A Data Optimizer for Region-Aware Self-describing Files in Scientific Computing

Yanjie Song
School of Info. Sci. & Tech.
ShanghaiTech University
Shanghai, China
songyj@shanghaitech.edu.cn

Tianyuan Wu
School of Info. Sci. & Tech.
ShanghaiTech University
Shanghai, China
wuty@shanghaitech.edu.cn

Yuanhao Li
School of Info. Sci. & Tech.
ShanghaiTech University
Shanghai, China
liyh12022@shanghaitech.edu.cn

Guancheng Li
School of Info. Sci. & Tech.
ShanghaiTech University
Shanghai, China
ligch2022@shanghaitech.edu.cn

Yuchen Liu
School of Info. Sci. & Tech.
ShanghaiTech University
Shanghai, China
liuych1@shanghaitech.edu.cn

Shu Yin*
School of Info. Sci. & Tech.
ShanghaiTech University
Shanghai, China
yinshu@shanghaitech.edu.cn

Wei Xue
Dept. of Computer Sci. and Tech.
Tsinghua University
Beijing, China
xuewei@tsinghua.edu.cn

Junchao Wang
China Meteorological Administration
Basin Heavy Rainfall Key Lab/
Hubei Key Lab for HV. Rain Mtrg. & WRNG.
Wuhan, Hubei, China
nanxingong@163.com

ABSTRACT

Acquiring data from scientific simulations for analytical purposes is inherently challenging due to the complex and irregularly shaped regions within which the data resides, particularly when using self-describing data formats. The process of region-based data distillation becomes even more arduous when employing persistent memory or parallel file systems. To tackle this challenge, we introduce RASTER (Region-Aware Self-describing daTa optimizER), a lightweight middleware designed for region-aware data preprocessing. RASTER dynamically reorganizes data into variable groups based on regional identifiers during runtime, thereby eliminating the need for sequential searches to locate the required data. We have developed a prototype of RASTER and successfully integrated it into three distinct computing environments: a single-node server equipped with Intel® Optane™ DC persistent memory, the Huawei® OceanStor cloud storage platform, and the Sunway TaihuLight supercomputer. We then conducted a thorough evaluation of the RASTER prototype on the latter two platforms using a real-world scientific application, CESM (Community Earth System Model). Our experimental results demonstrate that RASTER enhances data acquisition performance by up to 2.83× and achieves a 2.36× speedup over conventional netCDF and the state-of-the-art ADIOS2. Additionally, RASTER significantly reduces memory usage by up to 400%, showcasing its scalability potential.

CCS CONCEPTS

• **Software and its engineering** → **File systems management**; • **Computer systems organization** → *Cloud computing*.

KEYWORDS

high performance computing, cloud computing, file system

ACM Reference Format:
Yanjie Song, Tianyuan Wu, Yuanhao Li, Guancheng Li, Yuchen Liu, Shu Yin, Wei Xue, and Junchao Wang. 2024. A Data Optimizer for Region-Aware Self-describing Files in Scientific Computing. In

*Corresponding author.

SoCC '24, November 20–22, 2024, Redmond, WA, USA

ACM ISBN 979-8-4007-1286-9/24/11.
https://doi.org/10.1145/3698038.3698526

ACM Symposium on Cloud Computing (SoCC '24), November 20–22, 2024, Redmond, WA, USA. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3698038.3698526

1 INTRODUCTION

Scientific applications tend to generate and analyze a large amount of high-dimensional data in self-describing file formats such as HDF5 [16], netCDF [31], ADIOS [26], ASDF [21], FITS [36], and Zarr [15]. Improving the I/O performance of self-describing files is a long-standing problem, with several solutions proposed over the years. Studies such as PLFS [9], ADIOS [26], ADIOS2 [19], PnetCDF [25], and HDF5-PLFS plugin [28] aim to improve the parallelism of the data. Other optimizations, such as the combination of I/O operations and the joining of heterogeneous data sets, make data access more efficient and convenient [14, 18]. Managing self-describing data using chunking and indexing methods is crucial to optimize partial dataset acquisition and metadata querying. Bitmap indexing [10, 11, 20] and chunk sorting-based data re-organization [12, 13] accelerate dataset retrieval, while metadata indexing [40] enhances attribute querying.

However, domain scientists expect more than general I/O optimizations for self-describing data as they usually focus on a small portion of data out of the entire dataset, which consists of a specific set of variables. The data is often buried irregularly in files, which makes it difficult to organize and retrieve data. How to efficiently extract and acquire demanded data from an enormous dataset remains a crucial and growing problem, especially in fields of high energy physics [27], computational molecular dynamics [3], and climatology [24, 37].

Although next-generation storage devices such as persistent memory achieve better I/O throughput and lower latency [30], data acquisition in self-describing files is not accelerated as expected. In particular, heavy software stacks of the I/O libraries and poor data locality mean that hardware performance cannot be fully utilized. For example, the Community Earth System Model (a.k.a. CESM) usually reads a complex regional dataset (e.g., the Pacific Ocean dataset) from netCDF files into memory for further analysis. With the existing interface, CESM has to read a mixed dataset containing both desired and unrelated data according to the rectangle array (i.e., hyper-slab) access pattern. Reading the mixed dataset increases the I/O time and occupies more memory for the unrelated data. We learned from real-world applications that most regional division information is deterministic during the compute stage and has already been written to files as an array named *REGION_MASK*. Unfortunately, to our best knowledge, no approaches use such information to improve data acquisition efficiency. For example, the Weather Research and Forecasting (a.k.a. WRF) application writes the simulated data by variables instead of regions because region boundaries are usually irregular and hence hard to divide with existing interfaces. Although the reason is understandable, it inconveniences users and systems when performing complex data analysis.

As a common optimization approach, HDF5 grouping and chunking provides a flexible interface to manage hierarchical and heterogeneous scientific data [17]. Despite this ingenious design, there are many restrictions on this interface. Unfortunately, the read pattern of the applications usually does not match the write patterns, making it difficult to benefit from the chunking interface. For example, the read performance would be degraded if an application tries to read a row-wise chunked dataset by column or read a rectangle chunked dataset in irregular regions. Besides, users often misuse the HDF5 grouping interface to divide and manage many related multi-dimensional datasets to different groups. The misusage makes it even harder to retrieve the desired data. This ineffective utilization inspires us to repurpose the grouping and chunking interface to manage the datasets with richer semantics without introducing additional labor for API modification.

Based on these case studies, we present RASTER (Region-Aware Self-describing daTa optimizER), a middleware library to optimize irregular self-describing file accessing. It repurposes a set of existing interfaces for transparency and access optimization. RASTER partitions a large dataset into many variant-length chunks by mesh building mechanism and classifies these chunks to distinct HDF5 groups by the region semantics. When performing a regional read, only data related to the desired part will be brought to memory, significantly reducing I/O time and memory consumption. RASTER dynamically merges fix-sized HDF5/netCDF chunks with the same region identifier, producing fewer variant-length chunks. Consequently, RASTER reduces the number of iterative syscalls for retrieving chunks and the number of copies between kernel and user mode. We published the source code of RASTER at https://github.com/ShanghaiTech-LIONLAB/RASTER-SoCC24.

The main contributions of this paper include:

- To our best knowledge, RASTER represents the first library to optimize data organization for self-describing files based on region semantics.
- Instead of writing new APIs, RASTER repurposes the existing HDF5 grouping interfaces to conduct complex regional data accesses. RASTER ensures users achieve performance gain with the limited refactoring of existing code (within ten LOCs).
- RASTER enhances data acquisition by region identifiers while reducing software overhead, I/O access time, and memory usage.

- We have developed a RASTER prototype, integrated it into various computing platforms, and conducted extensive experiments under both benchmarks and real-world scientific applications.

The rest of the paper is organized as follows. Section 2 provides the background and motivation of this research. Design and implementation details of RASTER are proposed in Section 3, followed by an evaluation shown in Section 4. Section 5 discusses some lessons learned from this research, and Section 6 summarizes the related works. Finally, Section 7 concludes this paper.

2 BACKGROUND AND MOTIVATION

2.1 Challenges of HDF5/netCDF on fast storage

HDF5 [16] is a high-performance software library that provides a fast I/O interface to manage, access, and store heterogeneous scientific data. It is popular in the management of complex hyper-dimensional HPC data for its fewer restrictions on the number or size of data objects [27, 39]. Based on the HDF5 library, netCDF provides a higher level and more user-friendly interface for climatology, meteorology, oceanography, and molecular dynamics [1, 3, 4].

Management and accessing interleaved high-dimensional data is challenging. For example, computational fluid dynamics (CFD) applications like XCompact3D [6] use HDF5 to manage turbulence trajectories for visualizations. It stores a chronological series of particles' spacial positions and velocities in the form of a 4-dimensional HDF5 dataset. The dataset also contains environmental properties, which makes it hard to strip off particle values for the investigation perspectives. It also introduces read and memory overheads for unnecessary data. Molecular Dynamics (MD) simulators such as LAMMPS [3] and GROMACS [2] also use HDF5 or netCDF to manage traces via checkpointing. They divide the data into multiple sub-folders using the HDF5/netCDF grouping interface, each of which folder represents a sub-trace with a pre-defined time phase. It is convenient to read the entire data chronologically but hard to retrieve specific variables as it has to traverse multiple folders [23, 32].

As a common technique for partial dataset accessing, HDF5 chunking divides a large array into many fixed-size chunks, while a chunk is the basic I/O access unit [17]. With the chunking technique, the HDF5 library first looks for chunks belonging to the desired hyper-slabs in B-tree, then reads and copies them into the user's buffer. Compared with sequential data accesses, chunking layout enriches contiguous reads to a small portion of an N-dimensional array. The chunking performs well for many partial read scenarios as it reduces unnecessary data traverses in the sequentially stored dataset.

The dis-contiguous access delay becomes smaller on devices with better random I/O performance. However, the data management overhead for transformation and declaration becomes noticeable on NVRAM because of its ultra-low latency. We ran a test to evaluate the performance of reading a 10GB netCDF dataset with the commonly used netCDF-chunking layout on three types of storage devices: Hard Disk Drive (HDD), Solid State Drive (SSD), and Intel® Optane™ DC persistent memory (NVRAM) (shown in Fig. 1). The *yardstick* in the figure represents the peak performance of each device under the fully contiguous I/O pattern. We measured the yardstick by sequentially reading a 10GB netCDF file while neglecting its data format. Besides the performance improvement on faster devices, we observe that the access time of the netCDF-chunking layout is 10.3%, 9.3%, and 34.3% slower than the yardsticks accordingly. Note that netCDF introduces random reads to retrieve chunking indices, which causes the gap between the yardstick and netCDF chunking even under the sequential read scenario. The results support our findings that the HDF5 and netCDF data management overhead is getting more noticeable on fast storage devices with low latency.

Motivation 1: Self-describing data management mechanisms such as HDF5 and netCDF can not take full advantage of low-latency storage devices due to the overhead of software stacks. There is a great need to develop a lighter-weight

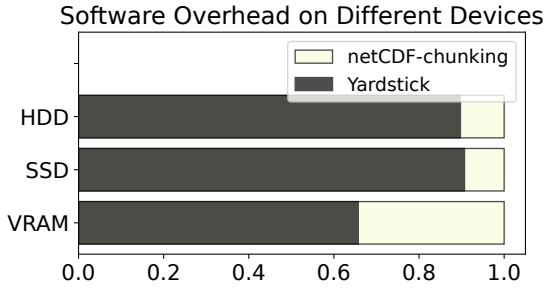


Figure 1: Read performance on different devices

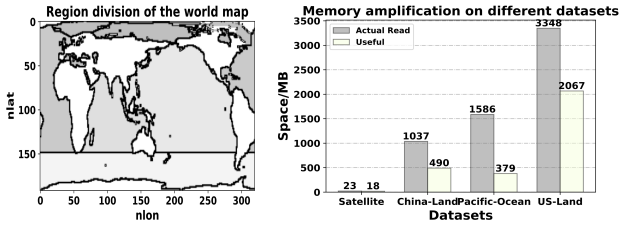


Figure 2: (a) World Region (b) Memory amplification

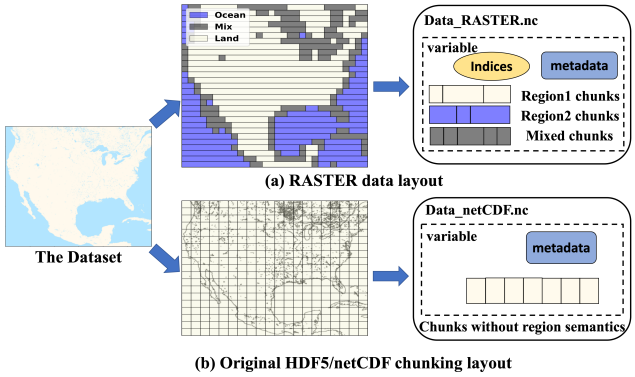


Figure 3: Difference on data layout between original HDF5/netCDF chunking and our design

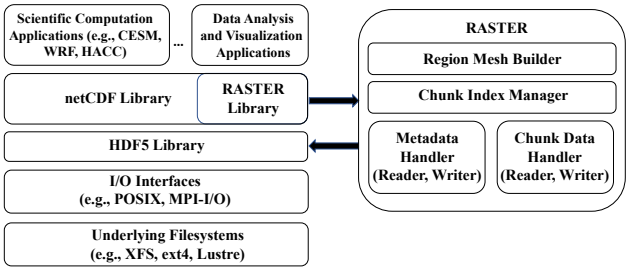


Figure 4: System Overview of RASTER

data optimizer for HDF5/netCDF to work with fast storage devices like NVRAM.

2.2 Inefficiencies in data pre-processing

Besides performance degradation by dis-contiguous data accesses, extracting desired data from regular-shaped arrays like hyper-slabs or bounding boxes consumes additional memory and processing time. For example, to obtain the Atlantic Ocean data from the CESM global dataset (Fig. 2(a)), climate scientists have to read the entire dataset into memory and crop the Atlantic with the *mask* variable. Note that scientific variables have rich semantics (e.g., longitude, latitude, altitude, time intervals), and domain scientists usually examine the data that belongs to specific regions [24, 37]. It is worth noting that the term *region* in HDF5 and netCDF is not restricted to geographic terrains. It can represent any user-defined area of data subsets. For example, a region can be a given spatial domain in high-energy particle investigations, a velocity range in computational fluid dynamics simulations, or political territories such as states or districts. The same characteristic of different types of *regions* is that they are often irregular in shapes so that the data with complex semantics scatter on the diverse HDF5 groups or netCDF chunks.

The consequence of dis-contiguous data management and accessing is that applications have to read multiple rectangle arrays of data (*hyper-slabs* in HDF5/netCDF) that contain the desired data. This data acquisition procedure is neither temporal nor spatial efficient because hyper-slabs read desired data along with unrelated ones. The increased data size requires additional read and process time to distill the desired data. Besides, it requires more memory to hold the data. We ran a test to evaluate the memory utilization of the CESM dataset and found a maximum 4.2× memory amplification when accessing the Pacific Ocean data shown above (illustrated in Fig. 2(b)). This means that to obtain 379MB of Pacific data, hyper-slabs have to read 1,586MB of data. It also proves that time consumption is more critical as distill operations are required. It implies why scientific applications are eager for efficient I/O even though the bandwidth and latency are getting better.

Motivation 2: There is a great need to design a new data layout that supports high-performance I/O while considering data semantics. In this way, the space and time overhead of reading useless data can be reduced, and scientific computation and data analysis applications can be accelerated.

These motivations drive us to develop a region-aware self-describing data optimizer (RASTER), a library that optimizes data acquisition and storage layout considering the access patterns and semantics of data. The key challenge of RASTER is to pass semantics like region information to file level and support complex regional reads. Another challenge is to achieve these new functions while not affecting existing parts or bringing additional software overhead.

3 DESIGN

3.1 Design Goals

RASTER should retain the original self-describing data format without affecting original functions or data operations. Second, RASTER should utilize device I/O performance by reducing dis-contiguous accesses. Third, RASTER aims to manage the dataset according to its semantics to enhance the efficiency in data acquisition and reduce memory usage.

As shown in Fig. 3(a), RASTER uses a set of region-aware, variant-length chunks to store multi-dimensional arrays instead of using the original fixed-sized chunks shown in Fig. 3(b). Therefore, applications can perform more contiguous reading when acquiring datasets, and software overhead is mitigated by reducing chunk numbers. Also, RASTER classifies chunks by region semantics and stores them in different groups according to the classification.

3.2 RASTER Architecture

Fig. 4 demonstrates the RASTER architecture. RASTER sits between netCDF and HDF5, and provides a standard `netCDF` interface to users. RASTER repurposed the HDF5 *grouping* interface to maintain transparency to both users and the underlying file system. RASTER Library overwrites the netCDF variable definition and functions, including chunking, reading, and writing while keeping the rest of the netCDF library unchanged. Developers only need to change the parameter named `size_t* chunksize` to an array that contains the region mask with less than ten LOCs modification. RASTER traps the multi-dimensional data I/O requests via `LD_PRELOAD` and stores the data as a group of objects. Instead of grouping data by variable types, *Region Mesh Builder* in RASTER extracts regionIDs from the dataset and guides *Chunk Index Manager* to group data into variant-length chunks by the IDs. *Metadata Handler* maintains an index table that stores the paired information of region identifier and the locations of corresponding chunks in metadata to guide *Chunk Data Handler* to perform actual data read and write.

Region Mesh Builder scans the data to be written on-the-fly and collects information on HDF5/netCDF chunks, including regionIDs, sizes, and offsets. Region Mesh Builder then combines the HDF5/netCDF chunks with the same regionID into a contiguous variant-length RASTER chunk and stores the RASTER chunk attributes (e.g., identifiers, offset, length, etc.) (detailed in Section 3.3.1). in a mesh structure

Chunk Index Manager takes the RASTER chunk attributes from the mesh structure as a guide to aggregate the chunks by regionIDs. It then constructs a KV-based index table for each regionID, where the key is the regionID and the value is the corresponding RASTER chunk information list (detailed in Section 3.3.2).

Metadata Handler maintains metadata of HDF5/netCDF files. Upon writing data in regions, Metadata Handler creates an HDF5 group structure for each regionID to store its index table and HDF5/netCDF attributes. When reading data with a given regionID, it loads the corresponding index table into memory to locate RASTER chunks associated with the ID. (detailed in Sections 3.3.3 and 3.3.4.

Chunk Data Handler performs the actual reads and writes of RASTER chunks to underlying file systems. Note that RASTER achieves better I/O performance by replacing the ordinary fixed-size HDF5 chunking operations with contiguous variant-length chunks. (detailed in Sections 3.3.3 and 3.3.4).

3.3 RASTER Operations and Implementation

In this section, we discuss the general operations of RASTER in building mesh structures, constructing index tables, and writing and reading data. We implement a RASTER prototype and integrate it into the HDF5/netCDF stack using a dynamical-linked library.

3.3.1 Mesh Construction. This subsection explains how *Region Mesh Builder* defines RASTER chunking variables in the function `nc_def_var_chunking`.

To write a 3-dimensional variable ($d \times m \times n$) with a 2-dimensional region mask ($m \times n$), Region Mesh Builder first constructs a region mesh structure for the variable. Recall that with predefined ($d \times h \times w$)-shaped chunks, HDF5/netCDF partitions variables into an array of fix-sized meshes, each of which is d in-depth, h in height and w in width. Let us use Fig. 3 as a simplified example to demonstrate the idea, where $d = 1$. Suppose ordinary HDF5/netCDF divides the variables representing the geographic world in a way of 20 columns by 20 rows, the variables are grouped into 400 fixed rectangles, each of which is a ($h \times w$) mesh (shown in Fig. 3(b)).

Note that some sequence of meshes in a column represents land variables while some represent ocean ones, Region Mesh Builder merges horizontally adjacent meshes with the same regionID and constructs a set of meshes in shapes like $(h \times k_1 w)$, $(h \times k_2 w)$, ... $(h \times k_n w)$. The values $(k_1, k_2, \ldots, k_n)$ represent the number of mergeable meshes (a.k.a. the number of chunks holding a specific regionID). The $(k_1, k_2, \ldots, k_n)$ vary with regionID values, leading to variant-length RASTER chunks. As a result, Region Mesh Builder stores the RASTER chunk attributes as `{chunkID, offset, length, mixed}` in a mesh structure, where `chunkID` identifies a RASTER chunk, `offset` is the start point of the chunk, `length` is the product of the pre-defined HDF5/netCDF mesh and the number of mergeable meshes, and `mixed` indicates if the RASTER

Table 1: An index table example (the last row in Fig. 3)

regionID	chunkID	offset	RASTER chunk len	mixed
Ocean	1	(76, 0)	(4×**28**)	False
	2	(76, 28)	(4×**4**)	True
	4	(76, 52)	(4×**4**)	True
	5	(76, 56)	(4×**8**)	False
	6	(76, 64)	(4×**4**)	True
	7	(76, 68)	(4×**12**)	False
Land	2	(76, 28)	(4×**4**)	True
	3	(76, 32)	(4×**20**)	False
	4	(76, 52)	(4×**4**)	True
	6	(76, 64)	(4×**4**)	True

chunk has multiple adjacent regions (e.g., the grey chunks in Fig. 3(a)).

Let us use the last row in Fig. 3 as an example while the predefined chunk shape is set to (4×4) (i.e., $d = 1, h = w = 4$). There are two regionIDs: `Ocean` and `Land` in the example. The mesh structure for the two regionIDs are {(1, 2, 4, 5, 6, 7), ((76,0), (76,28), (76,52), (76,56), (76,64), (76,68)), ((4×28), (4×4), (4×4), (4×8), (4×4), (4×12)), (False, True, True, False, True, False)}, and {(2, 3, 4, 6), ((76,28),(76,32),(76,52),(76,64)), (((4×4), (4×20), (4×4), (4×4))), (True, False, True, True)} accordingly.

We want to emphasize that RASTER enables users to pass region semantics via the existing netCDF interface `nc_def_var_chunking (int ncid, int varid, size_t* chunksize)`. Instead of passing an array of fixed-size chunks in `chunksize`, users can send a collection of `regionID` arrays in `chunksize`. RASTER traps the parameter and forwards it to Region Mesh Builder if `chunksize` carries values with a much smaller number of digits (e.g., 65536 as a regular `chunksize` vs. 7 as a `regionID`). In this way, Region Mesh Builder can build region mesh structures for RASTER variant-length chunks under the guidance of region information without any code modification in applications.

3.3.2 Index Construction. This subsection discusses how *Chunk Index Manager* constructs index tables to map regionIDs with RASTER chunks. Given the mesh structure `{chunkID, offset, length, mixed}` as a guide, Chunk Index Manager constructs a KV-based index table to map chunks to their belonging region. The key of the table is `regionID`, and the value is a chunk information list, which contains all the corresponding RASTER chunks associated with the `regionID`. For example, given the mesh structure of the last row in Fig. 3, Chunk Index Manager constructs an index table presented in Table 1.

3.3.3 Data Writing. This subsection discusses the RASTER data write procedure. *Metadata Writer* in *Metadata Handler* creates and assigns an HDF5 group structure for each region to store its index table and HDF5/netCDF attributes in the metadata of a file. *Chunk Data Writer* in *Chunk Data Handler* then writes data in a set of RASTER chunks guided by the index table. Upon handling mixed chunks that contain data from multiple regions, the Chunk Data Handler writes all of them in a separate group named `Mixed_Chunks`. Instead of writing the mixed chunks in every corresponding region group, RASTER writes them only once so that whichever refers to them will access the shared copy. In this way, RASTER avoids the redundant mixed chunks and the potential data inconsistency.

Fig. 5 illustrates the data write procedure: (1) RASTER traps the user's write request. (2) The region mask M is processed by Mesh Builder, such that a variant-length chunk

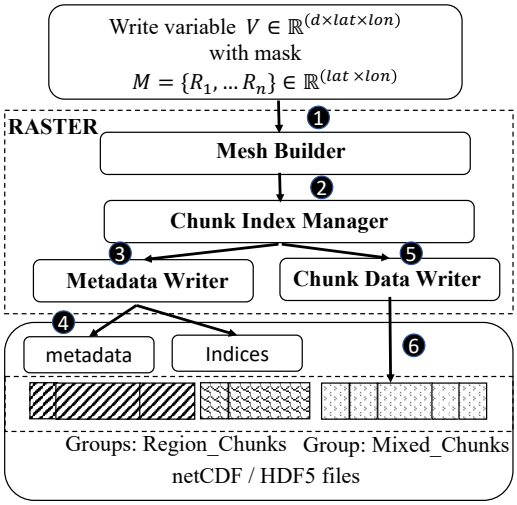


Figure 5: Write procedure of RASTER library

mesh is created. (3) Chunk Index Manager aggregates chunks and constructs the index table. (4) Metadata Writer creates region groups and writes the index table along with chunking information lists. (5) Chunk Writer receives the data chunks. (6) Chunk Writer performs a contiguous file write for each chunk.

3.3.4 Data Acquisition. Finally, we demonstrate how RASTER reads data. Besides the existing HDF5/netCDF interfaces that read data all at once or by *hyper-slab*, which is a regular-shaped array, RASTER can acquire data by irregular-shaped regions. RASTER first loads the metadata of a file to retrieve the region index table by *Medadata Reader* in Metadata Handler, then performs data acquisition by *Chunk Data Reader* in Chunk Data Handler. The datapath of read is illustrated in Fig. 6: (1) When the user acquires data in `region_i` (where i is the regionID), this read request is first trapped by the Chunk Index Manager. (2) Chunk Index Manager looks up the index table by the given `regionID` and then invokes the Metadata Handler to read the corresponding chunk info list. (3) Metadata Reader reads the chunk info list of the desired region. (4) Metadata Reader sends these chunk indices to the Chunk Data Reader. (5) Chunk Data Reader performs contiguous read operations to acquire these chunks. (6) Chunk Data Reader copies desired chunks to the user's buffer.

RASTER also supports traditional full-variable or hyper-slab reads. But the difference is we use variant-length chunks to reduce the number of fragmented chunks. This design reduces software overhead because of fewer function calls for reading chunks. In most cases, our data reading process is faster than the original, but some corner cases need to be improved. For example, reading a very thin hyper-slab

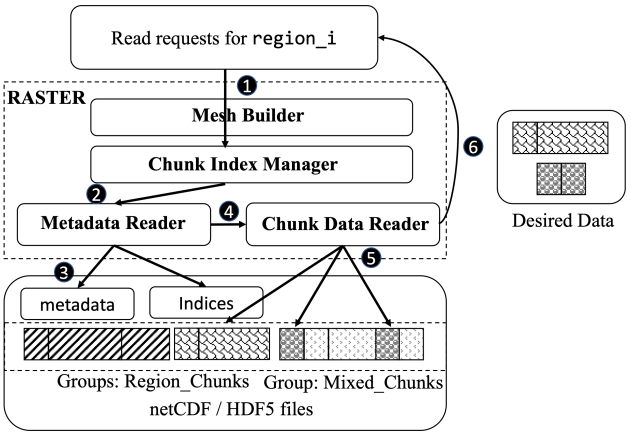


Figure 6: Read procedure of RASTER library

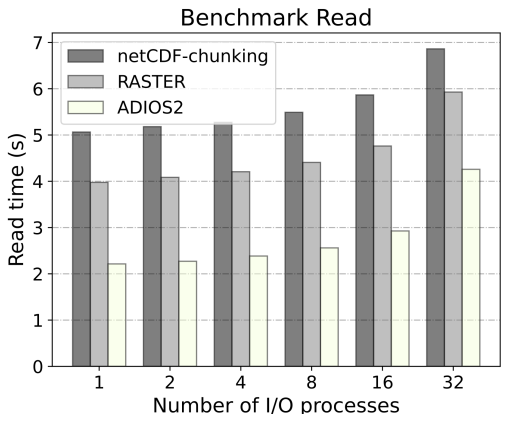


Figure 7: Read benchmarks on the AEP server

that is perpendicular to the merging direction may cause performance degradation. This is because the actual read size of RASTER will be larger than the netCDF chunking layout because our chunks are longer than the fixed-size chunks along the merging direction.

4 EVALUATION

4.1 Experimental Setup

We seek to answer the following questions: (1) How does RASTER perform read/write under benchmarks? (§ 4.2) (2) What is the data acquisition performance of RASTER in a real-world scientific simulation dataset? (§ 4.3) (3) Is RASTER scalable? (§ **??**)

Very often, self-describing files are produced by massive parallel scientific applications, then read and analyzed by data analysis programs. Since data may be produced by various applications, their scales, region shapes, and read patterns are also varied. The evaluation experiments should be well-designed to cover such cases. Therefore, we evaluate RASTER using I/O benchmarks and various real-world application datasets with a wide range of features and resolutions. Different from the state-of-the-art parallel scientific I/O libraries (e.g., ADIOS2 [26]) that require code modification for the dedicated APIs, RASTER runs via the unified interface with a few changes (usually within ten lines of code). RASTER supports data parallelism without deprecated PnetCDF interfaces because RASTER shares the parallel HDF5 interfaces.

We first evaluated RASTER on a single-node AEP server featuring two Intel® Xeon® Gold 6240M 2.60GHz 36-core CPUs, 12 × 16GB Hynix DDR4-2933MHz main memories, 12 × 128GB Intel® Optane™ DC persistent memory, running Linux 5.1.0.

We then performed large-scale evaluations of RASTER on a production cluster. The cluster consists of four Huawei® OceanStor Pacific™ 9550 storage nodes. Each node equips dual Intel® Xeon® Gold 6138T CPU @ 2.00GHz with 20 cores/40 threads, 384GB memory, and Mellanox Technologies MT27800 Family IB card. Each node equips 20×8TB HDDs to form a Huawei® Pacific™ distributed file system. We can not run ADIOS2 tests on the Huawei® cluster by the time of the submission as the verification of ADIOS2 is still undergoing.

Finally, we performed large-scale evaluations of RASTER on Sunway TaihuLight. We deployed RASTER on 256 Sunway compute nodes equipped with Shenwei SW26010 CPUs. Each SW26010 processor includes four core groups interconnecting with a network-on-chip and 32G DDR3 memory. A Lustre parallel file system runs on top of the storage system, which comprises one MDT and 12 OSTs. Each OST has a storage capacity of 3.1TB, leading to a total storage capacity of 3.6PB. We can not operate ADIOS2 tests by the time of submission because ADIOS2 is incompatible with the Shenwei SW architecture. It takes additional efforts to port ADIOS2 for Sunway TaihuLight.

4.2 Evaluation on I/O benchmarks

In this subsection, we evaluate the random I/O access performance of RASTER using the `H5perf` and `nc_perf` benchmark tools from HDF5 library. RASTER treats the whole dataset as a single region in this scenario because random accesses do not encounter any regional semantics. The benchmark evaluations show how RASTER performs under the worst case where no I/O access patterns can be referenced. Figs. 7 and 8 show that RASTER outperforms the netCDF chunking mechanism and achieves up to 1.16× speedup due to its better I/O contiguity and less overhead from the stacked software layout.

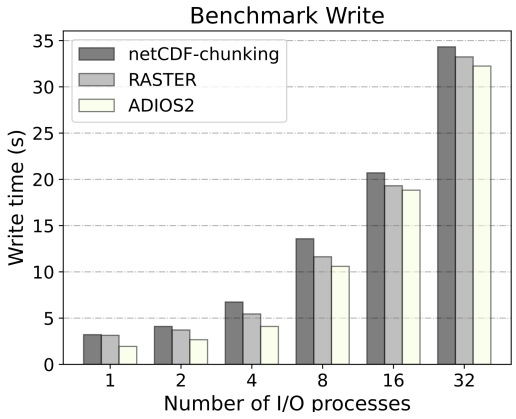


Figure 8: Write benchmarks on the AEP server

Besides the netCDF-chunking mechanism, we compare RASTER with a state-of-the-art scientific I/O library called ADIOS2 [19, 26]. We tuned parameters like `chunksize` and `H5ChunkDim` for ADIOS2 to observe its optimal performance as the baseline. We operated a group of weak scaling benchmark tests with different dataset scales on NVRAM (shown in Figs. 7 and 8). The benchmark sets the number of processes from 1 to 16 to read separate datasets. The per-process data size is set at 2.2 GB, and the overall dataset scale is over 72 GB. For read performance, we observed that RASTER achieves at most 15.8% faster than netCDF, while ADIOS2 achieves up to 61.1% speedup under the all-access pattern at 32 processes. We also evaluated the write performance of RASTER and ADIOS2, which shows RASTER achieves up to 23.7% optimization to ordinary netCDF, and ADIOS2 is at most 65.6%. ADIOS2 achieves better performance with a more efficient interface that bypasses the netCDF I/O routine in addition to specially tuned parameters. It either requires code modification or extra efforts to optimize the performance.

The benchmark is to evaluate I/O performance on random accesses without considering region semantics. Random access is the worst case for RASTER as its variant-length chunking and contiguous access features are invalid. However, we observe that RASTER outperforms ordinary netCDF solutions and performs closer to the state-of-the-art ADIOS2. It proves that RASTER reduces software overhead compared to netCDF and shows its potential in handling random access.

Real-world applications give a good account of RASTER as they usually access data with region semantics. We want to emphasize that the term "region" is not unique to geographical ideas. There are ample scenarios for RASTER's abilities as long as applications use identifiers to define the shape of the data. RASTER manages the data in varied-length chunks as long as applications use identifiers to define the shape of the data. In such a way, RASTER reduces the amount of data reads, and the data reduction rate is:

$$\text{Data Reduction Rate} = \frac{S_{BB}}{\sum_{i=1}^{N}(S_{RC,i}) + \sum_{i=1}^{M}(S_{MC,i})} \quad (1)$$

where N and M denote the number of regional chunks and related mixed chunks, respectively. $S_{RC,i}$ and $S_{MC,i}$ represent the i^{th} regional and mixed chunk size, respectively. S_{BB} is the bounding box size in the ordinary data acquisition process.

Note that RASTER also takes advantage of sequential accesses because it aggregates contiguous data that belongs to the same region into one chunk, hence the I/O speedup is:

$$\begin{aligned}\text{I/O Speedup} &= \frac{S_{BB}/BW_{rand}}{(\sum_{i=1}^{N}(S_{RC,i}) + \sum_{i=1}^{M}(S_{MC,i}))/BW_{seq}} \\ &= \text{Data Reduction Rate} \times \frac{BW_{seq}}{BW_{rand}}\end{aligned} \quad (2)$$

We can observe from Eq. 2 that the performance gain from RASTER is two-folded: 1. it reduces the amount of data read, and 2. it exaggerates the benefits of sequential read by aggregating data with the same region semantic into a contiguous one.

4.3 Evaluation on CESM dataset

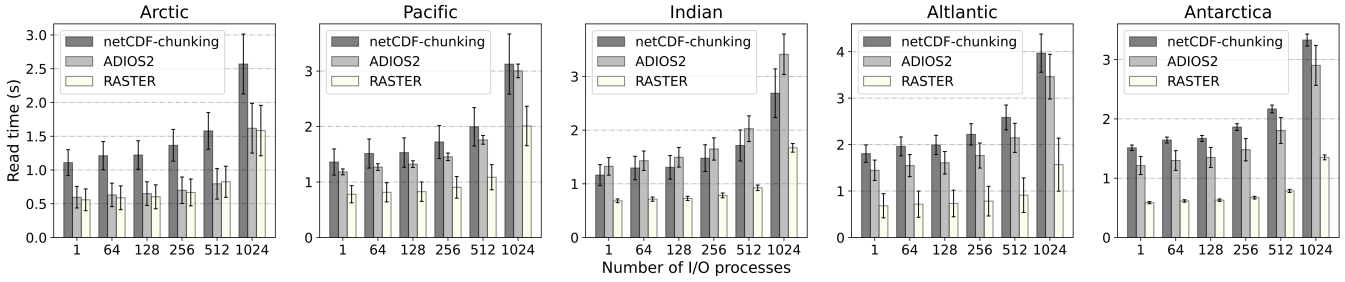
We choose the Community Earth System Model [1] (CESM) to evaluate the performance of RASTER under real-world applications. We collected the CESM dataset from the College of Ocean and Earth Sciences, Xiamen University. The dataset, generated by a 20-year global ocean simulation, contains more than 100 variables, while the domain researchers only pay attention to one-tenth of the variables. Each of the variables is a 4-dimensional array corresponding to time

Table 2: Region division of CESM global dataset

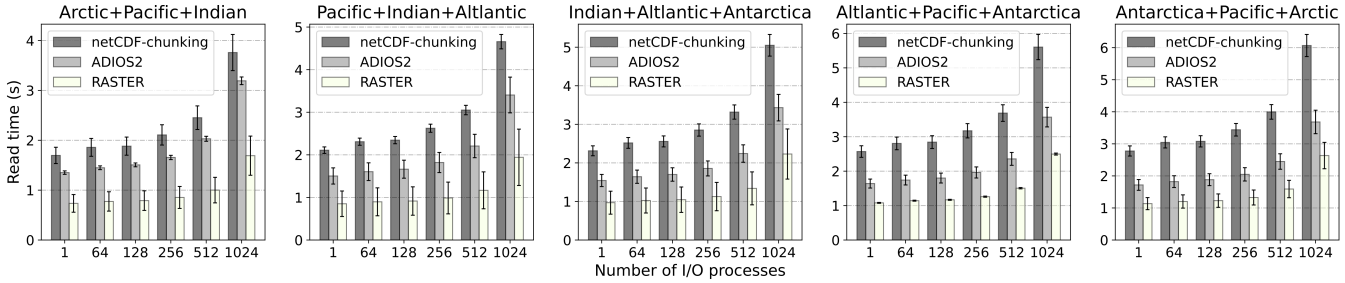
MASK_ID	Name	Percentage of Total	Valid
≤0	Lands	~29.8%	False
1	Antarctica	~17.7%	True
2	Pacific	~23.9%	True
3	Indian	~8.64%	True
6	Atlantic	~12.1%	True
10	Arctic	~4.12%	True

Table 3: Memory consumption saved by RASTER

Region	Arctic	Pacific	Indian	Atlantic	Antarctica
Memory Saved	11.22%	34.04%	61.00%	406.76%	23.14%
Region	Arctic+ Pacific+ Indian	Pacific+ Indian+ Atlantic	Indian+ Atlantic+ Antarctica	Atlantic+ Pacific+ Antarctica	Antarctica+ Pacific+ Arctic
Memory Saved	54.87%	49.93%	157.33%	67.99%	91.01%

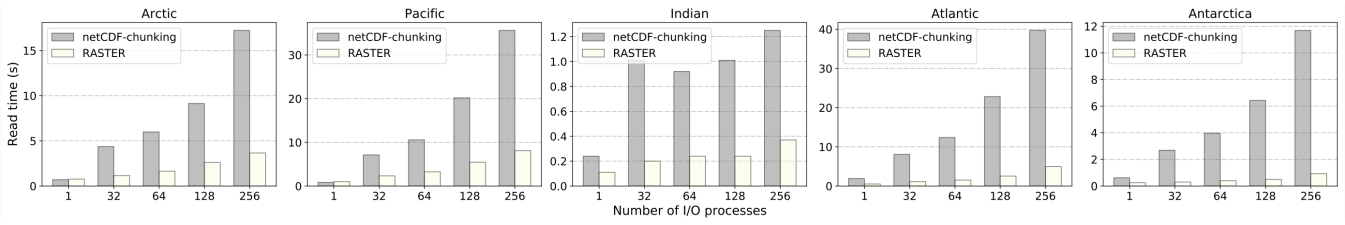


(a) Single-region read, number of regions=1

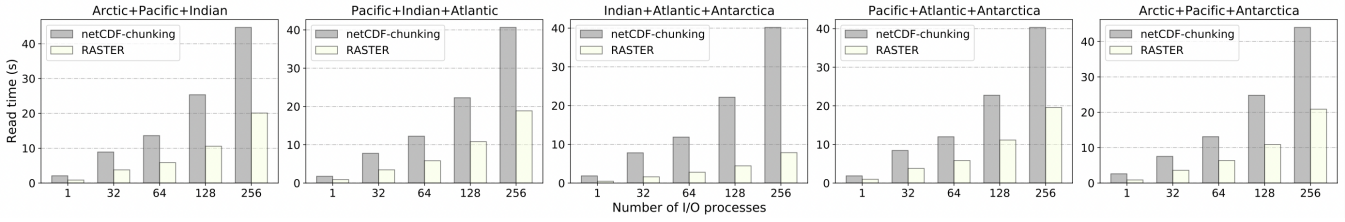


(b) Mixed (dis-contiguous+adjacent) multi-region read, number of regions=3

Figure 9: Read performance of CESM global marine dataset using different patterns



(a) Large-scale single-region read, number of regions=1



(b) Large-scale mixed (dis-contiguous+adjacent) multi-region read, number of regions=3

Figure 10: Large-scale read performance of CESM global marine dataset on Huawei® Pacific™ clusters

(T), depth to the ocean surface (*depth*), latitude (*nlat*), and longitude (*nlon*), the region mask is $nlat \times nlon$ that marks different continents and oceans as distinct integers. Only oceans and Antarctica are simulated, and values in positions corresponding to other lands are filled with invalid values (*FLOAT_MAX*) and will never be used. Table. 2 shows the

detailed region division of these datasets, note that in this division, south of the Tropic of Capricorn is almost all marked as Antarctica, which makes it larger than the generally accepted area. Each variable is in a separate file, with a file size is about 2.2 GB.

We use this dataset for a coarse-grained evaluation as it only has 6 regions representing land and five oceans. We first conducted experiments on the single-node AEP server. We choose 32 variables, the size of each of which is 2.2 GB, and use MPI for parallel data analysis scenarios.

Single Region Access: Fig. 9(a) presents read time comparison between RASTER and netCDF-chunking to access single-region data, including the Pacific, Arctic, Atlantic, Indian Ocean, and Antarctica. We omitted the testing results of sequential netCDF because they are much worse than the netCDF-chunking layout. Besides, modern scientific applications rarely use the sequential netCDF layout. We initialized the chunk shape to (1, 50, 50, 50) for both RASTER and netCDF.

We observe from Fig. 9(a) that RASTER achieves 2.18× speedup compared to netCDF-chunking, and 1.80× speedup compared to ADIOS2, on average. This is because RASTER dynamically merges the chunks with the same regionID, which introduces better data contiguity. We also observe that RASTER is 2.83× faster than netCDF-chunking and 2.36× faster than ADIOS2 when reading the Atlantic data. This is because the data lies at the extremes of the dataset requiring much more hyper-slabs to read in netCDF-chunking. On the contrary, RASTER only reads the chunks with `Atlantic` and mixed chunks, hence reducing read time by eliminating the number of unnecessary data read.

Dis-contiguous and Multi-region Access: We then evaluated RASTER's performance in reading multiple regions that consist of adjacent and disjunct regions. We observe from Fig. 9(b) that RASTER performs on average 2.45× faster than netCDF-chunking and 1.67× better than ADIOS2. We observe the peak performance gain from the combination of the Atlantic, Indian, and Antarctica regions, RASTER reaches 2.66× speedup when comparing with netCDF-chunking and 2.02× speedup when comparing with ADIOS2. This is because RASTER reads data in continuous variant-length chunks, which reduces the random I/O access latency. It shows that under complicated semantics, RASTER can accurately target the corresponding chunks to the given regionIDs and reduce the number of unnecessary data accesses.

Memory utilization: As discussed in Section 2.2, reading regional data by the original hyper-slab interface requires more memory capacity because the hyper-slab brings both required and unnecessary data into memory at the same time. RASTER, on the other hand, requires less memory capacity as it only retrieves required data. We admit that RASTER can not occupy as less memory as the theoretical value for the regional data because of the mixed chunks, which consist of data that belongs to other regions. But RASTER can still consume much less memory compared to the hyper-slab solution. Table. 3 shows how much more memory capacity is required by netCDF-chunking than that of RASTER in all the data access patterns discussed above. In this table, *Reg.* denotes the regionID and *Mem.Amp.* represents the amplification ratio of memory usage of netCDF-chunking to that of RASTER. The higher the percentile value, the more memory is required by netCDF-chunking than RASTER, a.k.a the more memory usage RASTER can reduce. We can find in the table that RASTER uses less memory in all the comparison cases and achieves up to 400% better than netCDF-chunking. We also can observe that the memory reduction shares a similar trend to the read size reduction, which proves the design goal of RASTER is achieved.

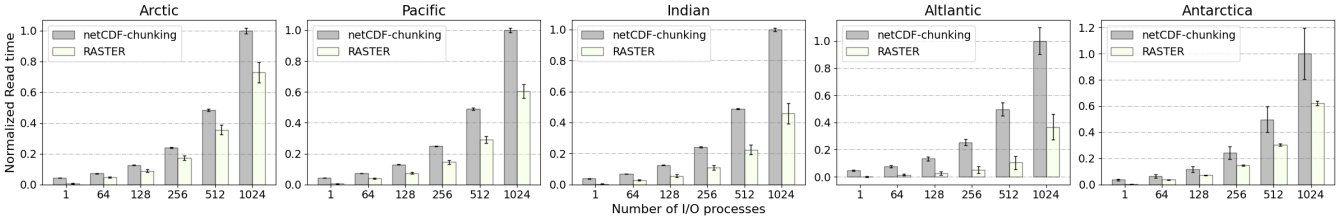
4.4 Evaluation on a production cloud cluster

We enlarged the CESM dataset to evaluate the scalability of RASTER on a production cluster by up-scaling the time dimension while holding the regional information to be the same. We assigned each process on the Huawei® Pacific™ cloud storage cluster to read a 4GB dataset. RASTER will handle 1TB of concurrent data read under the 256-process test case.

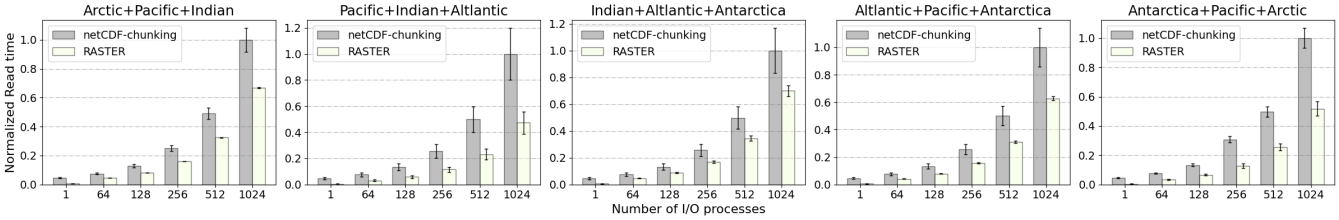
Fig. 10 shows that on all the test cases, RASTER outperforms the netCDF-chunking mechanism. In the single-region read cases, RASTER achieves 8.94× speedup when accessing the Atlantic dataset using 128 processes (shown in Fig. 10(a)). RASTER's advantages in data contiguity further reduce the number of random reads, hence eliminating a decent amount of communication overhead on distributed file systems. In the mixed multi-region read cases, RASTER gains a 5.11× speedup when accessing Indian+Atlantic+Antarctica ocean data using 256 processes. It indicates that RASTER has the potential to transform scattered data access pattern into a more contiguous one and optimize the remote I/O performance on distributed file systems.

4.5 Evaluation on Sunway TaihuLight

We implemented RASTER on the Sunway TaihuLight supercomputing machine to evaluate the scalability of RASTER. We enlarged the CESM dataset by up-scaling the time dimension while keeping the regional information constant. Each process on the TaihuLight cluster was assigned to read a 3GB dataset. In the 1024-process test case, RASTER can handle 3TB of concurrent data read. We can not compare RASTER with the state-of-the-art ADIOS2 on TaihuLight by the time of submission because ADIOS2 is incompatible with the Shenwei SW architecture.



(a) Large-scale single-region read, number of regions=1



(b) Large-scale mixed (dis-contiguous+adjacent) multi-region read, number of regions=3

Figure 11: Large-scale read performance of CESM global marine dataset on Sunway TaihuLight

Fig. 11 shows the results in the normalized form. We can observe that RASTER outperforms the netCDF-chunking mechanism in all the test cases. RASTER achieves 5.41× speedup when accessing the Atlantic dataset using 64 processes in the single-region case (shown in Fig. 11(a)). RASTER's advantages in data contiguity further reduce the number of random reads, eliminating the communication overhead on distributed file systems. RASTER gains a 2.4× speedup when accessing Antarctica+Pacific+Arctic ocean data using 256 processes in the multi-region case (shown in Fig. 11(b)). The results indicate that RASTER can transform scattered data access patterns into a more contiguous one and optimize the remote I/O performance on file systems. Considering that TaihuLight is a production system, our evaluation results could be even better without the I/O contentions by other applications running on it.

4.6 Evaluation on WRF Datatsets

We choose the Weather Research and Forecasting Model (WRF) as the second real-world application to evaluate the performance of RASTER. Different from CESM, WRF stores more regions for administrative divisions. WRF serves as a finer-grained test case for RASTER. We placed the evaluation results of WRF in the appendix due to the page limits.

The WRF dataset is collected from the Hubei Meteorological Center, China, which provides a finer-grained provincial rainwater investigation (denoted as rainwater). A 2-dimensional *REGIONMASK* is applied to divide data into provinces, with each province holding a unique region ID. In this experiment, RASTER reads the rainfall variables of a given province. The file size is 250MB per variable because of the high resolution. The total size of the dataset with all variables is 86.4GB. In this group of experiments, we chose 4 typical provinces in China: Hubei, Guangdong, Sichuan, and Shandong, for evaluation.

On the local AEP server, we evaluate regional read performance using 1 to 32 MPI processes. As demonstrated in Fig.12, we can observe that RASTER also outperforms the original for each region access pattern. Typically, RASTER is 2.03× faster than ordinary netCDF and 1.94× faster than ADIOS2 when reading data in Sichuan province using 32 processes. On average, RASTER performs a 1.89× speedup to netCDF and 1.55× speedup to ADIOS2.

On the Huawei® Pacific™ storage cluster, we set 1 to 256 processes to read this dataset. To show the scalability of RASTER on large-scale distributed storage scenarios, we enlarge this dataset by up-scaling the time dimension while holding the regional information to be the same. In the experiments running on the Huawei Cloud Storage cluster, each process reads a 4GB dataset, hence the dataset size using 256 processes is over 1TB. These large-scale WRF tests also show that RASTER outperforms the netCDF-chunking scheme. We observe from Fig. 13 that RASTER gains on average 8.34× speedup and achieves 16.59× acceleration when reading data in Guangdong province using 128 processes. This is because the provincial region division is complicated. RASTER identifies a finer border of the adjacent provinces and, hence reads less data than the ordinary netCDF-chunking technique.

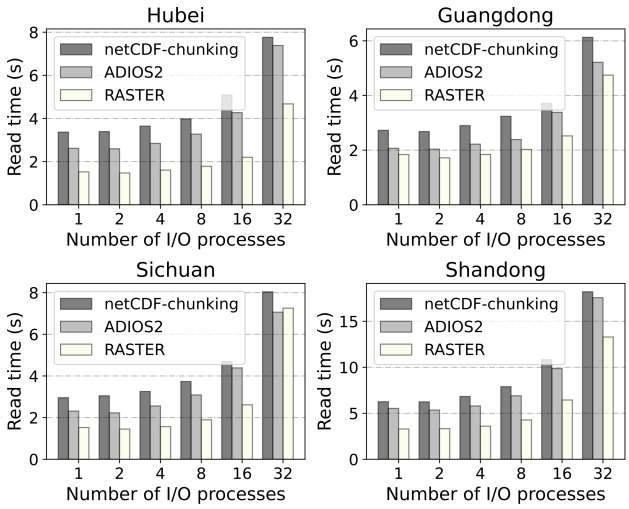


Figure 12: Read performance of WRF (rain water) on the AEP server

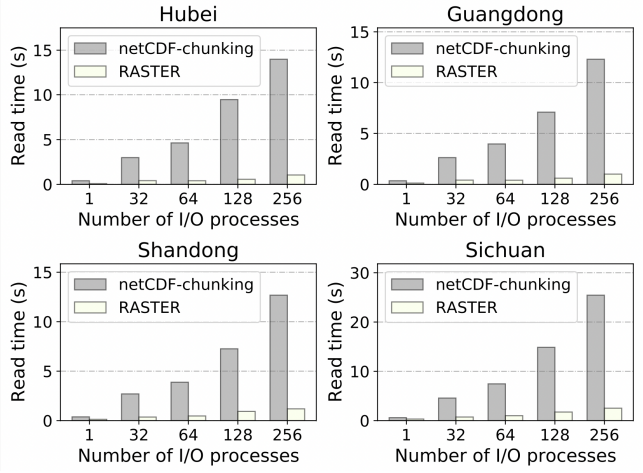


Figure 13: Read performance of WRF (rain water) on Huawei® Pacific™ clusters

From the evaluations of WRF, we can see that RASTER better fits the applications with finer-grained region semantics. It is because compared to ordinary netCDF-chunking, RASTER reads much less data in a contiguous way. The combined attempts on data reduction and sequential access help RASTER optimize the data acquisition performance. Besides, the reduced software stacks of RASTER further optimize the data access latency.

Through the evaluation in Section 4.2, we can confirm that RASTER has been validated on our x86-based servers with NVMe devices. The comparisons of RASTER and ADIOS in Figs. 7-9 are capable of validations from both benchmark and real-world tests. Using Chinese hardware and software (i.e. TaihuLight and Huawei clusters) is to evaluate the compatibility and scalability of RASTER on architectures that are different from x86. We are open to renting cloud storage instances from AWS or Alibaba to further demonstrate RASTER's scalability performance on larger x86-based platforms.

5 LESSONS LEARNED

I/O becomes a critical problem for many scientific applications which rely on hierarchical self-describing files. Our study shows that an oceanographic correlation analysis program takes more than 60% time to perform data I/O. The data management has to be carefully addressed to improve I/O performance. Besides this, there are a few lessons we learned from this project.

A semantic-driven data organization mechanism is crucial to self-describing data analysis: Traditionally, scientific data in self-describing formats are usually array-oriented, with its semantics and possible read patterns lost on the file level. However, most applications produce data with complex and irregular regional semantics. It can not only be spatial regions in geographic data but can also be time periods and any arbitrary user-defined characteristics such as elevation or urban/rural. RASTER can greatly boost regional data acquisition by its region-aware partitioning and grouping mechanism and can be integrated into applications of other domains such as fluid dynamics and high-energy physics (e.g., HACC, Cheetah, and OnDA). Moreover, RASTER achieves data re-organization in a reasonable time, which is much less than data I/O costs, and it may be optimized in future works.

Software overhead becomes dominant on fast storage devices: We observed that when acquiring data in a self-describing dataset, employing high-bandwidth I/O devices like NVRAMs cannot enhance data processing performance as expected. This is because software latency becomes a dominant factor that determines overall performance. In self-describing data processing, in particular, chunk retrieval and useless memory copy performed by the HDF5 library become the bottleneck when I/O is faster. RASTER reduces such overhead through its mesh-building mechanism, which reduces the number of chunks and improves data continuity so that chunk retrieval and memory copy can be accelerated.

Effective memory utilization is critical to data processing: We found memory amplification when users attempted to process irregular regions of the netCDF dataset. It is because the unrelated data is read into memory as well.

With data volume growing, effective memory utilization becomes an urgent problem. RASTER optimizes the memory utilization by only retrieving chunks that are needed.

6 RELATED WORKS

To the best of our knowledge, this research is the first work that focuses on optimizing the data layout of self-describing files with complex regional semantics. Therefore, we only found a few closely related works in the literature. This section summarizes commonly used I/O middlewares and existing optimizations for self-describing files, introduces popular spatial and temporal data storage systems, and proposes some HPC applications that may benefit from RASTER.

Existing I/O middlewares and optimizations: Although several I/O middlewares and libraries are proposed to enhance scientific data processing performance [12, 23, 28, 32, 38], they have limits in optimizing complex regional data acquisitions performance and memory utilization. PLFS [9, 28] and ADIOS/ADIOS2 [19, 26] provide a lower level optimization for scientific data writing but contribute less to data acquisition performance. ROMIO [33] enables users to perform high-performance I/O for non-contiguous requests but cannot be applied to self-describing scientific datasets without a lot of code modification. BORA [38] reorganizes ROS bag files by semantics yet cannot be applied to the self-describing dataset, either. Topology-aware data movement [34] improves data locality for HDF5 I/O on supercomputers while not considering data semantics and possible accessing patterns. Existing data re-organization methods [32, 35] only focus on read-write disorder problems, such as writing by rows but accessing by columns, but not address such complex regional data accessing inefficiency. Table. 4 compares I/O approaches for scientific data.

Spatial and temporal databases: Several data management systems have been proposed to access spatial or temporal scientific data efficiently. InfluxDB [29] and BtrDB [5] can manage data in time series efficiently, yet not designed for scientific array-based datasets. Spatial databases [8, 41] provide a way to manage spatial-temporal data, but they only implement SQL-like interfaces, with their data stored in database formats. As a result, they cannot be applied to self-describing files or support more complex data access requests.

Scientific applications with HDF5: Scientific applications running on HPC systems tend to write and read data via HDF5/netCDF and their variants. HACC [22] simulates sky surveys on supercomputing architectures, AMReX [39] runs adaptive mesh refinement massively in parallel, and both of them generate datasets in HDF5 or a variant called H5M. OnDA [27] and Cheetah [7] are designed for high-energy physics analysis, which requires a high-performance partial data acquisition from enormous data in self-describing formats. LAMMPS simulates molecular dynamics and records atom positions in the HDF5 variant called H5MD. CESM [1] and WRF [4] are commonly used in climatology, meteorology, and oceanography and produce heterogeneous data with complex region information in netCDF/HDF5 format. RASTER can assist many of them for better I/O performance and memory utilization.

Table 4: I/O Middleware System Comparison

	Inter-position	Usage	Regional Access	App. Modification
PLFS Plugin	FUSE Library	Scientific Data	No	No
HDF5 Re-organization	Library	Scientific Data	No	No
Spatial Database	Outside	Spatial Data	Partial	Heavily
ADIOS/ADIOS2	Library	Scientific Data	No	Heavily
RASTER	Library	Scientific Data	Yes	Barely No

7 CONCLUSION

Rapidly growing data volume and the complexity of data accessing patterns make self-describing scientific data management more and more challenging. However, equipping next-generation I/O devices can not improve the performance as expected, which is mainly attributed to software overhead and ineffective use of I/O and memory. To improve the read performance of complex regions while reducing software latency, we proposed an I/O optimizing library for self-describing files, called RASTER. Then, we integrate it into a server with persistent memories and evaluate its performance on different devices using benchmarks. Further, we evaluate RASTER in terms of regional data acquisition time under real-world applications with different scales and data access patterns. Our experimental results demonstrate that RASTER can enhance I/O bandwidth and significantly improve data acquisition performance. Besides, it reduces software overhead and memory amplification through its novel design.

As an ongoing project, we plan to extend RASTER with nestable region mesh builders to support 3-dimensional or hyper-dimensional region masks. We also plan to integrate RASTER into more scientific applications with more complex datasets.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their tremendous feedback and comments, which have substantially improved the content and presentation of this paper. We thank our shepherd Dev Purandare for advice on revising the paper. We thank Shanlin Wang from Xiamen University for her deep domain knowledge on CESM research and Wei Cao from Hubei Meteorological Service for his invaluable feedback on this work. We also thank Huawei Technologies Co., Ltd.'s data storage product line for providing support. Shu Yin's research is supported by National Key Research and Development Program of China under 2023YFB4502902.

REFERENCES

[1] 2022. Community Earth System Model, CESM. https://www.cesm.ucar.edu/. Accessed: 2022-1-1.

[2] 2022. GROMACS. https://www.gromacs.org/. Accessed: 2022-9-20.

[3] 2022. Large-scale Atomic/Molecular Massively Parallel Simulator, LAMMPS. https://www.lammps.org/. Accessed: 2022-1-1.

[4] 2022. Weather Research and Forecasting model, WRF. https://ral.ucar.edu/solutions/products/weather-research-and-forecasting-model-wrf. Accessed: 2022-1-1.

[5] Michael P Andersen and David E Culler. 016. Btrdb: Optimizing storage system design for timeseries processing. In *14th USENIX Conference on File and Storage Technologies (FAST) 16)*. 39–52.

[6] Paul Bartholomew, Georgios Deskos, Ricardo AS Frantz, Felipe N Schuch, Eric Lamballais, and Sylvain Laizet. 2020. Xcompact3D: An open-source framework for solving turbulence problems on a Cartesian mesh. *SoftwareX* 12 (2020), 100550.

[7] Anton Barty, Richard A Kirian, Filipe RNC Maia, Max Hantke, Chun Hong Yoon, Thomas A White, and Henry Chapman. 2014. Cheetah: software for high-throughput reduction and analysis of serial femtosecond X-ray diffraction data. *Journal of applied crystallography* 47, 3 (2014), 1118–1131.

[8] Peter Baumann. 2014. Rasdaman: Array databases boost spatio-temporal analytics. In *2014 Fifth International Conference on Computing for Geospatial Research and Application*. IEEE, 54–54.

[9] John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. 2009. PLFS: a checkpoint filesystem for parallel applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. IEEE, 1–12.

[10] Spyros Blanas, Kesheng Wu, Surendra Byna, Bin Dong, and Arie Shoshani. 2014. Parallel data analysis directly on scientific file formats. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 385–396.

[11] Jerry Chou, Kesheng Wu, et al. 2011. Fastquery: A parallel indexing system for scientific data. In *2011 IEEE International Conference on Cluster Computing*. IEEE, 455–464.

[12] Bin Dong, Surendra Byna, and Kesheng Wu. 2013. Expediting scientific data analysis with reorganization of data. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 1–8.

[13] Bin Dong, Surendra Byna, and Kesheng Wu. 2014. Parallel query evaluation as a scientific data service. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 194–202.

[14] Bin Dong, Surendra Byna, and Kesheng Wu. 2015. Spatially clustered join on heterogeneous scientific data sets. In *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 371–380.

[15] F. Alted, M. Durant, S. Hoyer, J. Kirkham, A. Miles, M. Ratsimbazafy, M. Rocklin, V. Schut, A. Scopatz, and P. Goel. 2022. Zarr. https://zarr.readthedocs.io/. Accessed: 2021-12-31.

[16] Mike Folk, Albert Cheng, and Kim Yates. 1999. HDF5: A file format and I/O library for high performance computing applications. In *Proceedings of supercomputing*, Vol. 99. 5–33.

[17] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. 36–47.

[18] Kui Gao, Wei-keng Liao, Alok Choudhary, Robert Ross, and Robert Latham. 2009. Combining I/O operations for multiple array variables in parallel netCDF. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 1–10.

[19] William F. Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, Axel Huebl, Mark Kim, James Kress, Tahsin Kurc, Qing Liu, Jeremy Logan, Kshitij Mehta, George Ostrouchov, Manish Parashar, Franz Poeschel, David Pugmire, Eric Suchyta, Keichi Takahashi, Nick Thompson, Seiji Tsutsumi, Lipeng Wan, Matthew Wolf, Kesheng Wu, and Scott Klasky. 2020. ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management. *SoftwareX* 12 (2020), 100561. https://doi.org/10.1016/j.softx.2020.100561

[20] Luke Gosink, John Shalf, Kurt Stockinger, Kesheng Wu, and Wes Bethel. 2006. HDF5-FastQuery: Accelerating complex queries on HDF datasets using fast bitmap indices. In *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*. IEEE, 149–158.

[21] Perry Greenfield, Michael Droettboom, and Erik Bray. 2015. ASDF: A new data format for astronomy. *Astronomy and Computing* 12 (2015), 240–251.

[22] Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, Tom Peterka, et al. 2016. HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy* 42 (2016), 49–65.

[23] Shuibing He and Xian-He Sun. 2018. A cost-effective distribution-aware data replication scheme for parallel I/O systems. *IEEE Trans. Comput.* 67, 10 (2018), 1374–1387.

[24] Xingying Huang and Paul A Ullrich. 2016. Irrigation impacts on California's climate with the variable-resolution CESM. *Journal of Advances in Modeling Earth Systems* 8, 3 (2016), 1151–1163.

[25] Jianwei Li, Wei-keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Robert Latham, Andrew Siegel, Brad Gallagher, and Michael Zingale. 2003. Parallel netCDF: A high-performance scientific I/O interface. In *SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. IEEE, 39–39.

[26] Jay F Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. 2008. 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*. 15–24.

[27] Valerio Mariani, Andrew Morgan, Chun Hong Yoon, Thomas J Lane, Thomas A White, Christopher O'Grady, Manuela Kuhn, Steve Aplin, Jason Koglin, Anton Barty, et al. 2016. OnDA: online data analysis and feedback for serial X-ray imaging. *Journal of applied crystallography* 49, 3 (2016), 1073–1080.

[28] Kshitij Mehta, John Bent, Aaron Torres, Gary Grider, and Edgar Gabriel. 2012. A plugin for hdf5 using plfs for improved i/o performance and semantic analysis. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 746–752.

[29] Syeda Noor Zehra Naqvi, Sofia Yfantidou, and Esteban Zimányi. 2017. Time series databases and influxdb. *Studienarbeit, Université Libre de Bruxelles* 12 (2017).

[30] Ivy B Peng, Maya B Gokhale, and Eric W Green. 2019. System evaluation of the intel optane byte-addressable nvm. In *Proceedings of the International Symposium on Memory Systems*. 304–315.

[31] Russ Rew and Glenn Davis. 1990. NetCDF: an interface for scientific data access. *IEEE computer graphics and applications* 10, 4 (1990), 76–82.

[32] Houjun Tang, Suren Byna, Steve Harenberg, Xiaocheng Zou, Wenzhao Zhang, Kesheng Wu, Bin Dong, Oliver Rubel, Kristofer Bouchard, Scott Klasky, et al. 2016. Usage pattern-driven dynamic data layout reorganization. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 356–365.

[33] Rajeev Thakur, William Gropp, and Ewing Lusk. 1999. Data sieving and collective I/O in ROMIO. In *Proceedings. Frontiers' 99. Seventh Symposium on the Frontiers of Massively Parallel Computation*. IEEE, 182–189.

[34] Venkatram Vishwanath, Mark Hereld, Vitali Morozov, and Michael E Papka. 2011. Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–11.

[35] Lipeng Wan, Axel Huebl, Junmin Gu, Franz Poeschel, Ana Gainaru, Ruonan Wang, Jieyang Chen, Xin Liang, Dmitry Ganyushin, Todd Munson, et al. 2021. Improving I/O Performance for Exascale Applications through Online Data Layout Reorganization. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2021), 878–890.

[36] Donald Carson Wells and Eric W Greisen. 1979. FITS-a flexible image transport system. In *Image Processing in Astronomy*. 445.

[37] Chenglai Wu, Xiaohong Liu, Zhaohui Lin, Alan M Rhoades, Paul A Ullrich, Colin M Zarzycki, Zheng Lu, and Stefan R Rahimi-Esfarjani. 2017. Exploring a variable-resolution approach for simulating regional climate in the Rocky Mountain region using the VR-CESM. *Journal of Geophysical Research: Atmospheres* 122, 20 (2017), 10–939.

[38] Jian Zhang, Tao Xie, Yuzhuo Jing, Yanjie Song, Guanzhou Hu, Si Chen, and Shu Yin. 2020. BORA: a bag optimizer for robotic analysis. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[39] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, et al. 2019. AMReX: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software* 4, 37 (2019), 1370–1370.

[40] Wei Zhang, Suren Byna, Houjun Tang, Brody Williams, and Yong Chen. 2019. Miqs: Metadata indexing and querying service for self-describing file formats. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–24.

[41] Xiaomin Zhang, Wei Song, and Liming Liu. 2014. An implementation approach to store GIS spatial data on NoSQL database. In *2014 22nd international conference on geoinformatics*. IEEE, 1–5.