Title
Usage Pattern-Driven Dynamic Data Layout Reorganization:

Authors
Tang, Houjun
Byna, Suren
Harenberg, Steven
et al.

Publication Date
2016-05-16

Usage Pattern-Driven Dynamic Data Layout Reorganization

Houjun Tang [1,3], Surendra Byna [2], Steven Harenberg [1,3], Xiaocheng Zou [1,3], Wenzhao Zhang [1,3], Kesheng Wu [2], Bin Dong [2], Oliver Rübel [2], Kristofer Bouchard [2], Scott Klasky [3], Nagiza F. Samatova [1,3,*]

[1] North Carolina State University, [2] Lawrence Berkeley Laboratory, [3] Oak Ridge National Laboratory

[*] Corresponding author: samatova@csc.ncsu.edu

***Abstract*—As scientific simulations and experiments move toward extremely large scales and generate massive amounts of data, the data access performance of analytic applications becomes crucial. A mismatch often happens between write and read patterns of data accesses, typically resulting in poor read performance. Data layout reorganization has been used to improve the locality of data accesses. However, current data reorganizations are static and focus on generating a single (or set of) optimized layouts that rely on prior knowledge of exact future access patterns. We propose a framework that dynamically recognizes the data usage patterns, replicates the data of interest in multiple reorganized layouts that would benefit common read patterns, and makes runtime decisions on selecting a favorable layout for a given read pattern. This framework supports reading individual elements and chunks of a multi-dimensional array of variables. Our pattern-driven layout selection strategy achieves multi-fold speedups compared to reading from the original dataset.**

I. Introduction

Large-scale scientific simulations and experiments produce massive volumes of data. This data is typically stored on a parallel file system in an organization (layout) that is optimal for writing and remains fixed afterwards. However, scientific data is often written once and read many times and the organization of the written data may not be efficient for the read patterns used in data analysis operations. For example, scientific simulations such as S3D combustion [5] and GTS core plasma fusion [24] write the data of *all* variables by time steps, yet analysis and visualization applications often read a *subset* of variables within a specific region over a number of time steps. Such mismatches between a write layout and a read pattern lead to poor read performance due to a large number of seek and read operations to hard disk-based file systems. This issue is exacerbated by the advancement towards exascale computing, leading to ever-increasing dataset sizes and thus presenting challenges to data management and I/O optimization for efficient data access.

To address this data layout mismatch issue, many layout reorganization methods have been proposed to increase the number of contiguous I/O accesses. For instance, *space-filling curves*, such as Hilbert-curve and Z-curve, are used to reorganize the original data [23]; array *transposition* is applied to create multiple full replicas [15] of data; and *merging* of multiple non-contiguous data blocks to a single contiguous chunk to create partial replicas [27], [13]. Each of these techniques has its own characteristics. Space-filling curves bring performance benefits to sub-region accesses by reorganizing the dataset and they require no additional storage when only the original data is reorganized. Array transposition leads to better performance for accesses that have significantly larger sizes along one dimension. However, transpositions may require multiple replicas of the data. Specialized merging with partial replication results in better performance, as the previously non-contiguous data accesses become contiguous.

Despite various advantages of reorganization, none of the strategies alone can provide near-optimal read performance for heterogeneous patterns of analytic applications. To support multiple read patterns, there is a need for managing different layout strategies. These organizations shall facilitate commonly used spatial selections defined by multi-dimensional bounding boxes as well as by element (point) selections. As storage space for managing multiple full replicas is expensive, support for managing partial replicas considering the storage budget is necessary. Transparent redirection of accesses to the data with preferable layout that may match fully or partially are required as well. To the best of our knowledge, a framework supporting these requirements is absent in scientific data management.

In this paper, we present the dynamic data reorganization framework that performs dynamic data access pattern tracing and identification functions, efficient storage of partial replicas to support multiple read patterns, and redirection of read accesses to a favorable layout at runtime. Ultimately, the layout decision making (Section III-B) and layout reorganization (Section III-D) methods of our framework will be integrated as services into our recently proposed Scientific Data Services (SDS) [26], [8]. Our data reorganization framework shows a broader applicability compared to existing methods, enabled by the following contributions.

Dynamic pattern identification. Our framework automatically traces read accesses and identifies the data usage patterns during an applications' runtime. The current implementation supports the HDF5 [22] library in tracing bounding box selections, known as *hyperslabs* in HDF5, and element selections.

Flexible multi-layout management with storage budgets. Instead of using only one layout reorganization technique, we provide more flexibility by supporting multiple layout reorganization techniques among those shown in Figure 1. Based on the user-specified storage constraint and current usage patterns,

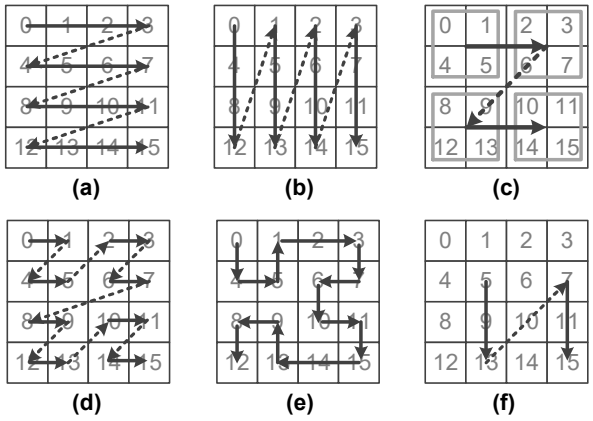

Fig. 1. Data reorganization techniques that our framework supports. The numbers in each cell are the starting offsets of original data, and the arrow lines are the order of the reorganized offsets: (a) (original) row-major layout, (b) column-major (transposition) layout, (c) blocked (chunking) layout, used as a pre-processing step before applying (d) and (e), (d) z-curve, (e) Hilbert-curve, (f) custom merging of a subset (data at offsets $5, 9, 13, 7, 11, 15$).

our framework evaluates the costs of reorganization and the benefits with accessing the reorganized data to select the most suitable technique.

Runtime decision making with partial match and redirection. By allowing partial matches between read patterns and the reorganized replicas, we extend the usability of existing layouts compared to the exact match strategy from previous work [13]. For a more accurate cost estimation, a page-level (instead of byte-level) cost model is used for estimating data access cost during the decision making process. Further, we enabled automatic read redirection to the model-selected layout for improved performance.

The remainder of this paper is organized as follows: We briefly discuss the related work in Section II. In Section III, we present a high-level overview of the proposed data reorganization framework and describe the functionality of different components. We demonstrate the application of the framework in Section IV using read patterns from multiple real applications and conclude our discussion in Section V.

II. Related Work

The linearization and organization of the extreme-scale datasets on parallel file systems are crucial to the scientific application's performance. SciDB [2] addresses sub-volume access patterns by applying array slicing, joining operations, and array division into regular/irregular chunks for multi-dimensional scientific data. EDO [23] optimizes sub-plane and sub-volume access patterns for spatial locality through Hilbert space-filling curves reordering and sub-chunking. However, SciDB and EDO both reorganize the original data layout and provide average performance for sub-region accesses. In contrast, we present a framework that maintains replicas with layout optimized for the common access patterns in scientific data explorations.

Chunking is another layout optimization technique that splits the dataset into multiple chunks and improves performance when operating on a subset of the data [12]. However, current approaches [20], [17], such as those supported by HDF5, are not flexible enough to meet the need for the dynamic patterns discussed in this work. The chunking is applied with fixed chunk dimensions and cannot be modified afterward, limiting its applicability for non-regular spatial patterns. OpenMSI [19] adopted chunking, compression, and data replication to improve the data access efficiency for MSI datasets. While it focuses on a specific domain and pre-generates all the reorganized replicas, our framework covers broader types of data usage patterns and dynamically adapts to the change of patterns.

MLOC [11] proposed a parallel layout optimization framework to achieve better performance for queries on scientific datasets with heterogeneous access patterns. Though multiple layouts are discussed, it focuses on one layout at a time and does not provide either runtime decision making or layout management. In contrast, our work supports multiple layouts chosen based on the users' specification as well as runtime decision making.

Given additional storage, creating multiple partial replicas, each optimized for a specific kind of access pattern, can greatly improve the read performance and meet the need of heterogeneous access patterns of scientific applications. PDLA [27] explored data replication for patterns with high regularity and selected from three layouts: 1-DH, 1-DV, and 2-D on the parallel file system. Earlier we introduced RADAR [13], which maintains partial replicas and selects the most optimized one for current access pattern during run-time. However, both approaches are limited to regular spatial patterns, with no optimizations for patterns induced by element selection.

The Scientific Data Services (SDS) system proposes to apply data management optimizations transparently without placing burden on scientific application developers [26], [8], [9]. One of the services SDS proposes is to reorganize and to replicate data on parallel file systems. SDS has a client-server architecture. The server would analyze the access patterns of I/O read calls, identify the data layouts that benefit the read patterns, perform data reorganizations, and manage the metadata of the reorganized datasets. SDS requires to traces the accessed files, variables, and the offsets (data locations) of the application's reads and pass them to access pattern analyzers for identifying read patterns. The analyzed results can then be stored as metadata and managed by SDS Metadata Manager, which is implemented using Berkeley DB. With the identification of the data usage patterns, the layout reorganizer will create replicas with optimized layout for the patterns. These replicas will be used for future accesses that have same or similar patterns. While SDS has capabilities and a framework to perform reorganizations and to redirect data accesses to suitable layout, several components are yet to be developed. For instance, components for capturing data accesses, for analyzing and detecting read patterns dynamically, and for estimating costs and benefits are still missing in SDS. The methods developed in this study, i.e., trace capturing and analyzing, layout decision making, and layout reorganization methods, are planned to be integrated into SDS to provide a broader applicability than its current implementations.

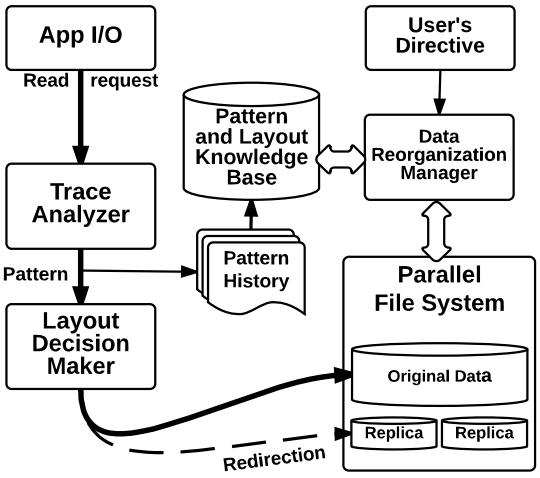

Fig. 2. At runtime, the framework traces and detects patterns of I/O and decides on optimal data layouts. The layout management, i.e., replica creation according to the optimal layouts is performed offline. When optimal layouts are available, redirection of file read calls to the replicated data is performed at runtime using binary instrumentation. The right side of the figure shows the components of the dynamic reorganization framework.

III. Dynamic Data Reorganization

We present an overview of the proposed dynamic data reorganization framework in Figure 2. The main components of the framework are *Trace Analyzer*, *Layout Decision Maker*, *Pattern and Layout Knowledge Base*, and *Data Reorganization Manager*. The Trace Analyzer uses a binary instrumentation method to trace I/O read calls and to identify data access patterns. Our current implementation supports the HDF5 library to trace hyperslab (a subset of a multi-dimensional array) definitions that access bounding box and element selections. We have developed in this work a cost model (Section III-B) to predict the number of disk drive page accesses by a read access pattern. The Layout Decision Maker analyzes the cost of accessing data using the available layouts of the requested data and selects a layout that would give the best access performance. The supported layout reorganization techniques, as shown in Figure 1, are designed as plugins so new layouts can be easily added for more specialized optimization. The Data Reorganization Manager uses the suggestions of improved layouts to reorganize and replicate data with optimized layouts. An advanced user can initiate a request to the Data Reorganization Manager to reorganize data. When multiple replicas of the data in different layouts are available, the Layout Decision Maker dynamically redirects the read calls to the selected replica for obtaining the best performance. The metadata related to the available layouts and data access pattern history are managed in the Layout and Pattern Knowledge Base. We discuss each of these components in detail in the following subsections.

A. Trace Analysis and Pattern Detection

The first step to understand the data usage of applications is tracing the I/O read calls and identifying patterns. Motivated by existing work [27], [13], [21], we characterize data usage patterns in accessing a particular dataset, by focusing on three major aspects: (1) variables within a dataset being accessed, (2) accessed region (one or more sub-planes or whole plane, one or more sub-volume or whole volume, scattered points) of variables, and (3) the size of the requests.

The runtime pattern detection operation is performed first by extracting the relative information from HDF5 read calls issued by the running application. Similar to our previous work [21], this operation is performed within each MPI process and we keep the related information in an auxiliary data structure. We then analyze the data selection information to identify patterns. For HDF5 and other I/O libraries such as NetCDF and ADIOS, element (point) and bounding box selections are the two typical types of data selection provided to users that result in different patterns. We use a compact representation for the identified patterns, as shown in Section III-C.

To identify different patterns induced by element and hyperslab selection, our framework first checks the selection type and then records the data selection information during runtime. This information is then used for selecting a high performant layout (Section III-B) and if necessary for creating an additional replica in offline layout management (Section III-D).

1) Bounding Box Selection: Many analysis applications read data from a variable that is bounded by spatial locations defined by multi-dimensional array coordinates. As categorized by Lofstead et al. [16], in a 2D array, this bounding box region is referred as sub-plane or a whole plane and in a 3D dataset, the region is called sub-cube. In HDF5, the bounding box selections are called *hyperslabs*. A HDF5 hyperslab selection can be regarded as a complex bounding box selection. It allows users to select multiple bounding boxes with arbitrary regions using set operations (e.g. intersection, union, etc.). Such flexibility simplifies users effort to read their interested data regions in one read function call. Dealing with complex definitions of hyperslab challenges the existing work (such as [23], [27], [13]), which deals with one bounding box selection at a time. One such example is when accessing a labeled dataset, where the data is partitioned into chunks and each chunk has a different label. The data of one label is scattered in a file and is determined by an auxiliary index (See Section IV-E for more details).

2) Element Selection: Element selection is commonly used when a query library is involved, where the coordinates of typically scattered elements are given and the corresponding data need to be read from file. The coordinates can often be obtained fast with indexing techniques such as FastBit [25] and ISABELA-QA [14]. However, reading the data often results in extremely low I/O throughput due to the large number of non-contiguous reads with small request sizes. The capability to optimize for such patterns would bring huge read performance improvements and thus motivates us to explore the methods for such optimization.

To optimize data reads, we only assume the coordinates of the data selected as input, specifically, we do not require that the high level criteria on which the selection was based (e.g., range query) is known. As a result, our optimization is generic and can benefit the existing indexing techniques directly.

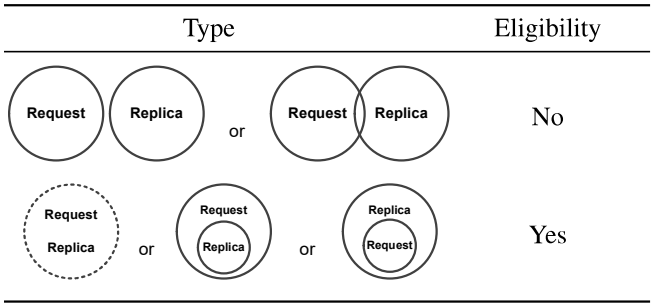

Fig. 3. The eligibility for a replica to be a candidate is determined by how much overlap it has with the read request. A replica is not eligible with no or partial overlap region while eligible in other cases.

For example, in the analysis and visualization of the VPIC dataset [3], only the particles with high energy are of interest. And thus a small subset of elements would be repeatedly accessed in a sequence of queries with value constraints such as $Energy > 1.3$, $Energy > 1.5$, $Energy > 1.8$, etc. By clustering those scattered elements with an intrinsic correlation into a contiguous chunk of data in file, and storing their original offsets, a large amount of time can be saved when future accesses include these elements. More details about the "clustering" part will be elaborated in Section III-D.

B. Layout Decision Making

The layout decision maker uses the pattern information recorded in the detection process and attempts to find the best matching replicas. The layout decision making process consists of two main steps:

1) Step 1: Candidate Selection: The layouts that cannot satisfy the request are first pruned to avoid the potentially large overhead of iterating through all layouts and loading their metadata. To be I/O efficient, a storage-lightweight catalog containing the start and end offsets for each existing layouts is maintained and used for the first round of coarse-grained pruning. The coarse-grained pruning prevents loading all metadata files. Another round of fine-grained pruning is performed, which loads the rest of metadata (the exact regions of data that a replica contains) of the remaining layouts and compared them with the requested data regions. A candidate set of potential replicas is generated using the following rules shown in Figure 3.

Note that we consider replicas that partially overlap with the requested data as not eligible. This is because the overlapping regions cannot be estimated accurately without loading the metadata of a replica. Accessing the metadata, especially for element selections (mappings to the original dataset), results in non-negligible I/O time as the size of metadata grows linearly with the data. Though it is best when the data of a replica is exactly the same as a request, we found performance improvements using the replica in two cases: 1) when the request region is larger than the replica, splitting the read request to read the entire replica and the rest is still beneficial especially when the overlapping region is relatively large (see Section IV-B); and 2) when the replica contains more data than the request, using the replica results in more contiguous accesses than using the original dataset and is expected to provide better read performance before a specific replica for that pattern is created.

TABLE I
PARAMETERS IN THE COST ANALYSIS MODEL.

Symbol	Meaning
N^i_{sp}	Number of I/O client processes accessing OST i
N^i_{pg}	Number of contiguous pages need to be accessed within OST i
N^i_{chk}	Number of non-contiguous chunks need to be accessed within OST i
T_{pg}	Average cost of reading contiguous blocks of data per page
T_{chk}	Cost of reading non-contiguous chunks on one storage node

2) Step 2: Layout Ranking Via Cost Model: The candidate replicas are ranked and the final decision is made via our page-level cost model. Inspired by the previous research work from [27], [13], we adopt a similar model with adjustments that better estimate the costs. As opposed to byte-level cost model, we chose to use a page-level cost model as it more accurately reflects the file read cost, especially in cases of element selection. The estimated read time T_r for replicas with different layouts is defined as follows (parameters defined in Table I):

$$T_r = \max\{(N^i_{pg} \cdot T_{pg} + N^i_{chk} \cdot T_{chk})N^i_{sp} \mid \forall i \in \mathcal{O}\} \quad (1)$$

where $\mathcal{O}$ is the set of Object Storage Targets (OSTs). Based on the request and the layout, we "flatten" the requested region into linear space and calculate N^i_{pg} and N^i_{chk} within the data stored on each OST. This cost model estimates the total time needed when reading data across multiple OSTs, and assumes each OST offers the same I/O rate as well the network and storage latency. T_{pg}, T_{chk} are measured beforehand and vary in different systems. The network and storage latency are considered as constants when comparing between the cost of different layouts, and are not included in the model. We compare the cost for all eligible replicas and the one with smallest T_r is selected.

3) I/O Redirection in HDF5: Once the layout decision is made, our framework automatically directs the read to the chosen replica. The replica's metadata such as the file and variable's path and name and the mapping of the layout to the original dataset is stored as part of the metadata, and becomes effective for the actual read. We have modified the related HDF5 read functions with our data selection procedure. When no replica is available, the normal HDF5 read process is used. If any replica is selected by our framework, the normal HDF5 read is skipped and the corresponding data in the replica is read instead.

C. Pattern and Layout Knowledge Base

To prepare the information needed for layout reorganization, our framework performs historic usage pattern analytics each

time new patterns are discovered during runtime. It is an offline incremental analysis process that extracts and analyzes data usage patterns. It is based on the previous results from runtime pattern analysis (Section III-A) with two more aspects added: (1) when and how many processes are issuing read requests together, (2) total size and I/O throughput. These are obtained after a read call completes. Based on the above aspects, our framework adopts a data usage patterns representation as {**variable name/path, selection type and spatial region, process IDs, start/end time, total size, I/O rate**} Each read request results in one such record and is inserted into the pattern history and the most important aspect is the spatial region.

A pre-processing step is performed to generate global patterns by merging the local patterns of each MPI process. The global pattern provides necessary information for the later data placement among OSTs (Section III-D3). Analysis of these patterns produces new information such as the "hot" data regions and pattern frequency for a dataset. This information, together with the metadata from existing layouts (replicas), is maintained in the "Knowledge Base". The layout metadata includes the replica's original file and reorganization information. Our framework automatically loads information from the knowledge base when the application starts. For offline layout management, the knowledge base supplies information to layout manager for layout creation and deletion.

D. Data Layout Reorganization

Layout management includes three tasks crucial to read performance: *replica creation* (when and how to create a new layout), *replica eviction* (which to remove when exceeding a user's storage budget), and *replica placement* (how the data is distributed among OSTs). This management occurs in an offline fashion, when the application terminates, to avoid runtime I/O contention. We assume that the data resides in parallel file systems such as Lustre, and the replicas with their metadata are stored in a special directory under the same directory with the original dataset.

1) Replica Creation: The layout manager makes the decision of when and how to create a new layout given the information from the knowledge base. This knowledge base can initialized with two options: (1) our framework can "learn" and decide what and how to perform layout optimizations, which takes effect after a few runs; or (2) users can instrument our framework with the patterns from their knowledge, allowing performance improvements at the first use. Three common replica creation scenarios are considered with the corresponding strategy that our framework automatically selects:

1) **The original dataset can be reorganized and limited additional storage space is allowed:** "Concatenation" is used to create partial replicas when existing replicas cover none of a trivial portion of the request (e.g., Section IV-B).
2) **The original dataset cannot be modified, but unlimited storage space is allowed:** "Concatenation" is used to create as many as possible replicas (e.g., Section IV-E).
3) **The original dataset can be reorganized but no additional storage is allowed:** Transposition or space-filling curves are used (e.g., D2 scenario 1 of Section IV-C).

The use of transposition and space-filling curves is thoroughly discussed in existing research [15], [23]. We provide more information on concatenation, which was explored in [27], [13], but only for single bounding box selection with high regularity of spatial patterns (kd-strided). To complement their work, our method adds support for both hyperslab selection with non-regular spatial patterns as well as element selection. We describe concatenation as follows: with a data selection that contains multiple chunks of non-contiguous data, the data between the chunks are removed and all the chunks are concatenated into one big contiguous chunk. By storing this big chunk as a partial replica, when the same (or overlapping subset) data selection occurs, the big chunk can be read all at once and thus brings read performance improvements.

2) Replica Eviction: Storage efficiency is achieved through the analysis of overlapping regions between new patterns and existing ones, and the deletion of old, less frequently used replicas. When the additional storage reaches the user-defined budget, the replicas are ranked according to a combination of their recent usage, size, and effectiveness (performance improvement $time_{old}/time_{new}$); older and less effective replicas are dropped to make space for new ones. As with maintaining layouts, each replica is associated with one "metadata" file containing the mapping to the original file. A separate "range" file is maintained for each dataset and stores the start and end offset of each replica for fast pruning as discussed in Section III-B.

3) OST-Aware Replica Placement: Even when the right layout organization technique is selected, read performance can still be far from ideal when treating the parallel file system (PFS) as a black box. Popular PFSs, such as Lustre and PVFS, use striping for data distribution among multiple storage devices. In Lustre file system, Object Storage Targets (OSTs) are for storing data. The data distribution on Lustre

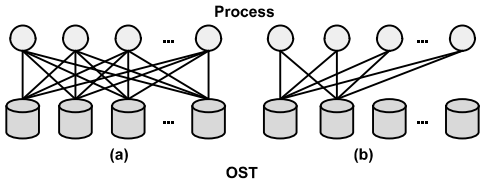

Fig. 4. Two bad cases where each OST is accessed by multiple processes. (a) Each process access data from all OSTs. (b) Each process access data from a subset of OSTs.

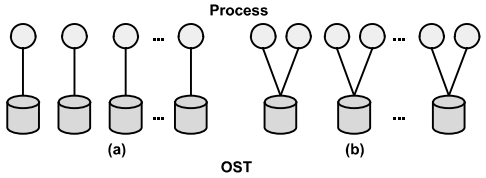

Fig. 5. Two ideal cases where each OST is only accessed by minimal number of processes. (a) The number of processes equals to that of OSTs. (b) The number of processes is larger than that of OSTs.

is decided by the stripe count (how many OSTs to use) and the stripe size (size of data to write to one OST before moving to the next). Even with manual adjustment, such simple striping strategy can not provide efficient read accesses. IO contention [10] is yet another important factor that affect overall read performance. Processes reading contiguous data with size comparable to $stripe\_count * stripe\_size$ would lead to the scenario where one OST is accessed by many processes, yielding contention (see Figure 4) that significantly degrades the overall read performance.

To avoid OST contention, the best practice is to make each OST be contacted by as few processes as possible–a guiding principle in our framework. Our framework analyzes the usage patterns (Section III-C) and rearranges the data so that future accesses that are the same, or similar, will have low read contention, as shown in Figure 5.

IV. Results

A. Overview of Evaluation

To demonstrate the effectiveness of our framework, we evaluated the read performance of kernels extracted from five different scientific applications or datasets from various science domains. We ran all our experiments on a Cray XC30 supercomputer named "Edison" at the National Energy Research Scientific Computing Center (NERSC). The data was stored on a Lustre file system equipped with 36 Object Storage Servers (OSSs) and 144 Object Storage Targets (OSTs). All available OSTs were used for storing data and each experiment was repeated multiple times to obtain consistent performance results. To avoid effects of caching, multiple copies of the data were created to guarantee that the same file is not accessed by any two consecutive experiments.

I/O kernels from five different real scientific applications were used in our experiments. These include: (1) querying a 188 billion particle plasma physics data produced by Vector Particle-In-Cell (VPIC) simulation of magnetic reconnection phenomenon [3] to demonstrate the support for element selection and partial match (Section IV-B); (2) accessing climate model and observation data, used for detecting atmospheric rivers (AR) [4], to demonstrate the ability of applying historical optimization strategies on new datasets (Section IV-C); (3)accessing Electrocorticography (ECoG) data [1] to demonstrate the support for non-regular patters with an ability to perform optimizations for different data regions (Section IV-E); (4) accessing data from Mass Spectrometry images [19] to show that our framework is able to support and manage different layout reorganization techniques at the same time (Section IV-D); and (5) accessing block-structured adaptive mesh (AMR) data [6] to show that our framework supports AMR data in addition to uniform grid data and point-based data (Section IV-F). These kernels represent the read accesses of several scientific data analysis applications. All datasets used HDF5 file format [22].

For each experiment, we compare the I/O time between accessing the data in its default layout (row-major), with unmodified HDF5, and that of the reorganized layout selected by our framework. In the climate and mass-spectrometry imaging accesses, we further compare the performance with two scenarios: with and without the availability of additional storage. In the case of no additional storage available, our framework selects one from transposition, Z-curve, and Hilbert-curve based on the pattern. This new layout replaces the original data file.

We demonstrate replica creation only in the AR detection experiment, and omit this process in the other experiments. Our framework initially creates replicas in an aggressive way, as the allowed storage space is sufficient. That is, data is replicated with the selected layout even if it is accessed once. When the total available storage becomes low with the increase of replicas, only the more frequently accessed data regions are reorganized and replicated. The oldest and less frequently accessed replicas are replaced. This entire process yields a shorter initial "training time", while maintaining good read performance for the true frequent patterns in the long run.

B. VPIC: Plasma Physics Particle Data

The VPIC dataset is generated from the first principles 3D electromagnetic relativistic kinetic particle-in-cell code [3], to simulate collisionless magnetic reconnection phenomenon of two trillion particles. It contains properties of particles that include the location (X, Y, Z), kinetic energy ($Energy$), and individual components of particle velocity (U_x, U_y, and U_z). We used a subset of the trillion particle dataset based on the condition of $Energy > 1.1$. Four variables are retrieved: "Energy," "X," "Y," and "Z," using 4096 MPI processes. Each variable contains about 188 billion elements with a size of 700 GB and each stored as a $1D$-array.

In our experiments, we use the queries similar to those of a previous analysis of this data [3] for visualizing the highly energetic particles. The corresponding queries are selecting variables with value constraints such as $Energy > 1.7$ and $Energy > 1.6$. We split the experiments by running queries on the plasma physics data in two groups: A and B, as shown in Table II. Group A ($A1$ to $A4$) represents the best cases where the created replica exactly matches the request (sufficient storage space). While for group B ($B1$ to $B3$), only a partially matched replica exists, which simulates when storage space can only afford replicas that cover part of the requests. In such cases, our framework automatically selects the replica with the largest overlapping region during runtime.

Table II compares the read time of accessing the original dataset and the read time of accessing the replicas that use "concatenation". The framework overhead is included in the "Optimized time", since the time to load the metadata

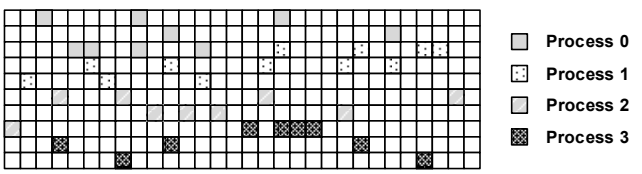

Fig. 6. An example of the typical accesses for the plasma physics dataset: each box corresponds to one element, each process reads a number of elements with their locations scattered throughout the dataset.

TABLE II
DATA SELECTIONS USED IN THE EXPERIMENTS FOR THE PLASMA PHYSICS DATASET.

Test case	High-level representation	Total data accessed	Replica overlap percentage	Original time (s)	Optimized time (s)	Overhead time (s)	Speedup
A1	Energy >1.7	588 MB	100%	15.9	1.5	0.7	10.6
A2	Energy >1.6	1151 MB	100%	27.4	2.8	1.9	9.8
A3	Energy >1.5	2378 MB	100%	115.2	5.1	3.1	28.1
A4	Energy >1.4	5441 MB	100%	677.7	7.5	4.2	90.3
B1	Energy >1.4	5441 MB	44%	677.7	387.5	5.7	1.7
B2	Energy >1.5	2378 MB	48%	115.2	64.6	3.6	1.8
B3	Energy >1.5	2378 MB	25%	115.2	89.7	2.0	1.3

(mapping from reorganized layout to original layout) is non-negligible. We can see a substantial performance improvement when accessing the replicas for various queries. The speedup increases with the total data accessed in Group A, with the highest speedup being $90X$ in $A4$. The reason for this speedup is that with more elements selected, more seek and read operations are involved, causing contention in the parallel file system and resulting in low I/O throughput (less than 10MB/s). In contrast, the reorganized replicas are read in large contiguous chunks, resulting in faster performance. For Group B, the speedup increases with the "overlap percentage", as more data is read contiguously with high throughput.

C. Atmospheric Rivers Detection

Extreme precipitation events on the western coast of North America are often traced to an extreme weather phenomenon, known as atmospheric rivers (ARs). We focus only on the I/O phase of the detection process [4], which is as follows: each MPI process reads the integrated water vapor (IWV) variable pertaining to one time step. Each time step typically represents the daily average of IWV. For each MPI process, a typical data read pattern consists of obtaining data related to an ocean basin. For instance, with the climate dataset for the entire globe, studying the AR events in the US western coast requires reading data corresponding to a rectangular region in a mesh (with a bounding box selection of the $2D$ dataset), as shown in Figure 7.

In this experiment, we analyze the performance of I/O kernel of the AR detection on two datasets. The two datasets $D1$ and $D2$ each have 4096 time steps and contain 1536×2304 double values per time step. This resolution represents a $0.25°$ climate model or observation output. With the same interested region, the data selections are identical. We used 4096 MPI processes and 4 different read requests, $Q1$ through $Q4$, that read 2D sub-planes with 5%, 10%, 15%, 20% of the total dataset size. In the first half of runs with $Q1$ to $Q4$ and dataset $D1$, our framework has no prior knowledge and no replica is available; it only reads from original datasets. The differences in time between the first 4 bars in Figure 8 are the overhead of our framework. The overhead is negligible, in the range of $4-5\%$, shown as labels above the bars. After each of the first 4 queries, with sufficient storage space, our framework automatically creates replicas with reorganized layouts of the previously accessed data. The user can also choose to apply such optimizations to another dataset. In our test, we chose to apply on a dataset $D2$, and the performance for accessing with $D2$ with reorganized replicas are shown as the last 4 bars of Figure 8. From the second half of the results, we can see that even with this small-sized dataset, our framework is still able provide speedups of $1.5X$ to $1.7X$ with concatenation, and $1.1X$ to $1.2X$ with Z-curve.

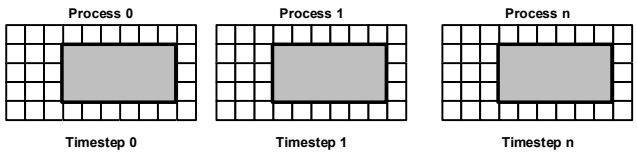

Fig. 7. An example of AR detection code's read accesses in a file. A subset of the rectangular region would result in multiple same-sized blocks with gaps in between when data is stored in row-major layout.

D. Mass Spectrometry Imaging

Mass spectrometry imaging (MSI) enables researchers to directly probe endogenous molecules within the architecture of the biological matrix. The data for each position is represented as a profile of intensity over a corresponding range of mass-to-charge (m/z) values. There are three types of frequently used patterns when accessing an MSI dataset [19]: (1) m/z slice selection ([:, :, z_{min} :z_{max}]); (2) spectra selection ([x_{min} :x_{max}, y_{min} :y_{max}, :]); and (3) 3D sub-volume selection ([x_{min} :x_{max}, y_{min} :y_{max}, z_{min} :z_{max}]).

MSI dataset can be described as a three-dimensional cube

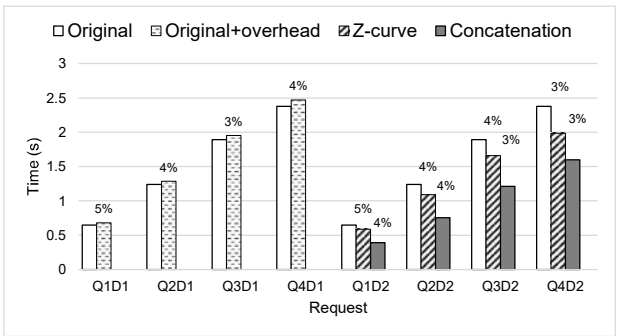

Fig. 8. $Q1D1$ to $Q4D1$ compares the time of reading from the original dataset $D1$ with ("Original+overhead") and without ("Original") our framework. $Q1D2$ to $Q4D2$ are combined results of two sceneries: (1) no additional storage allowed and our framework selects a "Z-curve" layout and replaces the original dataset with it, (2) sufficient additional space and our framework selects "Concatenation" and creates corresponding partial replicas. The percentage number labels are the framework's runtime overhead.

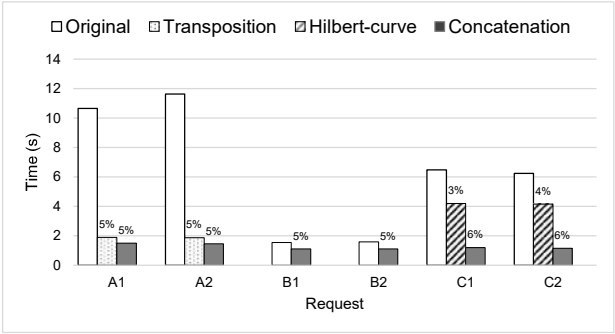

Fig. 9. Group A compares the time to read from the original dataset, a reorganized layout with "Transposition", a reorganized layout with "Concatenation", where 25 consecutive m/z slices per process are read; Group B compares the time to read from original dataset with the time to read from partial replicas reorganized with "Concatenation", where each process reads 3×3 spectra; Group C compares the time to read from original layout, a reorganized layout with "Hilbert-curve", and a reorganized layout with "Concatenation", where each process reading $20 \times 20 \times 10000$ sub-volumes. The percentage number labels are the framework's runtime overhead.

of $(x, y, m/z)$. Each spectrum describes the distribution of masses (m/z) at a given image location (x, y). For the experiments, we used a dataset with size $394 \times 518 \times 133092$. With this relatively small dataset, in order to avoid noise in I/O operations, only 144 MPI processes were used so that each process reads a non-trivial amount of data. We simulate the process of exploring the MSI dataset with 6 executions, including each of the above mentioned common patterns. For each type of pattern (referred as "Group"), we vary the location of the selected region and the results are shown in Figure 9.

For Group A and C, the read accesses include much more non-contiguous chunks when reading from the original row-major layout as the data selection is not aligned with the data linearizion. A new layout that uses transposition or a Hilbert-curve better matches the read pattern and thus offers more contiguous accesses and speedups ranging from $1.5X$ to $8X$. The concatenation further improves the read performance as a result of OST-aware replica placement, which is not supported for transposition (Transposition and Hilbert-curves are applied to the entire dataset, and can only use the parallel file system's striping parameters for data placement). For Group B, the read accesses match the row-major layout and are near-optimized, leaving the only option for performance improvement on data placement, which brings close to a $1.4X$ speedup. Each group has two tests with same data selection but at different locations (e.g. A1 and A2 are different only with the start location).

E. Electrocorticography (ECoG) Data

An ECoG experiment [1] records the electrical activity from the cerebral cortex when the patient is reading different words at different times. We used a $1GB$ dataset containing a $2D$-array of 541241344×256 64-bit double-precision floating point numbers. Associated with the data are metadata such as of indices and labels to locate the target data regions. The pattern of the ECoG benchmark is non-regular (strided with variant strips) with queries from users such as "select all data labeled with $KEE1$." Due to the small size, we used 24 processes from 6 compute nodes for the parallel reads.

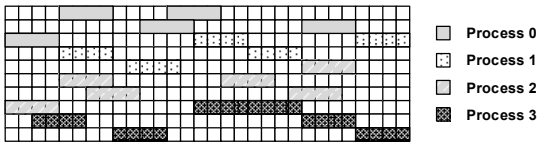

Fig. 10. ECoG dataset: data having the same label are stored in a row-major file with same-size blocks but variant-sized gaps in between.

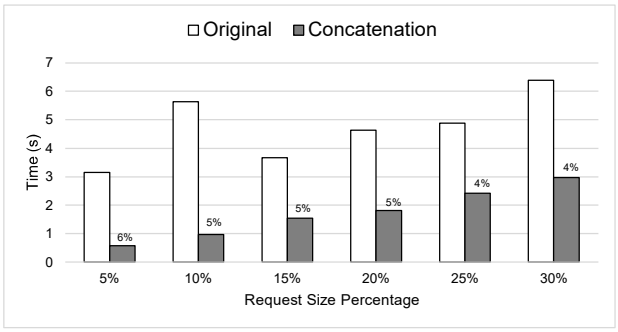

Fig. 11. Read time comparison with different query selections accessing the ECoG dataset. With non-regular data selection, our framework selects "Concatenation" to reorganize and create partial replicas. The percentage number labels are the framework's runtime overhead.

Note that data labeled with a certain type such as "KEE1" are scattered blocks stored in the file, as it is recorded at different times (e.g. Figure 10); their location is determined by the metadata "Event_EIndx" associated with the dataset.

In our experiment, we simulate the data read behavior of the ECoG dataset by selecting and reading data of multiple labels. The 6 different experiments shown in Figure 11 are selected with different sets of query labels, while varying the total request sizes ranging from 5% to 30% of the total data size. Our framework selects the "concatenation strategy". As the data associated with the different labels has different distributions in the file, and has different impact to the overall performance, it is expected that the time to read from the original layout does not grow linearly with the read size. While using concatenated dataset for reorganization, it is guaranteed that each of the requests is reading a contiguous chunk, thus the time grows close to linearly. The speedups of using our framework range from $2X$ to $6X$, as concatenation makes the previously non-contiguous data accessed from the original layout one contiguous chunk in the replica.

F. Adaptive Mesh Refinement (AMR) Data

To evaluate the benefit of our framework in accessing adaptive mesh refinement (AMR) data in addition to the uniform grid data, we have developed an AMR file read benchmark that allows user to specify a multi-dimensional region, and read the corresponding data of all levels. We used a $1TB$ dataset with HDF format. It has 3 levels and a refinement ratio of 4 generated with Chombo, which is a popular block-structured AMR infrastructure used by many applications [18], [7]. The experiments are run with 4096 MPI processes, and the selected data is distributed evenly.

Figure 13 shows the performance improvements brought by the optimizations of our framework. As discussed in

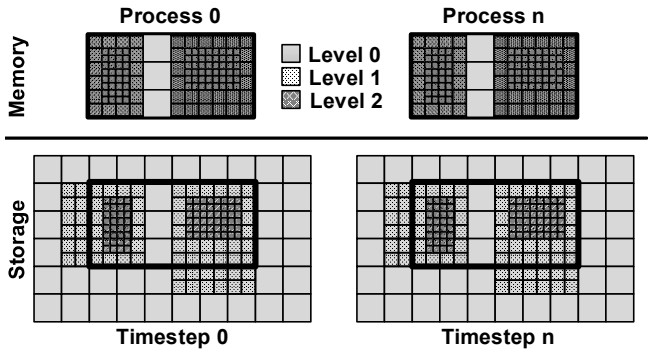

Fig. 12. An example of accesses of the AMR read benchmark: each process reads a rectangular subset of data on all three levels from one timestep.

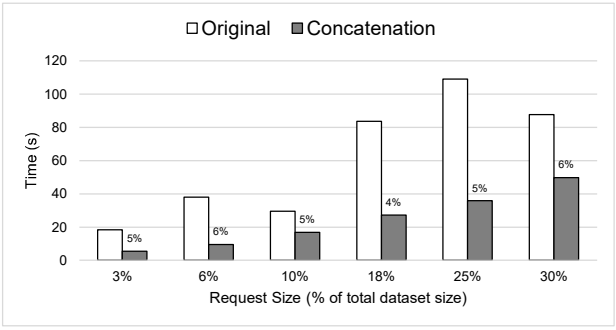

Fig. 13. Read time comparison with different queries accessing the AMR dataset. Our framework selects "Concatenation" to reorganize and create partial replicas due to the non-regular data selection. The percentage number labels are the framework's runtime overhead.

the previous ECoG experiment, the different distribution of accessed data in the file causes the non-linear growth of time with increasing read sizes. Concatenation is the reorganization strategy that our framework select, and the requested data from each layer in the AMR data are concatenated to provide more contiguous accesses. We have observed $1.8X$ to $4X$ speedups in the test cases with varying sizes of data.

G. Validation of the Cost Model

To Validate the correctness and effectiveness of the runtime decision making process (Section III-A), we have designed a series of queries with different patterns. We prepared three synthetic datasets with one, two, and three dimensional structure. For the $1D$ dataset, only concatenated datasets are supported, while transposition, and space-filling curves are supported for $2D$ and $3D$ datasets. The data selection type of the patterns (all are non-contiguous accesses) that we use are subset, sub-plane, sub-volume, and element selection.

For layout decision making, the model only needs to provide information to choose between candidates. As experiments are performed in a shared environment, the I/O performance can be affected by many different factors, such as other users accessing the parallel file system and the network. Thus, instead of comparing the absolute performance, we only compare the ranks of different layouts computed by the cost model; this is sufficient for runtime uses.

We varied the data selection types and layout candidates as shown in Table III. $SEL0$ selects a subset of the $1D$ array; $SEL1$ selects a subset along y dimension; $SEL2$ and $SEL3$ select a $2D$ sub-plane; $SEL4$ selects a $2D$ sub-plane with a significantly larger size along y dimension; $SEL5$ selects a $3D$ sub-volume with a significantly larger size along y dimension; $SEL6$ and $SEL7$ select a $3D$ sub-volume with comparable dimension size of all dimensions. The concatenated replicas always yields the best performance as they provide the most contiguous read accesses. To simulate the cases that no extra storage is allowed, we intentionally include it only once in the candidate layout. We can see that the proper decision is made among multiple layouts with different kinds of patterns.

H. Overhead

The storage overhead for bounding box selections is about the same size as the created partial replicas (the associated metadata are in a compact representation of near-constant size). While for element selection, the total storage overhead is about 1.3 times replica's size (assuming 32-bit data elements, and 16-bit integers together with compression used for metadata). Since space-filling curves approaches operate directly on original data, only metadata is needed with near-constant size.

The runtime overhead comes from the three parts in our framework. We labeled the overhead numbers above each of the bars in all the plots. From our measurement, the total overhead of trace analysis and I/O redirection is relatively small, less than 3% in all test cases. While depending on the data selection type as well as the number of available replicas, the overhead for metadata loading and decision making varied by a great deal. For bounding box selection, it ranges from 3% to 6%, while for element selection, due to the read for the much larger metadata information, it may take 10% or more depending on the number of candidate replicas after the first round of pruning as mentioned in the first Step of Section III-B. Note that even after considering the overheads of metadata loading, we observe substantial overall performance improvements as the resulting read from replica is much more contiguous than from original dataset.

V. Conclusion

We proposed a framework that selects the most suitable layout among the common layout reorganization techniques based on detected data usage patterns. It is capable of performing storage-efficient optimizations for heterogeneous patterns from both bounding box and element data selections. On datasets of scientific applications from various domains, our framework yields $1.3X$ to $90X$ time speedup in the plasma physics queries, $1.1X$ to $1.7X$ speedup in the AR detection, $1.4X$ to $8X$ in MSI, $2X$ to $6X$ in ECoG, and $1.8X$ to $4X$ in AMR experiments.

Apart from the optimization techniques supported by our framework, we are exploring other aspects to further improve read performance. Compression is a typical process that results in reduced size of data accesses, and is a topic of our future work. We are currently working on integrating this framework into the SDS framework for broader applicability of dynamic data reorganization.

TABLE III
COST MODEL VALIDATION

Selection	SEL0	SEL1	SEL2	SEL3	SEL4	SEL5	SEL6	SEL7
Dimension	1	2	2	2	3	3	3	3
Candidate Layout	R0, C0	R1, T1	R2, T2, H2, Z2	R3, T3, H3, C3	R4, T4x, T4y, T4z	R5, T5y, T5z, H5	R6, T6y, T6z, H6	R7, T7y, T7z, H7, C7
Selected	C0	S1	Z2	C3	T4y	T5z	H6	C7
Measured Best	C0	S1	Z2	C3	T4y	T5z	H6	C7

R: Original row-major. Tu: Transposition with u as first dimension. Z: Z-curve H: Hilbert-curve. C: Concatenation.

Acknowledgements.: This work is supported in part by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under contracts DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory and DE-AC05-00OR22725 at Oak Ridge National Laboratory, and by the U.S. National Science Foundation (Expeditions in Computing and EAGER program). This research used resources from the National Energy Research Scientific Computing Center and Oak Ridge Leadership Computing Facility.

REFERENCES

[1] K. E. Bouchard and E. F. Chang. Control of spoken vowel acoustics and the influence of phonetic context in human speech sensorimotor cortex. *The Journal of Neuroscience*, 34(38):12662–12677, 2014.

[2] P. G. Brown. Overview of SciDB: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 963–968. ACM, 2010.

[3] S. Byna, J. Chou, O. Rübel, H. Karimabadi, et al. Parallel I/O, analysis, and visualization of a trillion particle simulation. In *SC*, page 59. IEEE Computer Society Press, 2012.

[4] S. Byna, M. F. Wehner, K. J. Wu, et al. Detecting atmospheric rivers in large climate datasets. In *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities*, pages 7–14. ACM, 2011.

[5] J. H. Chen, A. Choudhary, B. De Supinski, M. DeVries, E. Hawkes, Klasky, et al. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science & Discovery*, 2009.

[6] P. Colella, D. Graves, T. Ligocki, D. Martin, D. Modiano, D. Serafini, and B. Van Straalen. Chombo software package for AMR applications-design document, 2000.

[7] S. L. Cornford, D. F. Martin, D. T. Graves, D. F. Ranken, et al. Adaptive mesh, finite volume modeling of marine ice sheets. *Journal of Computational Physics*, 232(1):529–549, 2013.

[8] B. Dong, S. Byna, and K. Wu. SDS: a framework for scientific data services. In *Proceedings of the 8th Parallel Data Storage Workshop*, pages 27–32. ACM, 2013.

[9] B. Dong, S. Byna, and K. Wu. Spatially clustered join on heterogeneous scientific data sets. In *2015 IEEE International Conference on Big Data*, pages 371–380, Oct 2015.

[10] B. Dong, X. Li, L. Xiao, and L. Ruan. Towards minimizing disk I/O contention: A partitioned file assignment approach. *Future Generation Computer Systems*, 37:178 – 190, 2014. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing.

[11] Z. Gong, T. Rogers, J. Jenkins, H. Kolla, S. Ethier, J. Chen, R. Ross, S. Klasky, and N. F. Samatova. MLOC: Multi-level layout optimization framework for compressed scientific data exploration with heterogeneous access patterns. In *ICPP*, pages 239–248. IEEE, 2012.

[12] M. Howison. Tuning HDF5 for lustre file systems. In *Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10),, Heraklion, Crete, Greece, September 24, 2010*, 2012.

[13] J. Jenkins, X. Zou, H. Tang, D. Kimpe, R. Ross, and N. F. Samatova. Radar: Runtime asymmetric data-access driven scientific data replication. In *Supercomputing*, pages 296–313. Springer, 2014.

[14] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, et al. ISABELA-QA: query-driven analytics with ISABELA-compressed extreme-scale scientific data. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 31. ACM, 2011.

[15] J. Liu, S. Byna, B. Dong, K. Wu, and Y. Chen. Model-driven data layout selection for improving read performance. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 1708–1716. IEEE, 2014.

[16] 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 *HPDC*, pages 49–60. ACM, 2011.

[17] B. Nam and A. Sussman. Improving access to multi-dimensional self-describing scientific datasets. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 172–179. IEEE, 2003.

[18] K. S. Perumalla, R. M. Fujimoto, P. J. Thakare, et al. Performance prediction of large-scale parallel discrete event models of physical systems. In *Simulation Conference, 2005 Proceedings of the Winter*, pages 9–pp. IEEE, 2005.

[19] O. Rübel, A. Greiner, S. Cholia, K. Louie, E. W. Bethel, T. R. Northen, and B. P. Bowen. OpenMSI: A high-performance web-based platform for mass spectrometry imaging. *Analytical chemistry*, 85(21), 2013.

[20] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Data Engineering, 1994. Proceedings. 10th International Conference*, pages 328–336. IEEE, 1994.

[21] H. Tang, X. Zou, J. Jenkins, D. A. Boyuka II, S. Ranshous, D. Kimpe, S. Klasky, and N. F. Samatova. Improving read performance with online access pattern analysis and prefetching. In *Euro-Par*, pages 246–257. Springer, 2014.

[22] The HDF Group. Hierarchical Data Format, version 5, 1997-2015. http://www.hdfgroup.org/HDF5/.

[23] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, R. Grout, N. Podhorszki, Q. Liu, Y. Wang, and W. Yu. EDO: improving read performance for scientific applications through elastic data organization. In *CLUSTER*, pages 93–102. IEEE, 2011.

[24] W. Wang, Z. Lin, W. Tang, W. Lee, S. Ethier, J. Lewandowski, G. Rewoldt, T. Hahm, and J. Manickam. Gyro-kinetic simulation of global turbulent transport properties in tokamak experiments. *Physics of Plasmas*, 13:092505, 2006.

[25] K. Wu. FastBit: an efficient indexing technology for accelerating data-intensive science. In *Journal of Physics*, volume 16, page 556. IOP Publishing, 2005.

[26] K. Wu, S. Byna, D. Rotem, and A. Shoshani. Scientific data services: a high-performance I/O system with array semantics. In *Proceedings of the first annual workshop on High performance computing meets databases*, pages 9–12. ACM, 2011.

[27] 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, 2013.