MLOC: *M*ulti-level *L*ayout *O*ptimization Framework for *C*ompressed Scientific Data Exploration with Heterogeneous Access Patterns

Zhenhuan Gong[1,2], Terry Rogers[1,2], John Jenkins[1,2], Hemanth Kolla[3], Stephane Ethier[4], Jackie Chen[3], Robert Ross[5], Scott Klasky[2], Nagiza F. Samatova[1,2,*]

[1]North Carolina State University, NC 27695, USA
[2]Oak Ridge National Laboratory, TN 37831, USA
[3]Sandia National Laboratory, Livermore, CA 94551, USA
[4]Princeton Plasma Physics Laboratory, Princeton, NJ 08543, USA
[5]Argonne National Laboratory, Argonne, IL 60439, USA
[*]Corresponding author: samatova@csc.ncsu.edu

***Abstract*—The size and scope of cutting-edge scientific simulations are growing much faster than the I/O and storage capabilities of their runtime environments. The growing gap gets exacerbated by exploratory data–intensive analytics, such as querying simulation data for regions of interest with multivariate, spatio-temporal constraints. Query-driven data exploration induces *heterogeneous access patterns* that further stress the performance of the underlying storage system. To partially alleviate the problem, data reduction via compression and multi–resolution data extraction are becoming an integral part of I/O systems. While addressing the data size issue, these techniques introduce yet another mix of access patterns to a heterogeneous set of possibilities. Moreover, how extreme-scale datasets are partitioned into multiple files and organized on a parallel file systems augments to an already combinatorial space of possible access patterns.**

To address this challenge, we present MLOC, a parallel *M*ulti-level *L*ayout *O*ptimization framework for *C*ompressed scientific spatio-temporal data at extreme scale. MLOC proposes multiple fine-grained data layout optimization *kernels* that form a generic *core* from which a broader constellation of such kernels can be organically consolidated to enable an effective data exploration with various combinations of access patterns. Specifically, the kernels are optimized for access patterns induced by (a) query–driven multivariate, spatio-temporal constraints, (b) precision–driven data analytics, (c) compression–driven data reduction, (d) multi–resolution data sampling, and (e) multi–file data partitioning and organization on a parallel file system. MLOC organizes these optimization kernels within a multi–level architecture, on which all the levels can be flexibly re-ordered by user–defined priorities. When tested on query–driven exploration of compressed data, MLOC demonstrates a superior performance compared to any state-of-the-art scientific database management technologies.

I. Introduction

Recent scientific computing projects, such as GTS core plasma fusion [1] and S3D combustion simulations [2], require *global–context*, *space+time*, *multi–variate* analysis of extreme-scale datasets as an integral part of their scientific discovery cycle. Their typical analytical workflows consist of iterative data querying for patterns of interest and fetching subsets of data with *heterogeneous access patterns* on parallel storage systems, for which current data management solutions are hardly optimized.

Storage and I/O optimizations for scientific data in HPC have been extensively studied at all levels of the software stack–from parallel file systems (PFS) (e.g., GPFS [3], Lustre [4], PVFS [5]) to I/O middleware (e.g., ADIOS [6], HDF5 [7], Parallel netCDF [8]), and data staging architecture (e.g., DataStager [9], PreDatA [10]). However, such optimizations have been primarily driven by needs for fast data offloads from the simulation runtime environment through maximizing the throughput of data *writes*. Storage access patterns induced by data writes in simulations are not only known as *a priori* but also *fixed*, thus allowing for individual pattern-driven performance optimizations.

In contrast, exploratory analysis of the simulation data so-produced does not enjoy such a luxury: it induces access patterns that are highly *heterogeneous*, *arbitrary*, and *hard–to–predict*. Scientific simulation codes generate multi-dimensional, multi-variate, time-series data of floating point values (usually double-precision) over different meshes and spatial grids and store them on PFS. The multitude of possible access patterns during data *reads* is inherently combinatorial, making *optimal linearization* of the simulation data and subsequent multi–file partitioning of this linearized representation and PFS layout an extremely challenging task. With the emerging trends of utilizing leadership-class computing facilities (LCF) not only for running simulations but also for performing visual exploratory analyses of their products, it is becoming imperative to optimize the I/O costs over the entire lifetime of the scientific discovery cycle and to accept potential extra up-front costs to mitigate performance degradations under different access patterns.

Moreover, the underlying data model of scientific data significantly differs from traditional relational data and storage models that typically utilize either row–store or column–store linearization of relational data tables. For simulation-driven data models, existing storage layout techniques have primarily focused on optimizations for a *particular* access pattern(s). For example, space-filling curves (SFC) [11] linearize multi-dimensional data for high spatial locality [12] [13]. While successful, they only improve performance for access patterns

induced by spatial constraints on sub-planes/sub-volumes of the data space. For value-constrained access patterns, the entire dataset must be scanned to select qualified points. FastBit [14] builds up binned bitmap indices of the data based on their values, keeping similarly valued data in the same bin to speed up value-constrained region queries. ISABELA-QA [15] enables efficient data access based on value constraints by performing value-based *binning* on ISABELA-compressed data [16]. However, both techniques are optimized only for value-constrained accesses and cannot handle space-constrained accesses efficiently. A naïve approach to support heterogeneous access patterns would be creating multiple copies of one dataset that favor different data layouts on storage optimized for different permutations of possible access patterns. However, the ever-increasing sizes of simulation data make multiple replications infeasible at extreme scale.

In this paper, we present MLOC (*M*ulti-level *L*ayout *O*ptimization of *C*ompressed Scientific Data) which optimizes the storage layout of scientific data for effective data exploration with *heterogeneous access patterns* in PFS environments. In its core are multiple fine-grained data layout optimization *kernels* that can be organically consolidated to support various combinations of access patterns. The kernels are optimized for patterns induced by (a) query–driven multivariate, spatio-temporal constraints, (b) precision–driven data analytics, (c) compression–driven data reduction, (d) multi–resolution data sampling, and (e) multi–file data partitioning and organization on a parallel file system. MLOC organizes these optimization kernels within a multi–level architecture, on which all the levels can be flexibly re-ordered by user–defined priorities. The paper makes the following contributions:

- It presents MLOC, a multi-level data layout optimization framework on top of PFS for HPC environments. By incorporating fine-grained data layout optimizations tuned for PFS in a flexible multi-level manner, it achieves optimization for multiple heterogeneous access patterns.
- It presents an efficient byte-level solution for precision-driven data access support. The solution achieves higher detail preservation than what is possible for traditional multi-resolution data sampling.
- MLOC offers a first-class treatment of data compression through support of different formats of compressed data, and optimizes data layout for each of them to reduce storage and I/O overhead and improve access performance.
- MLOC implements a data processing pipeline which is readily incorporated with existing data staging frameworks [9] [10] to achieve efficient in-situ data layout optimization and compression.
- Experiments show that compared to the state-of-the-art techniques like FastBit, our approach achieves both lower query delays and storage overhead.

II. Problem Statement

MLOC is aimed toward several important classes of *query-driven* and *compression-driven* access patterns for scientific multi-variate spatio-temporal datasets. The patterns include:

- *Value-constrained region-only data access* (generated by range or region queries) that requests spatial regions and/or corresponding variable values, subject to their or others' values constraints (*VC*). E.g., what (latitude, longitude) pairs at some time have an abnormally high temperature? What are those temperature values?
- *Spatial/Region-constrained value-retrieval data access* (generated by value queries) that requests spatial regions and/or corresponding variable values, subject to spatial/regional constraints (*SC*). E.g., what are the humidity values within New York at some time?
- *Value-and-spatial-constrained data access* that requests spatial regions and/or corresponding variable values, subjects to constraints on both the regions as well as variable values. E.g., what are the regions within New York with an abnormally high temperature?
- *Multi-variable data access* that applies to all of the above but may involve two or more variables. E.g., what are the temperature values within New York at some time, where the humidity is above 90%?
- *Multi-resolution data access* that applies to all of the above with multi-resolution support. If lower resolution of data is required, only a subset of the accessed data, or less-precise values of the data are fetched so that less data is transferred and processed to reduce I/O and computation overhead. More precise and complete data can be fetched if higher resolution is required.

MLOC also supports data compression for double-precision scientific data with different compression techniques. The compression can be lossy or lossless based on user requirements of precision, compression ratio, and throughput.

While current techniques optimize data storage layout for some *individual* access patterns, to the best of our knowledge no existing system meets *all* these requirements. It is challenging to optimize for *all* the accesses within a single linearization scheme, without additional data replication to save storage space. The major technical difficulties include:

1) Resolving Layout Optimization Conflicts: To reduce search space and response time for data access, layout optimization for one access pattern attempts to store data together to reduce seek overhead for that access pattern. For other patterns, the data that needs to be accessed may be scattered over different areas, increasing seek time. Thus, the layout optimized for one access pattern will almost always conflict the desired layout of other patterns and affect their performance.

2) Bounding Storage and I/O for Data and Indices: To speed up various access patterns, existing technologies [14] build multiple *heavy-weight* indices, often comparable with or larger than the size of the data alone. However, the fast growing size and complexity of scientific datasets make it infeasible to hold large indices in storage and transfer them via I/O efficiently. Thus, the solution must adopt an efficient and *light-weight* indexing scheme that can serve for different accesses without introducing too much additional storage.

3) Optimizing for Parallel File Systems: Previous layout optimizations usually dealt with single-disk environments.

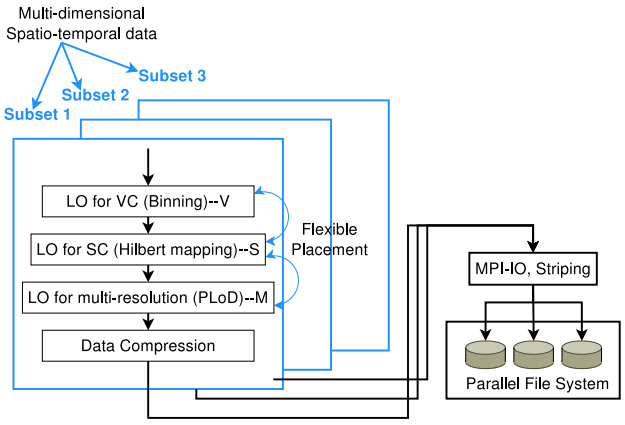


Fig. 1. An overview of MLOC's multi-level architecture. The positions of levels for layout optimization (LO) are flexible.

However, scientific datasets are processed and stored on PFS in HPC environments, which differs greatly from single-disk. The solution should take PFS-specific mechanisms into consideration and provide optimization for the PFS in HPC environments to achieve best parallel access performance.

III. Method

A. Approach Overview

1) Multi-level Layout Optimization: MLOC applies a flexible multi-level architecture, as shown in Figure 1. For each of the data processing requirements listed in Section II, MLOC applies one level of optimization with a specific technique. There are three levels of layout optimizations targeting access patterns induced by (a) queries with value constraints (*VC*), (b) queries with spatial constraints (*SC*), and (c) precision-based multi-resolution data reduction, as well as one level for a data compression-induced access. As Figure 1 shows, the input data are first divided into multiple subsets based on the hierarchical Hilbert curve mapping to separate data to support subset-based multi-resolution data access. For each subset, data are processed through a pipeline consisting of four layers, and each layer serves as a filter to optimize data layout or perform data compression. The following introduces how the MLOC architecture is designed and implemented to address the challenges listed in Section II.

2) Flexible Placement of Different Levels: Layout optimizations are performed in a certain order, as shown in Figure 1, and the optimization order may affect the performance for other access patterns. However, we observe that, for queries on certain scientific datasets, there exists a priority order of different queries based on the frequency they are executed. For example, for climate datasets, scientists may be mostly interested in queries of temperature values within a certain spatial region (queries with *SC*), while for fusion simulation datasets scientists may mainly be interested in queries of regions with temperature values higher than some threshold (queries with *VC*). The priority of different queries invokes the order of data layout optimizations for accesses.

Based on the above, MLOC allows each level to be placed in a hierarchical order and switched based on the priorities of optimizations. If queries with *VC* are the most frequently executed, then MLOC can be configured to emphasize optimization for *VC* by placing it at the top-most level and the remaining patterns at lower levels. It will also optimize for *SC* and multi-resolution accesses, though at a lower priority.

3) Light-weight and Efficient Indexing: To stay within the allowable storage and I/O bounds, MLOC applies a light-weight and efficient indexing scheme. The multi-dimensional data is first chunked and then distributed into different bins. The indices are built during binning to record the original spatial position of the points so that region queries with *VC* can be answered without the need to access the data itself. To optimize for value queries with *SC*, MLOC applies Hilbert Space-filling Curve (*HSFC*) [17] mappings to reorganize data chunks, improving spatial locality without introducing additional space overhead of indices for multi-dimensional data.

4) Parallel File System Optimization: MLOC divides data files to achieve the optimal file organization for PFS, taking into account partitioning, striping, and so on, to achieve optimal parallelism in data access. Details of the design and optimization are described in Section III-C.

B. Optimization Techniques in Each Level

1) Value-based Binning for Value-constrained Queries: MLOC bins the dataset by placing elements in each block into bins based on their values so that points with similar values are placed together on the storage. For queries with *VC*, MLOC only needs to access a portion of the bins according to the *VC*. If the bin bounds are contained within the bounds of the *VC*, further filtering is not required. For region-only queries, MLOC only needs to return the indices as output without accessing and recovering the values. Such bins are defined as *aligned bins*. Only bins whose bounds are not contained within the query bound need their contents analyzed. One major attribute that is critical to the performance of MLOC is the number of bins that must be accessed to answer each query. Moreover, the bin bounds should be selected carefully to form as many *aligned bins* as possible for queries and to achieve balanced access performance among different bins. MLOC applies equal frequency binning to prevent load imbalance.

2) Hilbert Curve Mapping for Spatial-constrained Queries: Since multi-dimensional arrays are linearized in one-dimensional (1-D) storage space, the performance to access values in different dimensions may vary greatly. To overcome this problem, MLOC divides multi-dimensional arrays into chunks and organizes the chunks in Hilbert Space-filling Curve (*HSFC*) order on disks to optimize for queries with *SC*. Space-filling curves help improve spatial locality of multi-dimensional data when they are linearized on the 1-D storage space. Thus, potential seek operations for access patterns on certain spatial regions are reduced, since one pair of seek and read operations should be able to load as many contiguous blocks as possible. It is worth noting that since the order of blocks is based on the *HSFC* itself and, contrary to other methods, every grid point is representative of a block, no additional metadata must be stored to track this order beyond

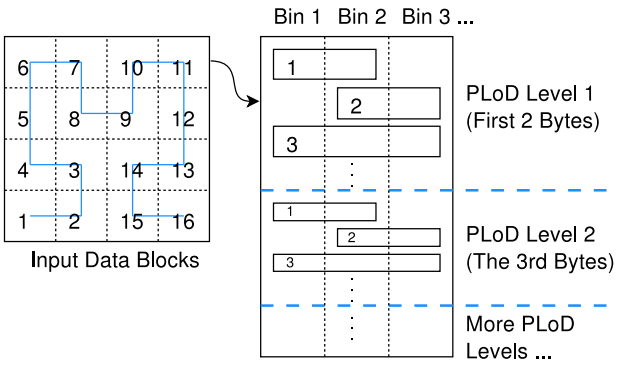


Fig. 2. A mapping from the input data, split into blocks, into bins based on variable ranges. Each bin shows, top-to-bottom, the (relative) order of the first few blocks in file, using Hilbert curve mapping.

the dimensions of the dataset in blocks. Figure 2 includes an example of organizing 2-D 4×4 grids in *HSFC* order. *HSFC* has been shown to have strong geometric locality properties [17] compared to other SFC, motivating its usage in the MLOC design.

3) Multi-resolution Support: MLOC supports two multi-resolution data organization approaches: the traditional *subset-based* approach and a new *precision-based* approach. The subset-based approach supports multi-resolution access by accessing partial points in datasets for certain resolution requirements. It applies a hierarchical Hilbert space-filling curve mapping to store data in the same resolution level together to speed up multi-resolution data access, which is similar to [13].

However, the subset-based approach misses a large number of points in lower-resolution accesses and is thus only suitable for visualization or analytics with low-precision requirements. To make up for its drawbacks, MLOC provides precision-based multi-resolution support by exploiting the memory layout of double-precision data, dividing a double-precision number into seven parts, as shown in Figure 3. The first part contains the first two bytes, which are the least required bytes to represent a double-precision number (one byte will not contain any bits from the fraction and only a partial exponent value). Each of the other six parts contains one byte for additional precision. The seven subsets of the eight bytes can represent the original value with certain precisions, which is called Precision-based Level of Details (*PLoD*). Each level of details contains all data points, but only partial bytes of the double-precision numbers (except for level 7, which contains all bytes and is regarded as full-precision) to save I/O cost. For example, while using *PLoD* level 2, only the first three bytes of all points are fetched from storage to assemble the original values. Experiments show that using *PLoD* level 2 (three bytes) only introduces a maximum per-point relative error of 0.008% for S3D dataset in mean value analysis, which is already enough for many statistic and data mining functions. Meanwhile, it reduces the I/O cost by 62.5% since only 3 bytes out of each 8-byte double-precision value are fetched. All the double-precision numbers are divided into 8 bytes, and bytes at the same position (which means the same *PLoDs*) are

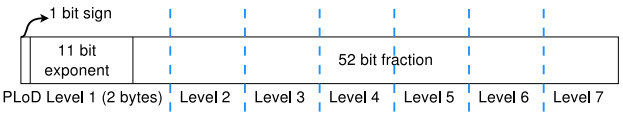


Fig. 3. Memory layout of double-precision floating point numbers and the division of *PLoD* levels for precision based multi-resolution access support.

stored contiguously. Users can choose to fetch partial bytes to assemble the original values with certain error rates, or all bytes to recover original values with full precision.

4) Data Compression: MLOC provides architectural support for various lossless and lossy data compression techniques of scientific data. Any compression technique, such as the standard *Zlib* compression, can be plugged into the framework. If *PLoDs* are used and different bytes are stored and compressed separately, standard compression techniques for ordinary buffers should be used. Otherwise, compression specifically designed for floating point numbers, such as FPZip [18], ISOBAR [19], or *B*-spline interpolation-based ISABELA [16], can be used. MLOC supports flexible block and binning size adjustment for different compression techniques to achieve best performance in the desired area, such as compression ratio and throughput.

5) Combination of Optimizations: Input datasets go through a multi-level pipeline built by MLOC to achieve multiple optimizations. As an example, assume that MLOC applies optimizations in the following order: accesses with *VC*, *M*ulti-resolution access, accesses with *SC* and compression (regarded as *V-M-S* order). With this permutation, MLOC will optimize the data layout as shown in Figure 2. In Figure 2, the entire dataset is first divided into bins. In each bin, the data is stored by byte groups, in which the first two bytes of all values are stored together, then the third byte, etc., which means optimization for *PLoD* is second. Finally, within each byte group, data chunks are organized in *HSFC* order to optimize for data accesses with *SC*. To achieve this layout, or others, MLOC divides the entire dataset into the smallest units (which is certain bytes of values inside a block within a bin) and organizes these by the order of priority to achieve the desired layout through MLOC's multi-level architecture.

C. File Organization on Parallel File Systems

How large files are partitioned and organized on PFS is critical to I/O performance. The basic approaches include *single shared file* and *unique files* approaches. Researchers have studied advantages and drawbacks for both [20] [21] in terms of throughput, metadata management, and file lock overhead. Subfiling [22] [23] was presented, which stands between the two above approaches. This method achieves balanced I/O throughput and file management overhead by dividing big files into blocks which are neither too large nor small. Thus, based on total size of the data, the chunk size and the number of bins, MLOC stores data within each bin in a single file. To yield suitable file sizes, the compressed index and data are stored separately in their bin files as shown in Figure 4. MLOC adopts this file organization for two reasons.

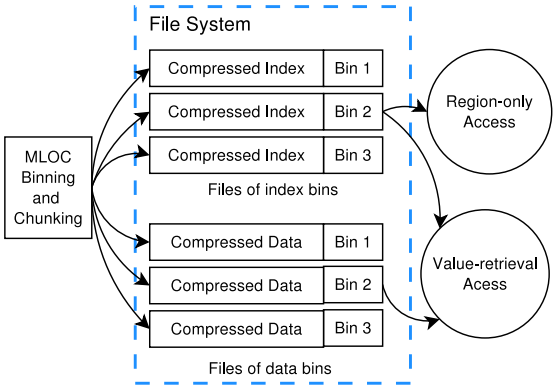


Fig. 4. An overview of the data layout on the disk. The indices and data for each bin are stored in separate files and accessed based on the requirements of the data access patterns.

Firstly, this approach generates files that are neither too small for the metadata nor too large for management, balancing their respective performances and overheads. For datasets of several hundred gigabytes, if 100 bins are used, each bin will be several gigabytes. Second, in subsequent data access and analysis, files are opened as read-only; thus there will be little overhead in file lock control since there are no requirements for synchronization of reading as there is for writing.

PFS apply file striping, in which files are divided into stripes and distributed to multiple Object Storage Targets (OST) [4]. Previous work showed that data access patterns should be aligned with stripe size to achieve best parallel access performance [24]. Thus, MLOC adjusts the chunk size based on total size and binning to ensure that the size of the smallest unit accessed is within one stripe (e.g., 1MB).

D. Parallel Data Access Requests Handling

MLOC applies MPI and MPI-IO [25] for parallel data access request handling and returns qualified data to users with high efficiency. The overview of the data access processing is shown in Figure 5. Given an access pattern on a sub-region/volume with a set of *VC* and *SC*, MLOC first decides the bins to access by comparing the bounds of *VC* and the bounds of bins. Then, MLOC calculates the blocks to access in the chosen bins by mapping the blocks in the accessed region to the Hilbert curve order. Each MPI process fetches and processes a subset of blocks across the bins, as shown in Figure 5. Equal numbers of blocks are assigned to processes to achieve load balancing. Moreover, the assignment of blocks follows the column order, in which as many blocks as possible within a single bin are assigned to a single process. Since data within each bin is stored in a single file, the column order ensures that each process accesses the least number of bins and thus the least number of files. In this way, I/O contention due to multiple processes accessing a single file is minimized.

Each MPI process fetches data blocks from bin files, decompresses them, and filters them based on *VC* and *SC* to get local results. The root process then gathers results from all processes to return the final results to users. In the following subsections, we describe handling data access requests for access patterns

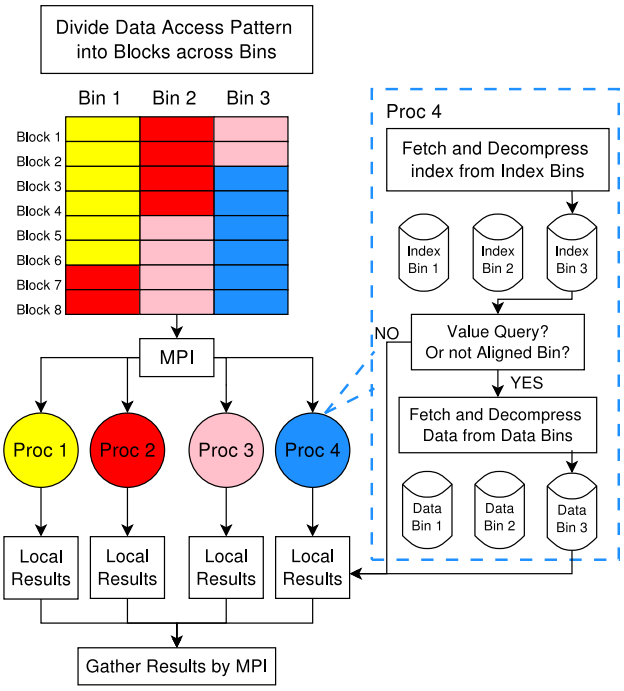


Fig. 5. An overview of the workflow of parallel data access processing. Blocks of the same color are assigned to the process of the same color.

generated by region-only queries, value queries, multi-variable queries, and multi-resolution support, respectively.

1) Region-only Access: Region-only access patterns are generated by queries to answer the positions of points that satisfy a specific *VC*. In this case, the values of variables are not required as output. Furthermore, if bins accessed are *aligned bins*, the values of points do not need to be reconstructed. As shown in Figure 5, for the *aligned bins*, MLOC only needs to access the index bins to answer the query without reconstruction. Significant time is saved through the reduction of I/O and CPU overhead.

For *misaligned bins*, there might be spatial regions that fail to satisfy the *VC*. Since determining in advance which of these points violate the *VC* is impossible for compressed data, all the values in the spatial region must be decompressed/reconstructed. The processing and filtering for *misaligned bins* are similar to the value-retrieval access pattern.

2) Value-retrieval Access: Value-retrieval access patterns are generated by queries that answer the exact values of points that satisfy specific *VC* and *SC*; exact values of points are required as output. So for MLOC, the values of all points satisfying the constraints need to be reconstructed by decompression. MLOC has to access data bins to fetch the requested data. Given a value query with *SC*, MLOC first decides which blocks to access by the spatial coordinates and then re-maps them to the *HSFC* order they are organized in on storage. The blocks are then fetched from storage for decompression and filtering. If *VC* are specified, only the bins within the blocks that satisfy the *VC* will be fetched and decompressed. The retrieved data blocks are decompressed and points within blocks are filtered based on *VC* and *SC*.

3) Multi-resolution Access: As shown in Figure 2, data in different *PLoD* levels can be stored separately. For access to a certain *PLoD* level, since data within the level are stored together, minimum seek and decompression are required only for the bytes of data that needs to be accessed. For example, if *PLoD* level 2 is required, only the first three bytes of the values are fetched from the storage and decompressed. MLOC assembles the three bytes to double-precision based numbers by appending five other dummy bytes. MLOC does not fill zeros in dummy bytes, since this will further decrease precision as the assembled values are always smaller than the original. Instead, MLOC fills $0x7F$ in the first byte and $0xFF$ in the remaining to improve average precision.

4) Multi-variable Access: In multi-variable data access patterns, spatial regions are usually selected by the values of one (or more) variable(s); values of other variables are fetched on the corresponding spatial regions. Thus, the process can be decomposed into two steps: region-only access for the first variable(s) and value-retrieval access for others. The indices derived by the first step can be directly used on other variables to retrieve, decompress, and filter results.

MLOC applies a light-weight and high-performance bitmap indexing to optimize for multi-variable data access. Spatial indices are represented by bitmaps to minimize the memory footprint and communication overhead, which are crucial performance metrics in parallel HPC environment. Bitmaps derived by region queries from all processes are synchronized for the use of value-retrieval on the other variables.

IV. Results

A. Experiment Setup

We conducted extensive experiments to evaluate MLOC's performance on Parallel File Systems in an HPC environment. All experiments were conducted on the Lens cluster at Oak Ridge National Lab, which runs on the Lustre parallel file system. Lens is a 32 node Linux cluster dedicated to data analysis and high-end visualization, which is suitable to serve as the testbed for MLOC. Each node contains four quad-core 2.3 GHz AMD Opteron processors with 64 GB of memory.

1) Datasets: We use two datasets generated by scientific simulation codes running on supercomputers for evaluation. As these scientific codes are typical, the experimental results are representative of MLOC's capability of dealing with various scientific datasets. The datasets we use are from:

- GTS [1], a particle-based simulation for studying plasma microturbulence in the core of magnetically confined fusion plasmas of toroidal devices in nuclear reactors. The data is 1-D, so we aggregate from multiple time steps to form a 2-D data space. In each time step, there are $32,768 \times 32,768$ double-precision floating point numbers, totaling to 8GB. To demonstrate MLOC's capability in processing large datasets, we replicate the dataset to 512 GB, which is $262,144 \times 262,144$ 2-D double-precision data. We use a chunk size of $2,048 \times 2,048$ to achieve an optimized unit size. The replication does not interfere with the experiment results, since in all experiments we perform queries with random value and spatial constraints and report the average. Also, MLOC's inherent benefits from its storage architecture help reduced data complexity and heterogeneity.
- S3D [2], a first-principles-based direct numerical simulation of reacting flows that aids the modeling and design of combustion devices. In each time step, we use $128 \times 128 \times 128$ 3-D double-precision data and replicate it to form $1024 \times 1024 \times 1024$ 8 GB datasets. We also further replicate it to 512 GB, which is $4,096 \times 4,096 \times 4,096$. We use a chunk size of $128 \times 128 \times 128$.

For all datasets, we divide them into 100 equal-frequency bins. This is achieved by calculating the binning boundaries from partial dataset, and then applying the boundaries to the whole dataset so that each bin has almost the same number of elements. For MLOC, we assume that value-constrained accesses are optimized first, followed by multi-resolution accesses, and finally spatial-constrained accesses, abbreviated as the *V-M-S* order. We also evaluate and compare with other orders under different data access patterns.

2) Scenarios for Comparison: Among existing systems, we chose FastBit and SciDB to compare with. FastBit is the state-of-the-art bitmap indexing tool for answering region queries. We use FastBit's precision binning based on the cardinality of bit patterns in double-precision scientific data, found to have the best response time. SciDB is designed to access sub-plane/volume of multi-dimensional matrices using chunking techniques [26], which is optimized for spatial-constrained access patterns. In the experiments, we apply the same chunking sizes as MLOC. We also compare with naïve sequential scan in which data arrays are linearized on disk in row-first order and are accessed by calculating the file offsets based on multi-dimensional spatial constraints. As a result, the performance of sequential scan mainly depends on the size of query region as determined by the spatial constraints.

We use three MLOC-based approaches for comparison: MLOC-COL, MLOC-ISO, and MLOC-ISA. Each of them uses a specific compression technique. MLOC-COL applies *V-M-S* optimization order (shown in Figure 2) and compresses byte columns using *Zlib*, yielding good support for byte-level multi-resolution access. MLOC-ISO applies ISOBAR, a high-efficiency lossless compression technique, for double-precision scientific data. MLOC-ISA applies ISABELA, a lossy compression technique that achieves high compression ratios with user-specified error-rates. These three approaches represent different requirements in scientific computing.

In all experiments, data is assumed to be located on file system storage instead of physical memory (RAM). This is crucial as the explosive growth of data makes it impossible for RAM to hold. Since the size of indices can also be huge like in FastBit, they should also be assumed to be on disks. To ensure this, after each round we clear the system file cache so that all data and indices are fetched from disk for all accesses. Random value and spatial constraints with certain selectivity are generated for queries, and in all sets of experiments we report the average results of 100 random queries.

TABLE I
SPACE REQUIREMENTS OF DATA AND DBMS INDEX FOR 8 GB RAW DATA. *SCIDB REPLICATES DATA ALONG CHUNK BOUNDARIES TO MINIMIZE THE NUMBER OF CHUNKS READ BY QUERIES, INCREASING THE DATA SIZE OVER THE RAW DATA.

	Data size	Index size	Total size
MLOC-COL	6.5 GB	1.6 GB	8.1 GB
MLOC-ISO	6.9 GB	1.6 GB	8.5 GB
MLOC-ISA (lossy)	**1.6 GB**	1.6 GB	**3.2 GB**
Seq. Scan	8.0 GB	N/A	8.0 GB
FastBit	8.0 GB	10.0 GB	18.0 GB
SciDB*	8.8 GB	N/A	8.8 GB

TABLE II
REGION QUERY RESPONSE TIME (SEC.) ON 8 GB DATASETS. NO *SC* ARE SET. VALUE SELECTIVITY IS 1% AND 10%.

	1% GTS	10% GTS	1% S3D	10% S3D
MLOC-COL	0.53	1.21	0.59	1.62
MLOC-ISO	0.41	**1.10**	**0.53**	**1.57**
MLOC-ISA	**0.34**	1.23	0.56	1.66
Seq. Scan	19.22	20.27	22.71	22.93
FastBit	36.81	37.48	37.27	37.83
SciDB	206.80	677.10	210.00	597.80

B. Storage Space Overhead Measurement

Table I summarizes the storage requirements to store and index 8 GB data under all scenarios: MLOC with different compression techniques, sequential scan (on raw data), FastBit, and SciDB. The table shows that by applying data reduction with compression techniques, MLOC reduces the total size of data and index to 38% of the raw data size using lossy ISABELA compression, and around 105% for lossless compression techniques, which is much smaller than FastBit (225% of raw data) using precision-based bitmap indexing, since the bitmaps produce large storage overhead for 100 bins. This data reduction not only saves disk storage space but also shortens I/O read time. While MLOC-ISA achieves the best data reduction, the compression is lossy and thus may not be desirable for computing with strict precision requirements.

C. Data Access Performance

1) Query-driven Data Access Patterns: We evaluate and compare the performance of value-constrained access patterns driven by *region queries* and spatial-constrained access patterns driven by *value queries* for MLOC and other approaches. For region queries, the value selectivity ranges from 1% to 10% of the total size of the dataset, and no spatial-constraints (*SC*) are set. For value queries, the region selectivity ranges from 0.1% to 10%, and no value constraints (*VC*) are set. For both access patterns, we test on 8 GB and 512 GB datasets and show their results, respectively. For MLOC-based approaches, 8 cores on the server are utilized for MPI-based parallel data access and decompression. 8 cores were also used for FastBit, and up to 8 cores were used by SciDB, as the number of cores used vary based on chunk partitioning.

2) 8GB Datasets Results: Table II shows the performance of value-constrained region-only data access patterns on 8 GB datasets in terms of response time for region queries. MLOC-based approaches achieve much better performance in all scenarios due to the fine-grained value-based binning greatly narrowing down the search space. FastBit does not perform well because it assumes the bitmap indices to be stored in the memory to achieve best performance, so it must load the huge indices into memory from disk each time before query executions. However, due to the huge size of datasets and bitmap indices FastBit produces, it is infeasible to assume them to be held in the memory. For SciDB and sequential scan, performances are poor since the whole dataset needs to be scanned to select the points that satisfy the *VC*.

TABLE III
VALUE QUERY RESPONSE TIME (SEC.) ON 8 GB DATASETS. NO *VC* ARE SET. REGION SELECTIVITY IS 0.1% AND 1%.

	0.1% GTS	1% GTS	0.1% S3D	1% S3D
MLOC-COL	3.07	5.06	3.51	5.26
MLOC-ISO	2.15	4.99	2.96	4.51
MLOC-ISA	**1.52**	**3.31**	**1.63**	**3.42**
Seq. Scan	4.38	5.92	1.81	4.75
FastBit	37.29	38.24	37.49	39.70
SciDB	29.10	122.50	143.20	469.10

TABLE IV
REGION QUERY RESPONSE TIME (SEC.) ON 512 GB DATASETS. NO *SC* ARE SET. VALUE SELECTIVITY IS 1% AND 10%.

	1% GTS	10% GTS	1% S3D	10% S3D
MLOC-COL	16.51	**41.18**	**18.94**	**39.25**
MLOC-ISO	**15.81**	42.06	19.43	41.55
MLOC-ISA	16.42	42.19	20.23	43.71
Seq. Scan	1596.52	2317.39	1423.45	2179.81

Table III shows the performance of spatial-constrained value-retrieval data access patterns on 8 GB datasets in terms of response time of value queries. Sequential scan achieves good performance because it only needs to access a small portion of data by spatial indexing. Note that this value query has no *VC*, which means all bins have to be accessed to fetch points within the *SC*. However, the within-bin curve ordering increases locality and minimizes the number of seeks within each bin. Furthermore, less I/O is needed due to compression. Thus, MLOC (with ISABELA as a backend) outperforms the others. The performance of FastBit for value queries is similar to region queries as it must still load the entire index before processing the query, which is the major overhead.

3) 512 GB Datasets Results: For 512 GB experiments, we only compare to sequential scan as the other approaches already show poor performances on smaller datasets. Table IV and Table V show the performance results for the two access patterns by region queries and value queries, respectively. MLOC-based approaches achieve better performance again, demonstrating MLOC's strong scalability on large scientific datasets. For region queries, sequential scan must read the entire 512 GB dataset to select qualified points, taking a significant amount of time due to the size. For value queries, MLOC-ISA achieves the best performance when accessing smaller portions of data due to the smaller size (Table I). However, when accessing larger portions of data (1% of 512 GB), the response time exceeds other MLOC approaches due to increased overhead of performing *B*-spline interpolation to recover original values. Still, MLOC-ISA's savings in storage space is a very attractive feature in certain cases.

TABLE V
VALUE QUERY RESPONSE TIME (SEC.) ON 512 GB DATASETS. NO *SC* ARE SET. REGION SELECTIVITY IS 0.1% AND 1%.

	0.1% GTS	**1% GTS**	**0.1% S3D**	**1% S3D**
MLOC-COL	13.25	33.03	15.24	39.34
MLOC-ISO	8.81	**23.77**	9.96	**37.66**
MLOC-ISA	**7.82**	40.99	**8.39**	44.04
Seq. Scan	37.22	248.87	40.74	230.26

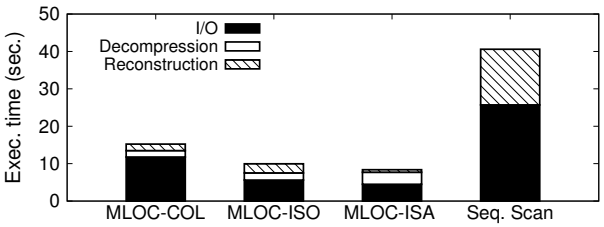


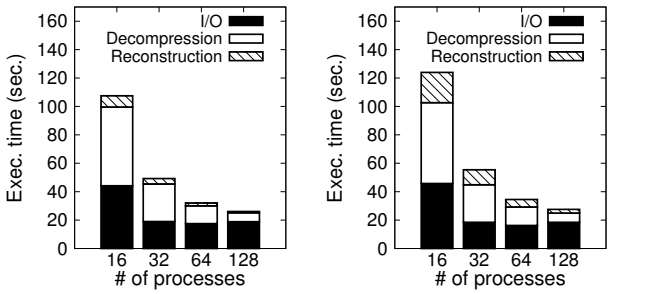
Fig. 6. Execution time of different components for value-retrieval data access (0.1% selectivity) on 512 GB S3D datasets.

4) Component Analysis: To better understand the performance for value-retrieval data accesses, we divide each data access into three parts: I/O, decompression, and reconstruction, and measure the overhead for each. I/O includes seek and read operations to load required data from storage to memory. Decompression is the time required to decompress the buffers. Reconstruction consists of filtering out unqualified values and assembling the final results. Figure 6 shows the overhead of different components for MLOC and sequential scan. Although decompression brings additional overhead, it successfully combines with the light-weight indexing and parallel optimization of MLOC to reduce I/O overhead and achieve improved performance. MLOC-ISA yields the least I/O time due to the best reduction in data size but spends more time in decompression because of the *B*-spline algorithm.

5) Parallel Scalability Analysis: To demonstrate MLOC's capability in parallel access on large datasets, we perform value queries using 10% selectivity on the 512 GB, replicated GTS/S3D datasets, and larger number of MPI processes. Figure 7 shows how performance improves as more processes are used, especially, for decompression and reconstruction. However, I/O does not scale well since more processes bring more I/O contentions on limited I/O resources. Yet, MLOC still achieves high throughput of 2 GB/s with 128 processes, which is much faster than similar techniques [27] that achieve throughput of several hundred MB/s in a similar environment.

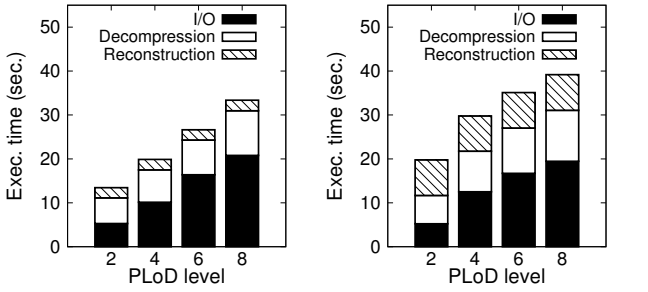
D. Multi-resolution Access Evaluation

1) Data Access Performance: We only show results for precision-based multi-resolution layout and access, since the subset-based approach has been widely studied by previous works [13]. Figure 8 shows the data access performance when accessing data of different *PLoDs* using MLOC-COL on 512 GB datasets. Accessing data of lower *PLoDs* significantly reduces response time. The majority of the reduction is in I/O time, as less bytes are fetched from storage to assemble the original values. Decompression time is not reduced significantly because the third to the eighth bytes of double-precision



(a) GTS data. (b) S3D data.

Fig. 7. Performance of value queries with more processes, 10% selectivity on 512 GB datasets.



(a) GTS data. (b) S3D data.

Fig. 8. Multi-resolution data access performance for value queries (1% selectivity) under different *PLoDs* on 512 GB datasets.

numbers are regarded as incompressible so that original bytes are stored. Thus, decompression is almost unnecessary when assembling the third to the eighth bytes. Reconstruction time remains the same since it is performed after the bytes are fetched and assembled and is irrelevant to the *PLoDs* used.

2) Accuracy Measurement for Data of Different PLoDs: To evaluate the precision of data in different *PLoDs*, we perform histogram construction and *K*-means clustering on the original data and data of different *PLoDs* using three variables of S3D data, each with 20 million points. In the histogram construction, equal-width histograms are constructed for original data first. These bin boundaries are then applied to data in different *PLoDs* to calculate the error rate, which is the number of points that fall into different bins compared to results for original data. In the *K*-means clustering experiments, *K*-means clustering is performed for both original data and data of different *PLoDs*. Mis-classifications are calculated, which refers to the percentage of points of each *PLoD* that were assigned to a different class than their corresponding original data points. The clustering was run for 100 iterations, using randomized centroids each time. Table VI shows that for both experiments, using 2-byte *PLoD* produces rather high error rate. However, 3-byte *PLoD* already produces very low error rates, and the error rates for 4-byte *PLoD* are almost negligible. Thus, through the usage of different *PLoDs*, we can achieve a reduction in analysis time with acceptable loss of precision.

To evaluate the performance under different orders of opti-

TABLE VI
HISTOGRAM AND K-MEANS ERROR RATES FOR S3D DATA.

Num Bytes	Histogram error			K-means error
	vu	*vv*	*vw*	*vv and vw*
2	8.241%	1.83%	1.834%	4.290%
3	0.029%	6.5E-3%	8.3E-3%	0.017%
4	1.6E-4%	4.5E-5%	3.5E-5%	6.6E-5%

TABLE VII
QUERY RESPONSE TIME (SEC.) OF DIFFERENT ORDERS OF OPTIMIZATIONS FOR VALUE-RETRIEVAL DATA ACCESS (1% SELECTIVITY) ON 512 GB S3D DATASETS.

	3-byte PLoD access	Full-precision access
V-M-S order	**19.45**	39.34
V-S-M order	23.70	**35.47**

mization placement in MLOC, we compare the default *V-M-S* order with an alternative, *V-S-M* order. In *V-S-M* order, value-constrained data accesses are optimized by binning at the first place, spatial-constrained accesses at the second place, and multi-resolution accesses at the third place. Table VII shows performance of the two orders on two of access patterns. For multi-resolution access with *PLoD* using three bytes, *V-M-S* achieves better performance, since it optimizes multi-resolution access at a higher priority by storing the first two bytes of all blocks contiguously to reduce I/O overhead. For *V-S-M*, it must go through the files to retrieve the first two bytes of each point, which are distributed over different areas. For full-precision access, in which all bytes of each point need to be fetched and assembled, the results are opposite since *V-M-S* has to access different bytes of points, which are stored separately on disks. However, in both cases, the difference for the two orders are not significant since both of them also optimize for the other two access patterns, although not in the primary place. This means MLOC's multi-level optimization is flexible and robust to the changes of access patterns. Users can specify different orders of optimizations to achieve best performance for the most frequently used access patterns with minimal impact on other access patterns.

V. RELATED WORK

Organization of data on disks is crucial to optimizing throughput of various analytics of large datasets. Simple chunking has been used in various manners, such as SciDB [28], which addresses sub-volume access patterns by applying array slicing, joining operations, and array division into regular/irregular chunks [26] for multi-dimensional scientific data. The majority of the work has been on spatial division of chunks and on performing algorithms at the chunk level rather than providing structures optimized for value-constrained region-only access. ISABELA-QA [15] operates on windows of chunked data before binning to enable efficient data access of value-constrained queries. It does this directly on ISABELA-compressed data [16], which uses B-spline interpolation with user-defined error rates. FastBit [14] optimizes for value and region constrained accesses by building binned bitmap indices based on values, keeping similarly valued data in the same bin. The bitmaps improve the response time of range queries based on values. However, FastBit indices have large space requirements of 30% to 200% of the raw data size in addition to requiring access to the raw data. Both ISABELA-QA and FastBit are optimized only for value-constrained access patterns and cannot efficiently handle spatial-constrained access patterns on sub-volumes.

More complex layouts attempt to optimize chunk access through spacial locality, particularly through Space-Filling Curves (*SFC*) [11]. SFC have been studied to map multi-dimensional data to one-dimensional space in order to achieve high spatial locality. A hierarchical $Z-$order indexing scheme develops a storage layout for rectilinear grids of data [13], while the usage of Hilbert space-filling curves (*HSFC*) in indexing sped up multi-dimensional data retrieval [12]. These works try to understand the nature of placing chunks of data in SFC order to increase throughput by improving spatial locality and minimizing seeks. EDO [27], an elastic data organization approach, uses Hilbert curve re-ordering and chunking to optimize parallel access to subsets of multi-dimensional data on parallel file systems. However, EDO only optimizes for spatial-constrained sub-volume access. In contrast, we present a hybrid approach that integrates various optimization techniques combined with *HSFC* mapping to ensure optimized performance for heterogeneous access patterns.

Given prior access patterns, prediction-based layouts can be used to optimize for future accesses. BORG [29], a self-optimizing storage system, performs automatic block reorganization based on the observed I/O workload. Resonant I/O [24] optimizes data striping patterns on the I/O nodes for particular request patterns. Such works assume that data access patterns are known or can be determined with the optimizations only serving that particular access pattern, which is difficult as the patterns are usually hard–to–predict. In contrast, we build a storage layout scheme that satisfies the requirements of heterogeneous access patterns.

Accompanying storage layout optimizations are considerations of the parallel file systems, particularly in regards to file size. Large shared files can achieve optimal performance in certain scenarios [21]. Improved performance was achieved by using small unique files for each process in select cases [30], though the bottlenecks are induced by large numbers of small files, and several architectural designs and optimizations of parallel file systems for small files can address these bottlenecks [31]. As a middle ground between these two filing methods, subfiling [22] [23] divides large files into smaller subfiles to achieve balanced performance between file opening and I/O throughput. As the access patterns are hard–to–predict, we use subfiling to achieve a balanced performance regardless of individual scenarios.

VI. CONCLUSION

In this paper we presented MLOC, a *M*ulti-level *L*ayout *O*ptimization framework for *C*ompressed Scientific Data. MLOC addresses the problem of analyzing large scientific datasets with *heterogeneous*, *arbitrary*, and *hard-to-predict* access patterns by using a flexible multi-layer pipeline to

optimize data layouts for multiple access patterns while allowing user-driven prioritization. These layers optimize value-constrained access through value-based binning and spatial-constrained access through chunking in space-filling curve order and enable multi-resolution support through the subset-based or Precision-based Level of Detail (*PLoD*) access, allowing a reduction in I/O necessary for various analysis functions. MLOC additionally provides both lossy compression (such as ISABELA) and lossless compression (such as *Zlib*) and organizes the resulting chunks using subfiling for improved access on parallel file systems.

MLOC shows improvements over a number of access patterns while minimizing the size of the data and index on disk. MLOC also scales to larger scientific datasets through data placement on disks and MPI/MPI-IO parallel implementation. Usage of *PLoD* allows a further reduction in response time with a negligible error rates for analysis functions. We are working on integrating MLOC with I/O middleware running in large parallel environments to further test and improve its run-time performance. Also, the data reorganization and parallel processing model in MLOC can be extended to other parallel data processing models (e.g., MapReduce) to improve the performance of data accesses with heterogeneous patterns.

VII. Acknowledgements

We would like to thank ORNL's leadership class computing facility, OLCF, for the use of their resources. This work was supported in part by the U.S. Department of Energy, Office of Science and the U.S. National Science Foundation (DE-1240682, DE-1028746). Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under contract no. DEAC05-00OR22725.

References

[1] W. X. Wang and et al, "Gyro-kinetic simulation of global turbulent transport properties in Tokamak experiments," *Physics of Plasmas*, vol. 13, no. 9, p. 092505, 2006.

[2] J.H. Chen, A. Choudhary, and et al, "Terascale direct numerical simulations of turbulent combustion using S3D," *Computational Science & Discovery*, vol. 2, no. 1, p. 015001, 2009.

[3] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *First USENIX Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, Jan. 28-30 2002.

[4] S. Donovan, G. H. Ibm, and et al, "Lustre: Building a file system for 1,000-node clusters," in *Proceedings of the Linux Symposium*, 2003.

[5] P. H. Carns, W. B. Ligon, III, and et al, "PVFS: a parallel file system for linux clusters," in *Proceedings of the 4th annual Linux Showcase & Conference - Volume 4*, Berkeley, CA, USA, 2000, pp. 28–28.

[6] J. F. Lofstead, S. Klasky, and et al, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," in *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, 2008, pp. 15–24.

[7] "HDF5." [Online]. Available: http://www.hdfgroup.org/HDF5/

[8] J. Li, W. keng Liao, and et al, "Parallel netCDF: A high-performance scientific I/O interface," in *Proceedings of Supercomputing*, Nov. 2003.

[9] H. Abbasi, M. Wolf, and et al, "DataStager: scalable data staging services for petascale applications," in *HPDC '09*. New York, NY, USA: ACM, 2009, pp. 39–48.

[10] F. Zheng, H. Abbasi, and et al, "PreDatA–preparatory data analytics on peta-scale machines," in *IPDPS*, Atlanta, GA, April 2010.

[11] H. Sagan, *Space-Filling Curves*. Springer-Verlag, New York, NY.

[12] J. K. Lawder and P. King, "Querying multi-dimensional data indexed using the Hilbert space-filling curve," *SIGMOD Record*, vol. 30, 2001.

[13] V. Pascucci, "Global static indexing for real-time exploration of very large regular grids." ACM Press, 2001.

[14] K. Wu, "FastBit: An efficient indexing technology for accelerating data-intensive science," *Journal of Physics: Conference Series*, vol. 16, no. 1, p. 556, 2005.

[15] S. Lakshminarasimhan, J. Jenkins, and et al, "ISABELA-QA: Query-driven data analytics over ISABELA-compressed scientific data," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, Seattle, Washington, 2011.

[16] S. Lakshminarasimhan, N. Shah, and et al, "Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data," in *17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011)*, Bordeaux, France, 8 2011.

[17] B. Moon, H. V. Jagadish, and et al, "Analysis of the clustering properties of the Hilbert space-filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, p. 2001, 2001.

[18] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 1245–1250, 2006.

[19] E. R. Schendel, Y. Jin, and et al, "ISOBAR preconditioner for effective and high-throughput lossless data compression," *International Conference on Data Engineering (ICDE)*, 2012.

[20] W.-k. Liao and A. Choudhary, "Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols," in *Proceedings of the ACM/IEEE conference on Supercomputing (SC)*, Piscataway, NJ, USA, 2008, pp. 3:1–3:12.

[21] S. Lang, P. Carns, and et al, "I/O performance challenges at leadership scale," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 40:1–40:12.

[22] K. Gao, W.-k. Liao, and et al, "Using subfiling to improve programming flexibility and performance of parallel shared-file I/O," in *Proceedings of the 2009 International Conference on Parallel Processing*, ser. ICPP '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 470–477.

[23] W. Yu, J. Vetter, and et al, "Exploiting Lustre file joining for effective collective IO," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, ser. CCGRID '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 267–274.

[24] X. Zhang, S. Jiang, and K. Davis, "Making resonance a common case: A high-performance implementation of collective I/O on parallel file systems," in *Proceedings of the IEEE International Symposium on Parallel&Distributed Processing*, Washington, DC, USA, 2009, pp. 1–12.

[25] W. Gropp, E. Lusk, and R. Thakur, *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press, Cambridge, MA, 1999.

[26] E. Soroush, M. Balazinska, and D. Wang, "ArrayStore: A storage manager for complex parallel array processing," in *Proceedings of the 2011 International Conference on Management of Data*, ser. SIGMOD '11. New York, NY, USA: ACM, 2010.

[27] Y. Tian, S. Klasky, and et al, "EDO: Improving read performance for scientific applications through elastic data organization." in *Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, ser. Cluster '11. Austin, TX, USA: IEEE, 2011.

[28] P. G. Brown, "Overview of SciDB: Large scale array storage, processing and analysis," in *Proceedings of the 2010 International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 963–968.

[29] M. Bhadkamkar, J. Guerra, and et al, "BORG: block-reORGanization for self-optimizing storage systems," in *Proccedings of the 7th conference on File and storage technologies*. Berkeley, CA, USA: USENIX Association, 2009, pp. 183–196.

[30] W. keng Liao, A. Ching, and et al, "An implementation and evaluation of client-side file caching for MPI-IO," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS), Long Beach, California*, March 2007.

[31] P. Carns, S. Lang, and et al, "Small-file access in parallel file systems," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–11.