SciDP: Support HPC and Big Data Applications via Integrated Scientific Data Processing

Kun Feng† Xian-He Sun‡ Xi Yang§ Shujia Zhou*
Illinois Institute of Technology†‡ Teradata Inc§ Northrop Grumman Information Technology*
kfeng1@hawk.iit.edu† sun@iit.edu‡ Xi.Yang@Teradata.com§ shujia.zhou@ngc.com*

***Abstract*—Modern *High Performance Computing* (HPC) applications, such as Earth science simulations, produce large amounts of data due to the surging of computing power, while big data applications have become more compute-intensive due to increasingly sophisticated analysis algorithms. The needs of both HPC and big data technologies for advanced HPC and big data applications create a demand for integrated system support. In this study, we introduce *Scientific Data Processing* (SciDP) to support both HPC and big data applications via integrated scientific data processing. SciDP can directly process scientific data stored on a *Parallel File System* (PFS), which is typically deployed in an HPC environment, in a big data programming environment running atop *Hadoop Distributed File System* (HDFS). SciDP seamlessly integrates PFS, HDFS, and the widely-used R data analysis system to support highly efficient processing of scientific data. It utilizes the merits of both PFS and HDFS for fast data transfer, overlaps computing with data accessing, and integrates R into the data transfer process. Experimental results show that SciDP accelerates analysis and visualization of a production *NASA Center for Climate Simulation* (NCCS) climate and weather application by 6x to 8x when compared to existing solutions.**

***Index Terms*—HPC, Big data, HPDA, Hadoop, R language**

I. Introduction

As the HPC community moves toward exascale, its data volume has increased dramatically [23]. Large scale scientific simulations, usually running in HPC environments with *Message Passing Interface* (MPI)-based computation and communication, produce tremendous amount of data and put a big burden on data analysis. In contrast, *Big Data* (BD) applications, such as data analysis and information retrieval, are usually carried out in distributed computing environments, such as Hadoop [7] and Spark [35]. Those environments are known for the usability, capability, and versatility in processing large amount of data. Using BD analysis applications can significantly enhance the data processing power, facilitate the exploration of huge data sets and lead to scientific discovery [11]. The combination of scientific computing, embodied in HPC, and BD environments, represented by *Apache Big Data Stack* (ABDS) [8], [31], for advanced data-intensive computing is impeccable. Therefore, there is an increasing demand to gain scientific insights using both the computation power of HPC environment and data analysis capabilities of the BD ecosystem.

There are three challenges keeping us from reaching the integration of the two different computing ecosystems. The first challenge is expensive data movement. The two computing ecosystems are built on top of different storage infrastructures. The conventional HPC systems use Parallel File System (PFS), such as Lustre [25] and GPFS [24]. Meanwhile, BD applications use distributed storage systems, such as HBase [1] and HDFS [28]. From an application'perspective, HPC and BD applications access data from their native data storage, PFS and HDFS, respectively, by default. However, the two kinds of underlying file systems have different design philosophies, deployment models and are specifically optimized for diverse targeted applications. Data movement between them is unacceptably expensive. The second challenge is mismatched data models. Scientific data is typically well structured, such as multi-dimensional arrays and stored in self-descriptive data formats (e.g. HDF5 [14] and netCDF [19]) for portability and extendibility. BD frameworks, on the other hand, excel in unstructured or semi-structured data (e.g. CSV files) processing, and do not natively support scientific data formats. The final challenge is incompatible programming user interfaces. Analysis performance optimization not only needs to consider the application execution time, but also the application development time, data movement time, data conversion, and any other execution preparation time. User interface is a significant factor of application development. Scientists prefer user-friendly and application-oriented programming tools and interfaces, such as MATLAB and R, over relatively new and system-originated environments, such as MapReduce and Spark. These traditional tool-based interfaces, however, are not capable of analyzing large volumes of data in a distributed environment. More importantly, they cannot be used to support an integrated environment to include both HPC and ABDS environments. Consequently, the unification at both the file system and programming interface level is essential to support an integrated environment.

Several works exist which tackle the above three technical barriers. From the HPC side, PFS connectors are designed to allow ABDS applications to access PFS without changing any application source code [17], [18], [26]. These solutions share the same design principle, which is to emulate HDFS APIs on top of PFS. For example, IBM developed a connector for GPFS, HDFS Transparency [17], suggesting a shared unified storage system for both HPC and ABDS applications. However, the shared PFS can only be configured and optimized for either PFS or ABDS applications, but not both. A compromised PFS with trade-off is therefore suboptimal. From the big data side, multiple works have been developed

to support scientific data formats in ABDS frameworks. SciHadoop and SciSpark, for instance, add netCDF processing support in Hadoop and Spark, respectively [9], [22]. However, these solutions target processing scientific data particularly on HDFS. The cost of slow data copy from PFS to HDFS is paid before data analysis can be carried out. It only can be used in off-line data analysis due to its slow data transfer, which greatly limits the productivity of scientific researches. PortHadoop [34] was proposed to enable direct data processing of PFS data in Hadoop. It utilizes the performance benefit and portability of MPI-IO to read data directly from PFS. However, the lack of scientific data format support requires unnecessary and time-consuming format conversion from scientific data format to ABDS-friendly text format. Additionally, the converted text file is typically much larger than the original format. It significantly increases the data movement across the network which could override the benefit of a copy-free design. To address user interface issues, R has been extended to support I/O libraries, such as *RNetCDF* and *rhdf5*. R also has been extended to interface with ABDS, such as *RHadoop* and *SparkR*, which enable the data access and processing power by utilizing the underlying Hadoop and Spark environment, respectively. Several works have been done in the R community to utilize the HPC and big data ecosystems, but not to integrate them.

In this research, we present SciDP, a runtime system for scientific data cross-system services. SciDP addresses the aforementioned technical barriers and was tested on NASA NCCS climate and weather applications for data analysis and visualization. SciDP supports an integrated data service at the file system level. It maintains the semantic data layout in scientific data formats. Users can launch data analysis on a Hadoop computing environment immediately after data is generated in an HPC environment. Additional R interfaces are developed to facilitate the data access and processing in an integrated environment. With SciDP, scientific and big data applications can directly access scientific data stored in HPC environments and processed in ABDS environments with the user-friendly R interface. The contributions of this paper include:

- Efficient integration of HPC and BD at file system and user-interface level.
- A general block mapping mechanism to support scientific input file formats used by the HPC community and avoids data conversion to shorten the workflow.
- Simplified direct processing of scientific data in BD environments via R user interface.
- SciDP speeds up the data analysis and visualization workflow by 6x to 8x on NASA *NASA-Unified Weather Research and Forecasting* (NU-WRF) application compared to existing state-of-the-art solutions.

II. Background

In this section, we will discuss methodology and challenges to data-intensive scientific research. Conventional solutions and their limitations will be

A. Scientific Simulation and Analysis Workflow

In a typical scientific research workflow, the process can be summarized into two computation phases: simulation/modeling, and data analysis/visualization. These phases usually form a dependency loop where one phase relies on the completion of the other phase to move forward. Additionally, results of one phase might be required for the next phase. The iterative workflow helps the scientists to gain potential scientific insights and verify their hypotheses in following iterations.

In the simulation phase, MPI-based simulations are running on an HPC cluster. The advance in mathematical models and computing power makes the simulation more sophisticated and represent the modeled physical world better in finer granularity. More data is used to improve the simulation accuracy. Thus, more data is generated and stored on PFS. The data is typically accessed via high level I/O libraries built on top of MPI-IO. These libraries provide user-friendly interfaces to allow semantic and metadata information to be stored along with the well-structured raw data. The data files are self-descriptive, which guarantees portability when the data is moved between different file systems in environments with various architectures. *Goddard Cumulus Ensemble* (GCE) [29], for instance, is a widely used *Cloud Resolving Model* (CRM) developed by NASA in the area of earth sciences to simulate the climate and weather system. These simulations typically write a large amount of data on the PFS. GCE takes 5 wall-clock days with 4096 computer processors at the NCCS Discover supercomputer and generates single-precision output data which is about 2.5TB in total with all the relevant variables [29], [36].

To gain meaningful insights from the simulation results, subsetting (e.g., selecting a part of the data) and in-depth data processing (e.g., visualization and data analysis) need to be performed upon the generated data. The targeted data will be subset and further analyzed and visualized for understanding and prediction. The subsetting, data analysis, visualization, and animation process on large data sets are not adequate on the MPI/PFS HPC cluster where the simulation is conducted. Scientists are using Hadoop and Spark as the data analysis framework to utilize their capabilities of scalable and powerful data analysis. These data analysis and visualization applications expect the residence of data on HDFS. The emerging machine learning techniques are also adopted to discover hidden patterns in the huge amount of data, which makes the learning curve of scientific data analysis steeper. Traditional interfaces and languages, such as MATLAB, R and SQL, minimize the learning cost and complexity for scientists, which improves the productivity in return. Writing ABDS application codes is a burden to scientists, which significantly lowers productivity.

Real Scientific Workflow Example: *Coupled Model Intercomparison Project Phase 5/6* (CMIP-5/6) is a typical workload in NCCS. It compares netCDF outputs from different MPI-based simulation models, whose size can be easily over several terabytes [30]. The comparison could be in either math-

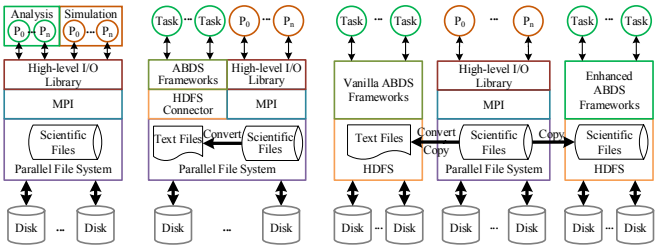

Fig. 1: Existing architectures and conventional solutions. (a) MPI-based; (b) Unified file system; (c) Separate file systems

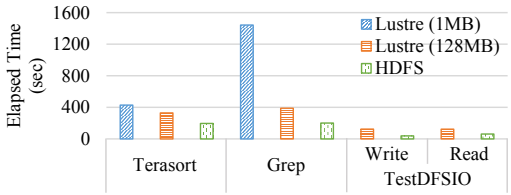

Fig. 2: Performance comparison between Lustre and HDFS.

ematical or visual form. Scientists can observe the differences from the numerical comparison results when the data size is small. In contrast, when data size is large, they have to use powerful data analysis and visualization tools to visualize the comparison outputs and reach meaningful conclusions of differences between models. The visual outputs are usually animations which consist of a series of images generated along a specific dimension. The exploration becomes increasingly easy as the animation granularity improves, either with prolonged time duration with more image frames, or higher image resolution of each frame. As the data size increases tremendously, it becomes more challenging to maintain the efficiency of such a workflow as before. Complicated user interfaces further increase the learning cost to obtain meaningful analysis and visualization results.

B. Conventional Solutions

MPI-based Solutions: Conventionally, data generated by MPI simulations is analyzed and virtualized on HPC clusters as depicted in Figure 1(a). MPI-based visualization tools (e.g., VisIt [5] and ParaView [3]) are used to perform the analysis. These tools usually run on the same hardware along with the simulation programs to reduce data movement. The hardware, on the other hand, is designed to maximize MPI-based simulations instead of data analysis and visualization workload. The nodes running data analysis applications will have to compete with simulation nodes for network bandwidth across nodes and remote global PFS which lowers the performance of both workloads.

Recently, ABDS frameworks, such as Hadoop and Spark, process outputs of large-scale simulations and have grown in popularity due to their capabilities in processing large amounts of data. These BD frameworks typically run on a separate cluster processing the data on a PFS shared with a larger simulation cluster for scientific research. To solve the data source problem, multiple solutions have been proposed.

Unified File System Solutions: HDFS connectors, such as IBMs HDFS Transparency [17], avoid the data copy by providing HDFS API and use PFS as the unified backend file system as shown in Figure 1(b), However, reading from PFS is not optimal since the PFS is optimized in favor of HPC workloads instead of BD analysis side. Figure 2 shows the performance comparison between Hadoop Terasort, Grep and TestDFSIO on native HDFS and Lustre laying underneath Seagates Lustre HDFS connector [26]. We use eight OSTs and eight Hadoop nodes. The stripe count for Lustre is configured to eight. We change the replication factor to one to force the same number of data copies for both solutions. We also configure Lustre to use large stripe size as the block size in HDFS to favor these workloads. It can be observed that native HDFS outperforms Lustre by 221% on average. We believe it can be attributed to the difference in access pattern preference of two file systems. PFS provides high bandwidth by aggregating data from multiple servers over network. HDFS minimizes latency and interference by maximizing local access. They emphasize on distinct characteristics to favor the native workloads in their environments. Therefore, unifying file systems using built-in connectors suffers in I/O performance. Additionally, the design does not support the merging in programming model level. Only file accessibility is enabled.

Separate File Systems Solutions: The architecture of two separate storage clusters can be used to provide better overall performance. Thus, multiple solutions have been developed to enable scientific data analysis and visualization in ABDS frameworks with the assumption of residence of data on HDFS. When data is copied from PFS to HDFS, users can select between default ABDS frameworks and enhanced ones for scientific research as illustrated in Figure 1(c). The vanilla ABDS frameworks do not natively support scientific data format. The typical preprocessing includes data format conversion. Scientific data files are converted to text files (e.g. CSV files) to enable processing in ABDS frameworks. Such process can be extremely slow and the converted data files are orders of magnitude larger than the original files, which significantly increases the data access time. The enhanced ABDS frameworks, such as SciHadoop and SciSpark, support direct processing of data in scientific format. New APIs are introduced to access, partition and process the scientific data on HDFS. The data copy, nevertheless, cannot be avoided. SciDP also targets the separate file system architecture with a novel software design.

PortHadoop: Lastly, PortHadoop is a dynamic PFS reader implemented in [34]. PortHadoop fetches data on-demand from PFS and stores the data in the memory of Hadoop nodes. PortHadoop adopts several design concepts and optimization techniques. 1) Instead of copying data between two file systems, only metadata information is saved on HDFS in PortHadoop when the input directory points to PFS. Virtual blocks are created in NameNode accordingly to map HDFS file blocks to the names and offsets of the source files on the PFS. 2) In each Map task, PortHadoop translates the metadata information in the virtual block into the corresponding file seg-

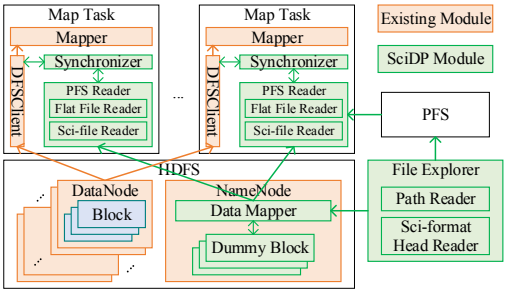

Fig. 3: SciDP component design.

ment on the PFS. It fetches the data via the standard MPI-IO interface. As a result, multiple file readers are spawned on multiple nodes to fetch the data in parallel. 3) PortHadoop inherits the portability from the standard interfaces. Thus, PortHadoop can access as many PFS (e.g., GPFS, Lustre, PVFS [10] and Ceph [33]) as the MPI-IO implementations deployed in the system support.

C. Limitations of Existing Solutions

There are several drawbacks of the existing solutions.

Offline scientific data processing. In existing MPI-based and separate file system solutions, scientific data is processed offline. A time consuming process, data copy or format conversion, has to be carried out before data is processed.

Unnecessary data transfer and format conversion. Unmodified ABDS frameworks and PortHadoop do not support direct processing of scientific data on PFS. Data copy in enhanced ABDS frameworks, such as SciHadoop and SciSpark, and data conversion in unmodified ABDS introduce significant unnecessary overhead to the workflow.

Inefficient support of merging two environments. Currently, the only attempt to merge HPC and BD environments is the HDFS connector solutions. However, those solutions suffer in performance.

III. SciDP Design

SciDP borrows the virtual mapping idea from PortHadoop, creates mirrored PFS input files on HDFS transparently. It is re-designed with considerations of general input file format support. SciDP minimizes the data path of typical scientific workflows with the direct processing power of scientific data on PFS in ABDS frameworks. The R interface is developed to facilitate using of SciDP and demonstrated using the analysis and visualization of a production simulation in Section IV.

A. SciDP Architecture

Three main components, File Explorer, Data Mapper and PFS Reader, are developed to enable direct scientific data processing in ABDS frameworks, as shown in Figure 3.

1) File Explorer: File Explorer is created to recognize the structure of the input data. For example, a path on the PFS with one netCDF output file and one CSV output file, plot_18_00_00.nc and plot_19_00_00.csv, is specified as the input path on the PFS. The netCDF output file contains two variables, *var_A* and *var_B*, for one simulated timestamp. The Path Reader in File Explorer searches the input path, and the Sci-format Head Reader examines the format of each file in the directory. Files are sorted into two different categories based on the decision of Sci-format Head Reader. Different mapping mechanisms will be used respectively in Data Mapper. Files are marked as flat (e.g., plot_19_00_00.csv) if they cannot be recognized by the Sci-format Head Reader using any supported scientific data format library, such as HDF5 and netCDF. A corresponding virtual file will be created on HDFS and the metadata of the source file is saved and sent to Data Mapper. If a scientific data file is detected (e.g., plot_18_00_00.nc), the metadata of this file will be extracted using the scientific I/O libraries (e.g., netCDF) and sent to Data Mapper as well. Since scientific data formats typically contain data variables organized as groups in a tree structure, similar directory structures are used to map to corresponding groups. A root directory is firstly created on the HDFS with the same name as the input files on the PFS (e.g., plot_18_00_00.nc). Virtual HDFS files are created to correspond to data variables (e.g., *var_A* and *var_B*). The path from the root directory to the files is created to mirror the group structures in the scientific input files. If the input files are in the data formats which support hierarchical structure, such as HDF5, deeper directory structures will be created correspondingly.

2) Data Mapper: When the virtual HDFS files are created, the mapping of the file content will be done by Data Mapper. Data Mapper utilizes the metadata obtained by File Explorer to calculate the virtual mapping between the HDFS blocks and file segments on the PFS. The HDFS blocks are dummies since no actual data is transferred and stored in HDFS. The dummy HDFS block works as a placeholder. This technique makes data accesses to scientific data on remote PFS transparent to ABDS applications. ABDS frameworks can obtain the file sizes and number of blocks through the native APIs to schedule the tasks. There is no location information in the dummy blocks as the native HDFS blocks. The data will be fetched from the PFS by the PFS Reader inside each task. Flat data files and scientific data files are mapped in two different ways due to the differences of internal structure, which will be discussed in Section III-B.

This process is similar to PortHadoop. However, the lack of scientific data format support in PortHadoop forces the conversion and leads to the loss of the semantic information of the original data. Manual data partition and indexing need to be done to use PortHadoop before any processing can be carried out. SciDP, on the other hand, inherits the metadata information in scientific data and brings it to the HDFS using the Data Mapper. With this information, SciDP can calculate the partition without any indexing beforehand.

3) PFS Reader: Compared with the conventional solutions, SciDP introduces the PFS Reader to read data directly into the memory of each Hadoop node without involving the master Hadoop node. PFS Reader requests the information about the source files on the PFS from the Data Mapper in the NameNode running on the master node. The actual I/O access happens independently in each task. SciDP follows the Hadoop

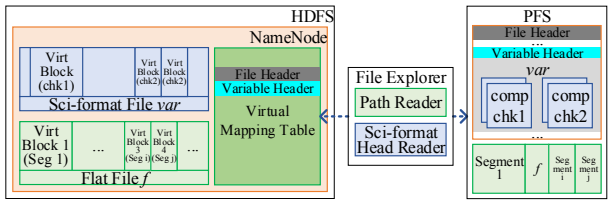

Fig. 4: Two data mappings in SciDP.

design and processes data by block. The original Hadoop reads 64KB data at a time until the end of the split. SciDP, on the other hand, reads the entire block in a single I/O request to maximize the bandwidth. The block size of mapped flat files is 128 MB by default for our implementation in Cloudera Hadoop [12]. The mapping of the scientific data will be discussed in Section III-B. Each task spawns a PFS reader to fetch data from the PFS. With multiple tasks running on multiple nodes at the same time, multiple PFS readers read the data in parallel. With the help of the high bandwidth provided by PFS, the data access can be done with high efficiency. Note that the physical memory capacity will not be a limitation for SciDP since the CPU cores are typically not over-subscribed in Hadoop environment [16]. Thus, it avoids extra memory consumption compared with the vanilla Hadoop.

Notice that SciDP does not affect how Hadoop processes data on HDFS but introduces an additional data source to the Hadoop framework. Data residing on HDFS is accessed and processed as default workloads. SciDP fully exploits the concurrency of the PFS since the PFS readers are spawned on multiple nodes to read data at the same time. Also, the data access in one task can be overlapped with data processing in other concurrently running tasks. SciDP is currently implemented based on Hadoop. However, the design of SciDP is general and can be applied to any ABDS framework.

B. Data Mapping in SciDP

Figure 4 illustrates the two different data mappings in SciDP. Firstly, Path Reader starts to scan the input path on PFS, count the input files and Sci-format Head Reader detects file formats. When flat files, such as file f in Figure 4, are detected, file blocks are mapped as they are on the PFS. The generated dummy blocks work as a mirror of the input flat files on the PFS. Each 128MB dummy block is mapped to a 128MB file segment on PFS. In this case, the boundaries between logical data blocks misalign with the boundaries between HDFS blocks. PortHadoop addresses this issue by reading extra data across the boundaries until the end of the record is found or by a scan-based indexing to align data records. Neither sequence files nor typical scientific data models, such as netCDF, is supported in PortHadoop. Analysis of scientific data, however, is typically carried out in a coarser granularity. The record will be much larger than one word or line of text which is commonly used in traditional big data analysis in ABDS frameworks. In such a workload, reading extra data bytes would significantly hurt the performance. SciDP avoids this problem by supporting analyzing scientific data files directly.

When the Sci-format Head Reader recognizes that the input files are in scientific data format, in contrast, the metadata in the input files will be utilized to build the mapping between two file systems. Dummy blocks will be created to map the internal structure of each file. Directories will be created to mirror the input files. Files will be created virtually following the variables in each input file. For the example in Figure 4, file *var* is created on HDFS according to the variable *var* in the input file. The file and variable header information is extracted by the Sci-format Head Reader and stored in the Virtual Mapping Table using APIs from scientific data format libraries, such as *nc_inq*, *nc_inq_var* for netCDF files. These headers contain information such as variable names, dimensions, data types, offsets and sizes.

In Figure 4, the data of variable *var* is partitioned into two compressed chunks. The first dummy block is created with the same size as the original chunk size instead of the default 128MB HDFS block size since the scientific data are typically processed following the structure of the variables. Unaligned data access will have a much higher overhead, due to reading extra compressed chunks, and, therefore, hurt the performance. User can also change the block size to tune the partition. Take the second chunk in Figure 4 for instance, it is mapped to two dummy blocks to split the workloads into two tasks. For the target application in this paper, NCCS Earth scientists may want to process the data in a finer granularity. Then the dummy block size can be adjusted to the actual size of one data grid. With this data mapping mechanism, SciDP also supports subsetting at the variable level. The user can specify a list of variables they want to analyze. SciDP will ignore the unrelated variables and attributes in the netCDF input files and minimize the time to build the mapping table.

SciDP's data mapping is universal. It does not only support netCDF. A similar procedure can be applied for other scientific data formats, such as HDF5. Ultimately, the input file format support is designed to be modular. Users only need to provide a file structure explorer and a corresponding reader to add support of arbitrary file formats.

C. Data Path of SciDP

TABLE I: Data Path of Existing Solutions and SciDP

Solution	Data Path		
	Conversion	***Data Copy***	***Processing***
Naive	Yes	Sequential	Sequential
Vanilla Hadoop	Yes	Parallel	Parallel
PortHadoop	Yes	No	Parallel
SciHadoop	No	Parallel	Parallel
SciDP	No	No	Parallel

As discussed in Section II, existing solutions have a wide variety of data paths. Table I lists the comparison of data paths of existing representative solutions and SciDP when they are used for analysis of scientific data. The naive solution does not take any merits of either system to parallelize any operation. The vanilla Hadoop solution utilizes the parallelism of the Hadoop cluster and parallelizes the data copy and processing. PortHadoop eliminates the data copy and improves

the performance. However, both solutions require conversion before starting copying or processing data. The misaligned boundaries caused by the loss of metadata information further degrade the performance. SciHadoop avoids the conversion by supporting scientific data format. The data copy of vanilla Hadoop and SciHadoop could be either sequential or parallel depending on user preference. It is listed as parallel in Table I to represent the fully optimized implementations of existing solutions. Compared with all existing solutions, SciDP maximizes the performance by avoiding both the conversion and the data copy. Parallel processing can start immediately when the data is generated. SciDP has the shortest data path and fully utilizes the parallelism of both two clusters to provide better support for the merging of the two environments.

IV. NU-WRF Case Study

In this section, we use the data analysis and visualization of NU-WRF simulation outputs as a use case to demonstrate how SciDP accelerates scientific data processing in Hadoop environment. For NU-WRF, NCCS scientists decide to deploy two separate clusters (as shown in Figure 1(c)), to carry out their scientific research as discussed in Section II-A. Recall the NU-WRF simulation runs on an HPC cluster and writes output in netCDF format onto PFS. The output is processed on an ABDS cluster. With SciDP, the workflow is completed with highly efficient data access, analysis and visualization.

A. Data Model

The simulation resolution of NU-WRF is represented in the format of time $\times$ altitude $\times$ longitude $\times$ latitude. NU-WRF uses 23 single-precision floating-point variables in the simulation. The output data of NU-WRF is in netCDF format. Each output file contains data of one timestamp. The spatial resolution can be configured to be high (i.e., 50x2,500x2,500) or low (i.e., 50x1,250x1,250). In this work, we use the data from a 48-hour low-resolution run and we use the variable rainfall (QR) as an example to demonstrate the proposed SciDP framework. Other variables can be analyzed and visualized in a similar way. Each variable is about 298MB in raw binary format. NU-WRF utilizes the chunking and compression feature introduced in the latest netCDF (i.e. netCDF-4) to reduce the size to around 91MB for each variable on average. In total, about 98GB of data is generated by the example NU-WRF run.

B. Optimized Data Access

As mentioned in Section II-B, data format conversion is required if vanilla ABDS frameworks or PortHadoop is used. The data size increases dramatically after the conversion. For the example NU-WRF run used in this study, the converted text data is ˜33x larger than the compressed binary data. Enhanced ABDS frameworks, such as SciHadoop, avoid the conversion. However, the inescapable data copy is still required to move the data between two clusters. Moreover, typically not all variables are accessed for data analysis. Since the netCDF file is not dividable in the variable level, the whole file has to be moved, which introduces redundant I/O. In contrast, SciDP only reads the variables that users select to the big data cluster.

C. Parallel Image Plotting in R

The extensive graphical abilities make R very popular in data analysis and visualization. *RHadoop* is a series of R packages for Hadoop. We choose R over the conventional *Interactive Data Language* (IDL) for more flexibility in image plotting and unification of programming interface with data analysis. The unified R interface certainly is a natural choice for system simplicity. Our focus, then, can be concentrated on integrating R with data transfer. In SciDP, the visualization is carried out using users' Map and Reduce function written in R. Multiple Map tasks are spawned to plot images in parallel to utilize the parallelism of the Hadoop cluster. The data access in each task is also performed in parallel and overlaps with each other across all nodes.

D. Parallel Data Analysis in R

Since we use R for visualization, we can also analyze the data along with the visualization. Users can manipulate the data as in a typical R application since the data has been converted into native R data structures by SciDP. Moreover, the data analysis is carried out in parallel as the visualization since multiple Map and Reduce tasks are running at the same time across multiple nodes. Users can develop task-independent data analysis in each Map function and perform cross-task analysis in each Reduce function.

E. Implementation

Our prototype proof-of-concept SciDP[1] is implemented in Hadoop and supports netCDF format. The design of SciDP is general and not bound to R interface and netCDF format. Support of other interfaces (e.g., Python) and scientific formats (e.g., HDF5) can be implemented on top of SciDP as well.

1) Main Components: We modify the *addInputPath* function in *FileInputFormat* class in Hadoop to implement the File Explorer. During the input path parsing, the input path will be compared with the predefined PFS prefix (e.g., gpfs:// or lustre://). The prefix can be added as an option when the job is submitted. If a match cannot be found, SciDP will behave as the original Hadoop and read data from HDFS. If the predefined prefix is matched, dummy blocks will be created as described in Section III-A. The Sci-format Head Reader detects the format of each file in the input path by attempting to open it using the open or the format checking function provided by scientific data format libraries, such as *nc_open* and *H5Fis_hdf5*. A boolean value will be returned to indicate the format of the input files.

The Data Mapper is also implemented in *FileInputFormat* class. Once the File Explorer returns a list of input files with their formats, Data Mapper will create files and directories following the design in Section III-A. A *DFSClient* instance is created to talk with NameNode for file and directory creation. The block size is determined using the default or customized

[1]Source code available at: https://bitbucket.org/fkengun/scidp

size given by the user. The metadata of each block is enriched with information of the mapped file segment or hyperslab.

We also modify the *MapTask* class in Hadoop to enable the PFS reader. The PFS reader accesses the file segments from the PFS using MPI-IO functions (e.g. *MPI_File_open*, *MPI_File_read_at*, *MPI_File_close*). NetCDF APIs (e.g. *nc_open*, *nc_get_vara*, *nc_close*) will be used to access hyperslabs of variables when netCDF files are detected.

2) API: We implemented an R interface to enable an easy-to-use access to SciDP. Users need to specify the format of input files. For instance, the NU-WRF simulation data used in this study is in netCDF format and consists of multiple variables with 3D single-precision floating-point array. Multi-dimensional array will be prepared as R data frame. Thus, users can manipulate the simulation data as they want.

3) Parallel Processing: For the NU-WRF case study, we conduct both image plotting and data analysis in R. The dimension information in the block mapping table will be used to generate the coordinate. The rainfall value is used to plot the image using the *image2D* function provided by *plot3D* package on the graphical device created by *CairoPNG* function in the *Cairo* package. All the intermediate outputs in Map tasks are indexed by the key which is generated from coordinate attributes in input files. The data is reused for local analysis without extra data read. SQL queries are supported by the *sqldf* package. It converts the SQL queries into operations upon R data frames since R data frames are similar as tables. The images and the analysis results will be aggregated according to the key. The results will be combined and stored into HDFS using the *rhdfs* package in Reduce task. *rmr2* provides the fundamental API support to communicate with underlying Hadoop. Notice that SciDP only requires the *rhdfs* and *rmr2* package to work. *Cairo*, *plot3D* and *sqldf* packages are only used for the visualization and analysis of the NU-WRF data. They are not required for the key function of SciDP.

V. Evaluation

A. Setup

Testbed: We have implemented and evaluated SciDP on the Chameleon cluster available at the Texas Advanced Computing Center (TACC) [2]. Each compute node in Chameleon is equipped with two 12-core Intel Xeon E5-2670 v3 CPU, 128 GB memory, and a 250 GB 7200 RPM SATA hard drive. All nodes are connected via a 10 Gigabit Ethernet. On the other hand, each storage node has 64GB memory and 16 2TB 7200 RPM SAS hard drives. Eight compute nodes are configured as Hadoop slaves by default. Three storage nodes are used as MGS, MDS and OSS in Lustre. 24 OSTs are managed by two OSS nodes. The replication factor of HDFS is set to one.

Software: We implemented SciDP based on Cloudera Hadoop 5.3.3 [12]. We obtained from NCCS the output from a 48-hour simulation of the NU-WRF model with a resolution of 50x1,250x1,250. Data of 48 timestamps is saved in 48 files. However, the data set is not large enough for our test. We developed a synthetic data generator to produce a much larger data set. We increased the input data to 96, 192, 384 and 768 timestamps, respectively. Notice that the synthetic data sets follow the same dimensions, chunking and compression ratio as the real data set. It can evaluate the existing solutions and SciDP more comprehensively. The data sets are in the netCDF format. For SciDP, we process the original netCDF files on the PFS directly. The default resolution of images is 1,200x1,200. For other solutions, copy and conversion are carried out as necessary to meet the preconditions of different solutions.

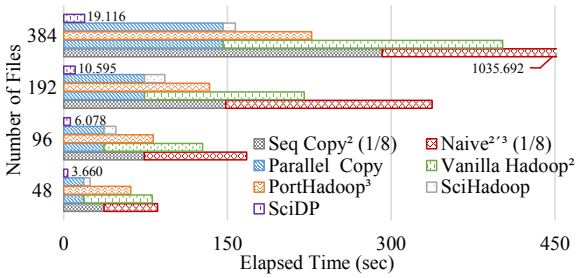

Fig. 5: SciDP and existing solutions' total execution time.[2, 3]
[2] Serial copy and naive solution are too slow to fit in. They are presented with 1/8 of their actual execution time. [3] The conversion time required by these solutions is too long and not presented here.

TABLE II: Representative Workloads

Workload	Image Plotting	Animation	Analysis
Img-only	Yes	No	No
Anlys	Yes	Yes	Yes

Workloads: Table II shows the workloads we use to test SciDP. Img-only workload includes only the image plotting phase which can be fully parallelized. The Anlys (short for Analysis) workload covers all three phases. The analysis phase could be conducted via SQL query or R-based statistical analysis. We compare SciDP with the conventional solutions using Img-only workloads. Additionally, Anlys workload is added to evaluate SciDP in a more sophisticated scenario.

Considerations: In this section, we compare SciDP with those four solutions listed in Table I, which represent all existing solutions discussed in Section II. We implement all solutions in R except SciHadoop since it does not have R interface support. We do not compare with unified file system solutions, since the PFS connectors have very poor performance as shown in Figure 2. Notice that the conversion time is not presented in this paper because it is known to be extremely slow. We convert the 14GB NU-WRF outputs. It finishes in more than one hour. Therefore, we do not count the conversion time into the total time in any tests of this paper. The data copy step is individually measured and explicitly presented in this paper since there is no overlap between data transfer and the processing for naive, vanilla Hadoop and SciHadoop. The processing time of each solution is added on top of the data copy time if it is required.

B. Performance Speedup for Img-only

Figure 5 shows the performance comparison between existing solutions and SciDP. The naive solution is the slowest. It takes orders of magnitude longer time than other solutions to

TABLE III: Speedup of SciDP Over Existing Solutions

Number of Files	Naive	Vanilla Hadoop	Porthadoop	SciHadoop
48	187.65	22.16	16.80	6.58
96	220.63	20.93	13.49	7.86
192	254.78	20.79	12.60	8.74
384	284.63	21.04	11.88	8.22

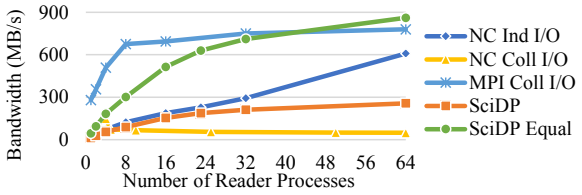

Fig. 6: I/O bandwidth of SciDP and other I/O methods.

plot images for all grids (too long to fit in, plotted using 1/8 of total execution time). In addition to that, the sequential copy makes it slower (plotted similarly to image plotting time). The vanilla Hadoop does both copying and plotting in parallel, which increases the overall performance. PortHadoop saves more time by getting rid of the data copy. However, all these three solutions require format conversion which cannot be presented in the figure as mentioned in Section V-A. SciHadoop, in contrast, removes the requirement of conversion and processes the data after they are moved from the PFS to the HDFS. The data copy time, although accelerated by the parallel copy in *distcp*, is significantly longer than the processing time. It slows down SciHadoop dramatically. Such an unbalanced data access time and processing time can be attributed to the redundant I/O of copying the whole dataset while only several variables are typically needed for the processing. SciDP outperforms all the existing solutions by 6.58x to 284.63x according to Table III. In our previous work [4], a 15x speedup over native solution was achieved to process CSV input files. SciDP processes netCDF files directly without the conversion and improves the speedup to 284.63x compared with native solution. It shortens the data path by avoiding the conversion and the data copy. The data access in Img-only workload is also minimized by only fetching the required variables from the PFS. More importantly, the data transfer between two clusters is overlapped with the image plotting, which further boosts the performance. SciDP utilizes the parallelism of both systems and overlaps the data access with data processing to generate the best performance.

C. I/O Efficiency

SciDP targets to solve the data access problem between two systems. Figure 6 demonstrates the I/O efficiency of SciDP compared with the typical HPC I/O mechanisms. Two netCDF methods (NC Ind I/O and NC Coll I/O) read data via netCDF library APIs (e.g. *nc_get_var*) using different I/O modes (i.e. independent and collective), respectively. MPI Coll I/O shows the maximal bandwidth we can reach by ignoring the netCDF structure and reading the input as a flat file. It is only used to illustrate the upper limit of the ideal performance. The measured I/O bandwidth of SciDP is presented as SciDP and

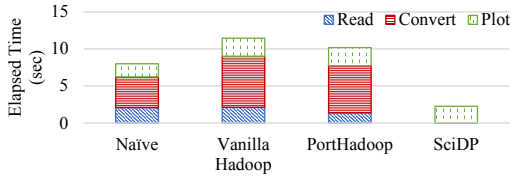

Fig. 7: SciDP and existing solutions' task time decomposition.

SciDP Equal. They are calculated by dividing the compressed data size and raw data size over I/O time, respectively. The I/O time includes both the actual data access time and the decompression time. Thus, SciDP Equal is more practical to evaluate the I/O efficiency of SciDP as it is calculated using the actual processed data size. It can be observed that SciDP is very efficient in I/O. The equivalent bandwidth (SciDP Equal) is closer to the pure MPI collective I/O method (MPI Coll I/O) when the number of readers/processes increases. The parallel I/O design of SciDP is the source of the highly efficient I/O. In SciDP, the reader in each task works similar with a single rank MPI application. All readers work together to fetch data in parallel to guarantee the high I/O efficiency.

D. SciDP Task Time Decomposition

As discussed in Section IV-E2, SciDP reads data from PFS first and consequently converts into R data format before plotting. Read, Convert and Plot are three sub-phases in the image plotting phase. Figure 7 presents the decomposed execution time of three sub-phases in one run of Img-only workload when processing 384 data files. It shows that Convert takes the longest time for naive, vanilla Hadoop and PortHadoop. That is because they use the *read.table* function which sequentially converts data from raw format (i.e. text) into correct data types. In contrast, SciDP does not need this text-to-binary conversion. The binary data fetched from the PFS can be converted to R structure in a very short time. The I/O cost, the Read time, takes about 2 seconds in each task for three existing solutions. SciDP takes about 1.75 seconds to read a variable with 50 levels, which is equal to 0.035 seconds per one level data shown in Figure 7. Vanilla Hadoop, PortHadoop and SciDP have nearly the same Plot time since we implement the image plotting in the same way for these three solutions after the data is converted to R structure. Naive takes less time to plot a level of data compared with other solutions. That is because it processes data in a sequential fashion without resource contention in memory and disk bandwidth. The benefit of the parallelism in the other three solutions overrides the performance gain of contention-free sequential code. With improvement and optimizations in both I/O and parallel processing, SciDP outperforms the exiting solutions in average task time.

E. Scalability Evaluation of SciDP

The scale-out evaluation of SciDP for Img-only workload is shown in Figure 8. 4, 8 and 16 compute nodes are deployed with SciDP to evaluate SciDP's performance as the system scales out. In this test, 8 tasks are run on each node which leads to 32, 64 and 128 possible parallel tasks for four test cases,

respectively. SciDP shows good scale-out performance. The image plotting time reduces nearly in half when the number of nodes doubles which leads to a near-optimal speedup. This can be attributed to the nature of the workload. Image plotting in each task is completely independent. There is no inter-task communication and data dependency across tasks. Scale-up evaluation shows similar performance as scale-out results. Due to the page limit, we do not include them here.

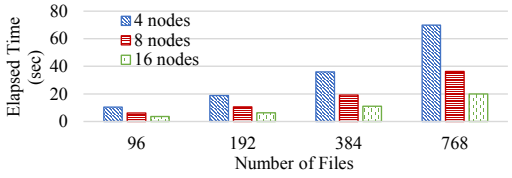

Fig. 8: Scale-out evaluation of SciDP.

F. Parallel Data Analysis Using SQL Query

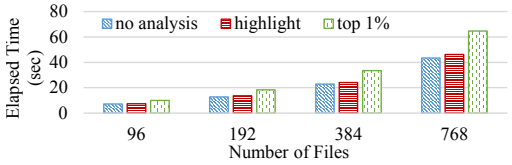

Fig. 9: Data analysis performance of SciDP.

To demonstrate the integrated analysis performance of SciDP, we use two typical analysis in the Anlys workload and run them after image plotting in the same Map task. In Figure 9, SQL queries are used for the analysis. no analysis represents the Img-only workload as the base line. The top 10 data points are highlighted in the highlight case. Top 1% data is selected and stored in HDFS in top 1% case, respectively. In Figure 9, we can observe that the highlight case takes almost the same time as the no analysis case. It indicates that the analysis takes very short time. The reason is threefold. Firstly, it is because the computation of highlight is relatively small and negligible. Secondly, the extra data gathered from each Map task is also small (the difference in image size is very small after highlighting). Thirdly and most importantly, it is because no extra data read is required as described in Section IV. Thus, simple data analysis does not take extra execution time and performs really well.

On the other hand, the data processing time of more complex SQL queries is much larger than that of the simple data analysis. Data size generated in each Map task is proportional to the total input data size. In this evaluation, the size of the query result for top 1% case is about 596 MB for one variable. Since the SQL query results are stored on HDFS after partial results are collected from each task, more data are written into HDFS in top 1% case. More network data transfer and more disk writes lead to an increase in the execution time.

VI. Related Work

There are several works to enable data access and processing of HPC data in Hadoop. MRAP [27] extended the MapReduce paradigm to support HPC data (e.g., netCDF) semantic description in Hadoop. Similarly, SciHadoop [9] was proposed to facilitate scientific data processing in Hadoop by maintaining the physical-to-logical mapping. SIA [20] was designed to build spatiotemporal index for big array-based climate data. A table was created and stored for each grid in the climate data to allow indexed data access in Hadoop. IBM builds a Hadoop connector to allow Hadoop to communicate with GPFS directly [17]. Multiple HDFS connectors were developed for Lustre as well [18], [26]. Ceph [33] provided three sets of APIs to support both HPC and BD workloads. Cephs file system (CephFS) can be mounted as a POSIX-compatible file system using Ceph client in Linux kernel since 2.6.34. The block storage system of Ceph can be used for ABDS frameworks. All these works either require an explicit data copy to transfer data from PFS to Hadoop or are designed for a specific PFS. In contrast, SciDP is designed for the environments with two separate clusters and prepares the data in memory from any PFS which MPI-I/O and POSIX I/O support. SciSpark [22] and H5Spark [21] extend Spark to support netCDF and HDF5, respectively. However, SciSpark does I/O in sequential for a single input file [21]. H5Spark has a similar design with SciDP for Spark while SciDP is based on Hadoop.

Researchers in HPC also manage to enable MapReduce-like data processing on HPC platforms. Smart [32] is a framework to allow HPC users to analyze the generated data in an in-situ and MapReduce-like way. ArrayUDF [13] allows users to define custom function to apply over the data. However, users have to learn the new API instead of utilizing the mature and powerful existing works in Hadoop to accomplish the analysis. In contrast, SciDP can exploit all the merits of Hadoop to meet users various requirements.

R is a popular and powerful programming language for data analysis and visualization. To our best knowledge, SciDP is the first framework to enable direct analysis of HPC data in R environment. RHIPE builds an integrated programming environment for R and Hadoop to support processing large complex data [15]. RHadoop [6] is a collection of five R packages to enable R data analysis under Hadoop environments. However, they do not support any scientific data format and expect data stored on HDFS. On the contrary, SciDP enables R to directly process remote scientific data stored on PFS.

VII. Conclusions

Data-intensive HPC, *High Performance Data Analytics* (HPDA), and other newly emerged HPC and big data applications demand both high-performance computing power and high-performance data processing power. There is a need to utilize the currently separated HPC and big data ecosystems to support both computing and data processing. However, how to support this integrated requirements effectively is still a subject of research. In this study, we introduced the SciDP system to support the demand of integration. SciDP supports processing data in scientific formats, such as netCDF and HDF5, under a Hadoop environment using an R-based interface. SciDP reaches a 6x to 8x speedup on a typical Earth science

application in MPI-Hadoop cross-platform data analysis, in comparison with the state-of-the-art solutions which support scientific data format in Hadoop. SciDP illustrates the vital important issue of supporting the merging of HPC and big data: maintaining, utilizing, connecting, and integrating the potential of existing HPC and big data technologies into an integrated environment. SciDP provides a successful example and feasible solution for utilizing the potential of PFS and Hadoop, and for supporting the merging of computing and data analysis. Because of the general representativeness of the NASA Earth science application, the results presented in this study are general and can be extended to other applications.

Integrating HPC and big data ecosystems is a long-term goal. The ecosystem integration can be achieved at various levels. SciDP supports the integration at file system level. It is the first step of supporting the merging requirements of HPC and big data systems. SciDP can be extended to support other BD frameworks, such as Spark and Impala, and the general integration of computing and data analysis. In the future, we plan to extend SciDP to more ABDS frameworks to enable more users with the direct processing power of PFS data.

VIII. Acknowledgement

This research is supported in part by NSF under NSF grants CNS 0751200, CCF 0937877, CNS 1162540 and NASA AIST 2014.

References

[1] "Apache HBase." [Online]. Available: https://hbase.apache.org/
[2] "Chameleon." [Online]. Available: https://www.chameleoncloud.org/
[3] "ParaView." [Online]. Available: http://www.paraview.org/
[4] "Reaching for the Stormy Cloud with Chameleon - Latest News - Texas Advanced Computing Center." [Online]. Available: https://www.tacc.utexas.edu/-/reaching-for-the-stormy-cloud-with-chameleon
[5] "VisIt." [Online]. Available: https://wci.llnl.gov/simulation/computer-codes/visit
[6] R. Analytics, "RHadoop." [Online]. Available: https://github.com/RevolutionAnalytics/RHadoop
[7] Apache Software Foundation, "Apache Hadoop," 2011. [Online]. Available: http://hadoop.apache.org/
[8] ——, "Apache Impala," 2012. [Online]. Available: http://impala.apache.org/
[9] J. B. Buck, N. Watkins, J. LeFevre, K. Ioannidou, C. Maltzahn, N. Polyzotis, and S. Brandt, "SciHadoop: Array-based Query Processing in Hadoop," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*. New York, New York, USA: ACM Press, 2011, p. 1.
[10] P. H. Carns, I. I. I. Walter B. Ligon, R. B. Ross, and R. Thakur, "PVFS: A Parallel Virtual File System for Linux Clusters," in *Proceedings of the 4th Annual Linux Showcase and Conference*, 2000, pp. 317–327.
[11] N. Chaimov, A. Malony, S. Canon, C. Iancu, K. Z. Ibrahim, and J. Srinivasan, "Scaling Spark on HPC Systems," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing - HPDC '16*. New York, New York, USA: ACM Press, 2016, pp. 97–110.
[12] Cloudera, "Apache Hadoop open source ecosystem — Cloudera." [Online]. Available: https://www.cloudera.com/products/open-source/apache-hadoop.html
[13] B. Dong, K. Wu, S. Byna, J. Liu, W. Zhao, and F. Rusu, "ArrayUDF: User-Defined Scientific Data Analysis on Arrays," *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 53–64, 2017.
[14] M. Folk, A. Cheng, and K. Yates, "HDF5: A File Format and I/O Library for High Performance Computing Applications," in *Proceedings of Supercomputing*, 1999.
[15] S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, and W. S. Cleveland, "Large complex data: divide and recombine (D&R) with RHIPE," *Stat*, vol. 1, no. 1, pp. 53–67, oct 2012.
[16] Hortonworks, "Managing CPU Resources in your Hadoop YARN Clusters." [Online]. Available: https://hortonworks.com/blog/managing-cpu-resources-in-your-hadoop-yarn-clusters/
[17] IBM, "HDFS Transparency," 2012. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/STXKQY_4.2.0/com.ibm.spectrum.scale.v4r2.adv.doc/bl1adv_Overview.htm
[18] Intel, "Hadoop Adapter for Lustre (HAL)." [Online]. Available: https://github.com/intel-hpdd/lustre-connector-for-hadoop
[19] L. Jianwei, L. Wei-keng, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netCDF: A High-Performance Scientific I/O Interface," in *Proceedings of the ACM/IEEE ConferenceSupercomputing*, 2003.
[20] Z. Li, F. Hu, J. Schnase, and D. Duffy, "A spatiotemporal indexing approach for efficient processing of big array-based climate data with MapReduce," *International Journal of Geographical Information Science*, vol. 8816, no. February, pp. 17–35, 2016.
[21] J. Liu, E. Racah, Q. Koziol, R. S. Canon, A. Gittens, L. Gerhardt, S. Byna, and M. F. Ringenburg, "H5Spark : Bridging the I/O Gap between Spark and Scientific Data Formats on HPC Systems," *Proceedings of the Cray Users Group*, 2016.
[22] R. Palamuttam, R. M. M. M. Mogrovejo, C. Mattmann, B. Wilson, K. Whitehall, R. Verma, L. Mcgibbney, and P. Ramirez, "SciSpark : Applying In-memory Distributed Computing to Weather Event Detection and Tracking," in *2015 IEEE International Conference on Big Data (Big Data)*. Santa Clara, CA, USA: IEEE, oct 2015, pp. 1959–1965.
[23] R. L. Rob Ross Rajeev Thakur,, Marc Unangst, and B. Welch, "Parallel I/O in Practice," in *Supercomputing*, 2009.
[24] F. Schmuck and R. Haskin, "GPFS: A shared-disk File System for Large Computing Clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, 2002, pp. 231–244.
[25] P. Schwan, "Lustre: Building a File System for 1000-Node Clusters," in *Proceedings of the 2003 Linux Symposium*, vol. 2003, 2003.
[26] Seagate, "Diskless Hadoop 2 (YARN) on Lustre." [Online]. Available: https://github.com/Seagate/hadoop-on-lustre2
[27] S. Sehrish, G. Mackey, J. Wang, and J. Bent, "MRAP: A Novel MapReduce-based Framework to Support HPC Analytics Applications with Access Patterns," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*. New York, New York, USA: ACM Press, jun 2010, p. 107.
[28] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010*. IEEE, 2010, pp. 1–10.
[29] W. Tao and J. Simpson, "Goddard cumulus ensemble model. Part 1: Model description," *Terr. Atmos. Oceanic Sci*, no. 1, pp. 35–72, 1993.
[30] K. E. Taylor, R. J. Stouffer, and G. A. Meehl, "An overview of CMIP5 and the experiment design," *Bulletin of the American Meteorological Society*, vol. 93, no. 4, pp. 485–498, apr 2012.
[31] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, aug 2009.
[32] Y. Wang, G. Agrawal, T. Bicer, and W. Jiang, "Smart: a MapReduce-like framework for in-situ scientific analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15*. New York, New York, USA: ACM Press, 2015, pp. 1–12.
[33] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A Scalable, High-performance Distributed File System," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, 2006, pp. 307–320.
[34] X. Yang, N. Liu, B. Feng, X.-H. Sun, and S. Zhou, "PortHadoop: Support direct HPC data processing in Hadoop," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, oct 2015, pp. 223–232.
[35] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, p. 10, 2010.
[36] S. Zhou, X. Yang, X. Li, T. Matsui, S. Liu, X. H. Sun, and W. Tao, "A Hadoop-based visualization and diagnosis framework for earth science data," *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pp. 1972–1977, 2015.