" [8] instead of the additive

linear model we used earlier. Using the 'multiplicative' model [3], Equation 4 can be refined as Equation 6.

$$\begin{aligned} F =& \alpha_1 \times BD + \alpha_2 \times BG + \alpha_3 \times Nproc/OST \\ &+ \alpha_4 \times BD \times BG + \alpha_5 \times BD \times Nproc/OST \\ &+ \alpha_6 \times BG \times Nproc/OST \\ &+ \alpha_7 \times BD \times BG \times Nproc/OST + \varepsilon \end{aligned} \tag{6}$$

With a multiplicative model, the interactive effect of variables are measured, e.g., $BD \times BG$. This is reasonable in the real system, because with different $Block\ Gap$, the blocks could be located on the outermost tracks, innermost tracks, or across the entire disk surface, such that the disk read speed can be significantly different to access same $Block\ Depth$ [13]. The equation for calculating the I/O phase of collective I/O is similar to that of Equation 6, where $Nproc$ is replaced with $Nagg$.

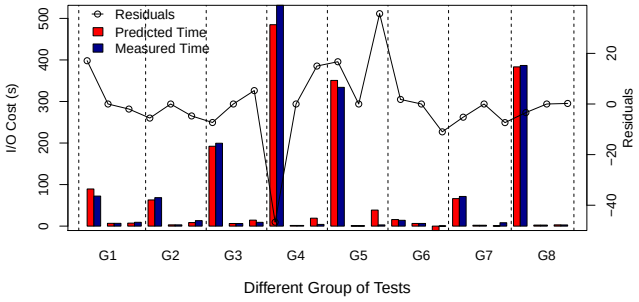

Figure 5: Comparison of Predicted and Measured I/O times for Independent I/O with 1024 Processes

Figures 5 and 6 show the comparison of predicted and measured I/O costs for accessing 3D pattern sub-cube data using MPI independent I/O and collective I/O, respectively. Each plot compares 24 tests with 8 variations of 3D patterns accessing data from three layouts. We also show the 'Residual error' in the plots.

The average residual error in the independent I/O case is now reduced to 40.7. The coefficient of determination (R^2) is 0.9951, which indicates close fitting of the linear regression and improved accuracy of prediction.

The average residual error in the collective I/O case (Figure 6) is 3.95. The coefficient of determination (R^2) is 0.9969, which is better than the independent I/O case.

We have applied the refined models for selecting organizations. We ran three 3D pattern tests for accessing data from three organizations and the "Predicted Best" organizations matched with the "Measured Best" cases with no errors for both independent I/O and collective I/O tests. This proves that the model is capable of selecting data layouts for

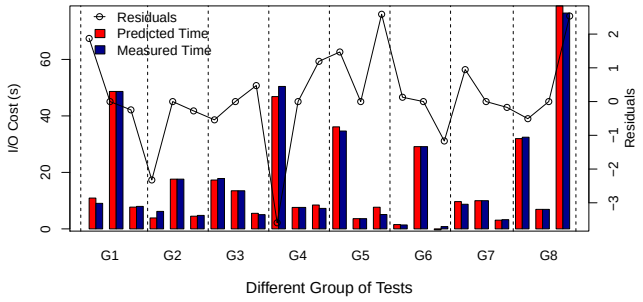

Figure 6: Comparison of Predicted and Measured I/O times for Collective I/O with 1024 Processes

improving read performance.

VII. Conclusions and Future Work

Many scientific discoveries heavily rely on the collection and analysis of large quantities of data. The performance of reading scientific data has been a critical factor in determining scientific discovery productivity throughout these analysis processes. In this study, as observing read performance is tightly related with the organization of data on physical storage devices, we try to answer the question of how data organization is correlated with the read performance and how an optimal data organization can be automatically determined and selected to maximize scientific discovery productivity.

The contributions of this study are three-fold, and the conclusions of this study can have a broad impact. First, we identify and show that maintaining multiple layouts plays an important role in scientific discovery and can have significant performance benefits. Second, We design a new parallel I/O performance model to select optimal data organizations for different I/O request patterns. By transforming the logical I/O requests to the physical I/O accesses, our model predicts the I/O performance for various data organizations and redirects the I/O to an optimal layout automatically. Third, we have integrated this model-driven data layout automatic selection into the Scientific Data Services (SDS) framework we are developing. We have applied the model-driven layout selection approach in SDS and evaluated performance of reading data based on the heuristic guidance provided by the performance model. In the evaluations, we verified the model with respect to different transposing organizations. The results confirmed that the linear regression model we have introduced can distinguish different layouts, and the SDS framework integrated with the performance model was able to select the optimal data layout for incoming I/O requests in the majority of cases. The proposed model-driven data layout automatic selection can be a new methodology

that guides file systems and parallel I/O libraries to better meet growing data-intensive scientific discovery needs, and the integration with the SDS framework can have direct impact on scientific applications with enhanced productivity.

For future work, we will further investigate the model to enhance the prediction performance and reduce the real time overhead. Besides, we will use this model to re-define the I/O pattern by grouping the I/O based on their model prediction cost, such that we can use the SDS to quickly detect the I/O pattern and redirect to the better layout. We are also seeking to design a partial layout selection based on the model, in which, not all the data are replicated to another layout, only a small portion of hot data are replicated.

VIII. Acknowledgment

This work is supported in part by the Director, Office of Laboratory Policy and Infrastructure Management of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, by the National Science Foundation under grant CNS-1162488, and used resources of The National Energy Research Scientific Computing Center (NERSC). We thank the members of Scientific Data Management group at LBNL, especially Arie Shoshani, for their constructive feedback. We are also thankful to Steve Luzmoor (Cray) and Andreas Dilger for providing the details of Lustre and Cy Chen (LBNL) for his comments on modeling I/O.

References

[1] IPCC Fifth Assessment Report. http://en.wikipedia.org/wiki/IPCC_Fifth_Assessment_Report.

[2] J. Bent, G. Gibson, G. Grider, et al. PLFS: a checkpoint filesystem for parallel applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, 2009.

[3] T. Brambor, W. R. Clark, and M. Golder. Understanding interaction models: Improving empirical analyses. *Political Analysis*, 14(1):63–82, 2006.

[4] J. S. Bucy and G. R. Ganger. The DiskSim simulation environment version 3.0 reference manual. Technical Report CMU-CS-03-102, Carnegie Mellon University, School of Computer Science, Jan. 2003.

[5] 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 *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, 2008.

[6] S. Byna, J. Chou, O. Rübel, Prabhat, H. Karimabadi, et al. Parallel I/O, analysis, and visualization of a trillion particle simulation. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 59:1–59:12, 2012.

[7] K. Cha and S. Maeng. Reducing communication costs in collective I/O in multi-core cluster systems with non-exclusive scheduling. *The Journal of Supercomputing*, 61(3):966–996, 2012.

[8] J. Cohen and P. Cohen. *Applied multiple regression/correlation analysis for the behavioral sciences.* Lawrence Erlbaum, 1975.

[9] B. Dong, S. Byna, and K. Wu. Expediting scientific data analysis with reorganization. In *Proc. of the IEEE International Conference on Cluster Computing (Cluster'13)*, 2013.

[10] B. Dong, S. Byna, and K. Wu. SDS: A framework for scientific data services. https://sdm.lbl.gov/ sbyna/research/papers/2013-p2013-SDS.pdf, 2013.

[11] H. Finkel. Porting and Tuning HACC on Mira. Technical report, Available at: http://www.alcf.anl.gov/sites/www.alcf.anl.gov/files/darkuniverseesptechreportwrapped.pdf, 2013.

[12] M. Francis. Future telescope array drives development of exabyte processing. Technical report, Available at: http://arstechnica.com/- science/2012/04/future-telescope-array-drives-development-of-exabyte-processing/, 2012.

[13] Hard Drive: Inner or outer. http://www.pythian.com/blog/hard-drive-inner-or-outer/.

[14] T. Hey, S. Tansley, and K. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft, Oct. 2009.

[15] A. Kalyanaraman, S. Aluru, S. Kothari, and V. Brendel. Efficient clustering of large est data sets on parallel computers. *Nucleic Acids Research*, 31(11):2963–2974, 2003.

[16] J. Liu, Y. Chen, and Y. Zhuang. Hierarchical I/O scheduling for collective I/O. In *the Proc. of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'13)*, 2013.

[17] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu. Six degrees of scientific data: reading patterns for extreme scale science I/O. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 49–60, 2011.

[18] ORIONNetwork. The ATLAS experiment: Big data and the hunt for the god particle. http://www.orion.on.ca/wp-content/uploads/2012/04/BIGideas19-higgs.pdf, 2012.

[19] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar, and M. Best. Characterizing parallel file-access patterns on a large-scale multiprocessor. In *9th Intl. Parallel Processing Symp. (IPPS)*, pages 165–172, Apr. 1995.

[20] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proceedings of the Tenth International Conference on Data Engineering*, pages 328–336, 1994.

[21] H. Shan, J. Shalf, and K. Antypas. Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. In *SC'08*. ACM/IEEE, Austin, TX, Nov. 2008.

[22] E. Smirni, C. L. Elford, D. A. Reed, and A. A. Chien. Performance modeling of a parallel I/O system: An application driven approach. In *PPSC*. SIAM, 1997.

[23] H. Song, Y. Yin, Y. Chen, and X.-H. Sun. Cost-intelligent application-specific data layout optimization for parallel file systems. *Cluster Computing*, 16(2):285–298, 2013.

[24] The Computing Community Consortium (CCC). Advancing Discovery in Science and Engineering: The Role of Basic Computing Research. http://www.cra.org/ccc/files/docs/Natl_Priorities/web_data_spring.pdf, September 2011.

[25] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, et al. EDO: Improving Read Performance for Scientific Applications through Elastic Data Organization. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER'11.

[26] Y. Tian, S. Klasky, W. Yu, H. Abbasi, B. Wang, N. Podhorszki, R. Grout, and M. Wolf. Smart-io: System-aware two-level data organization for efficient scientific analytics. In *2012 IEEE 20th International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012.

[27] M. Wang. Dissertation: Storage device performance modeling using machine learning techniques, 2008.

[28] 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 *IPDPS*, pages 345–356. IEEE Computer Society, 2013.

[29] H. You, Q. Liu, Z. Li, and S. Moore. The design of an auto-tuning I/O framework on Cray XT5 system. In *Cray Users Group Conference (CUG'11)*, Fairbanks, Alaska, may 2011.

[30] S. Yu, M. Winslett, J. Lee, and X. Ma. Automatic and portable performance modeling for parallel I/O: A machine-learning approach. *ACM SIGMETRICS Performance Evaluation Review*, 30(3):3–5, Dec. 2002.