Enabling Scientific Data Storage and Processing on Big-data Systems

Saman Biookaghazadeh[*], Yiqi Xu[†], Shujia Zhou[‡], Ming Zhao[*†]

[*] School of Computing, Informatics, and Decision Systems Engineering, Arizona State University

[†] School of Computing and Information Sciences, Florida International University

[‡] Northrop Grumman Information Technology

***Abstract*—Big-data systems are increasingly important for solving the data-driven problems in many science domains including geosciences. However, existing big-data systems cannot support the self-describing data formats such as NetCDF which are commonly used by scientific communities for data distribution and sharing. This limitation presents a serious hurdle to the further adoption of big-data systems by science domains and prevents scientific users from leveraging these systems to improve their productivity. This paper presents a solution to this problem by enabling big-data systems to directly store and process scientific data. Specifically, it enables Hadoop to efficiently store NetCDF data on HDFS and process them in MapReduce using convenient APIs. It also enables Hive to support standard queries on NetCDF data, transparently to users. The paper also presents an evaluation of the proposed solution using several representative queries on a typical geoscientific dataset. The results show that the proposed approach achieves substantial speedup (up to 20 times) and space saving (83% reduction), compared to the traditional approach which has to convert NetCDF data to CSV format for Hadoop and Hive to use them.**

Keywords— Scientific data, big data, NetCDF, Hadoop

I. Introduction

Big data is an important computing paradigm increasingly used by many disciplines for knowledge discovery, decision making, and other data-driven tasks based on processing and analyzing large volumes of data. Big-data systems are typically built upon programming frameworks that can effectively express data parallelism and exploit data locality (e.g., MapReduce [1]) and storage systems that can provide high scalability and availability (e.g., Google File System [2], Hadoop HDFS [3]). A variety of high-level data services (e.g., BigTable [4], HBase [5], Hive [6]) can be further built upon such frameworks.

Typical geoscience models have multi-scale physical processes. For example, climate and weather models have physical processes involving spatial resolutions from meters to miles and temporal resolutions from seconds to hours. With current high performance computing power, ultra-high-resolution, long-time simulations are feasible with a few thousands of computer processors. Consequently huge amounts of data (easily over 100 TB) are produced. Big-data technologies are demanded to analyze the simulation outputs to address questions such as climate change and hurricane tracking.

Scientific data are often stored in self-describing data formats (e.g., NetCDF [7], HDF5 [8]). Self-describing data are understandable to both human and machines, with which programs can use existing procedures to not only access but also convert and probe the data. Today observational data as well as simulation data of geosciences are being widely shared among peer research organizations and available for public. Therefore, self-describing data formats are crucial for enabling this kind of data distribution and sharing.

However, big-data systems do not support these scientific data formats. For example, a NetCDF file loaded into HDFS as raw data cannot be processed by MapReduce applications. Consequently, scientific users who wish to use big-data computing for their applications often have to convert their data to a much more primitive data formats (e.g., Comma Separated Values (CSV)) which causes substantial time and space overhead. For instance, our results show that, to convert a 20GB NetCDF3 file to CSV, it requires 119GB disk space and 1.4 hours on a commodity server. This overhead is even worse when considering that the data need to be replicated at least three times on a big-data system in order to tolerate failures. Therefore, lack of support for scientific data formats presents a serious hurdle to the further adoption of big-data technologies for data-driven sciences, including geosciences, and to the further improvement of scientific productivity in such domains.

This paper presents an approach to addressing the above issue by enabling commonly used big-data systems to directly support the storage and analysis of scientific data stored using self-describing formats. First, we extend Hadoop, the most widely used big-data system to store NetCDF data on HDFS, and to allow MapReduce jobs to parse and process these data. We also optimize MapReduce to read variables of the NetCDF data in batch for much improved performance. Second, we extend Hive, a commonly used query-based big-data system to allow users conveniently using queries to process NetCDF data stored on HDFS. Our extensions are largely transparent to users. In MapReduce, our new APIs for processing NetCDF data are provided in the same fashion as the existing APIs for other data formats, and users can conveniently use them in their programs. In Hive, our handling of NetCDF data is completely transparent to users who can use standard HIVE SQL queries to easily process the data.

The first two authors contributed equally to this work.

We have developed a prototype of this approach based on Hadoop 2.5.2, Hive 1.2.0, and NetCDF3. Our prototype allows NetCDF3 data to be directly stored on HDFS and be directly used by MapReduce jobs and Hive queries. We have evaluated the performance of our approach using a typical geoscience dataset on a nine-node compute cluster. The results show that our approach is able to substantially improve both performance and disk space consumption compared to the traditional CSV-based approach. It increases performance by up to 20 times and decreases the disk space usage by 83%.

To the best of our knowledge, our work is the first to provide native support of widely used scientific data format on big-data systems. Related work [11] also considered the use of HDFS to store scientific data and MapReduce to implement array queries. But in its design MapReduce interacts with data via the scientific data model and loses control and knowledge of the physical data distribution, which causes performance problems due to the mismatch between the logical view and physical view of the data. In comparison, our approach enables big-data systems, including both MapReduce and Hive, to directly support scientific data and optimize task scheduling based on the physical data placement. Therefore, our approach allows users to conveniently and transparently use the existing big-data frameworks to process scientific data with good scalability.

The rest of this paper is organized as follows. Section 2 introduces the background and related work. Sections 3 presents the design and implementation of our approach. Section 4 discusses the experimental evaluation results. Section 5 concludes the paper and outlines the future work.

II. Background and Related Work

A. Big-data Systems

Typical big-data systems are built upon a highly scalable and available distributed storage system. For example, Google File System (GFS) [2] and its open-source version, Hadoop Distributed File System (HDFS) [3], provide fault tolerance while storing massive amounts of data on a large number of datanodes built with inexpensive commodity hardware; while MapReduce [1] applications are executed in a data-parallel fashion on the datanodes where their data are stored. High-level data services such as databases (e.g., BigTable [4], HBase [5], Hive [6]) can also be built upon such a big-data computing framework.

When data are loaded into a big-data file system such as GFS and HDFS, they are split into large data blocks which are distributed across the datanodes in the system. Both the map and reduce phases of a MapReduce application can spawn large numbers of map and reduce tasks, depending on the size of the input, on the datanodes of a big-data system to process the data in parallel. To take advantage of data locality, which is key to the performance of MapReduce applications, the map tasks are preferably scheduled onto the datanodes that have the data blocks for them to process locally, thereby shipping computing to the data. Moreover, the data blocks on the big-data file system are typically replicated at least three times across the datanodes and across the racks of nodes in the system in order to tolerate node-level and rack-level failures.

Big-data systems often support several common data formats for storing data on the systems. For example, Hadoop supports *SequenceFile*, *NLine*, *KeyValue*, *FixedLength*, etc. It provides libraries for MapReduce jobs to parse data stored using these formats on HDFS and process them in parallel based on the layout of the data across the datanodes. There are also other related libraries (e.g., Apache Parquet [12]) which support more complex data formats on Hadoop. However, because such big-data systems were originally not designed with scientific data in mind, they typically do not support the self-describing data formats commonly used by scientific data.

While MapReduce makes it much easier to program for data parallelism, it is still an involved and time-consuming task. Therefore, many users prefer using high-level data services such as Hive to simplify the use of big-data systems for processing their data. Hive stays on top of Hadoop and enables users to process data stored in HDFS using common SQL queries. It transforms a query into a set of map and reduce tasks to be deployed on Hadoop, and returns the final result back to the user.

B. Need of Support for Scientific Data Formats in Big-data Systems

The de facto data formats used in many science domains, including geosciences, are the self-describing formats such as NetCDF [7] and HDF [8]. They provide a concise and efficient way of storing array-oriented scientific data in binary. They are self-describing and machine-independent, which means that the description of data is not only well-defined in machine-understandable way but also meaningful to human and conforms to relevant conventions [8]. For scientific applications, geosciences in particular, a wide variety of named dimensions and variables have been frequently used. They share the usage of a large scientific user community. Existing conventions enable the cooperation and reuse of both standards and codes to transform, combine, analyze, and display specified fields of the data [7]. For example, the setting of grids and physical units for climate and weather simulations vary among different models. Self-describing data formats facilitate the sharing of climate and weather data. NetCDF and HDF have been used in Earth System Grid [13]. The Earth System Grid Federation (ESGF) is an international collaboration with a current focus on serving the World Climate Research Programme's (WCRP) Coupled Model Intercomparison Project (CMIP) and supporting climate and environmental sciences in general.

Many science domains are increasingly data driven, requiring the processing of large amounts of simulation, experimental, and observational data for scientific discoveries. For example, the experimental data from Large Hardon Collider may provide better answers to the fundamental questions in physics; the observational data from the upcoming Large Synoptic Survey Telescope will provide greater insights into the structure of the Universe; and to improve the predictability of hurricane tracking, a large amount of real-time sensor data from various types of instruments need to be processed and incorporated into forecasting models.

Therefore, big-data systems are also important platforms for these science domains by providing the necessary scalability and reliability for storing and processing big scientific data. They are indeed increasingly used by scientists from different domains including geosciences. However, existing big-data systems do not support the data formats commonly used by scientific data. For example, one can load NetCDF files into HDFS as binary data, but MapReduce applications cannot interpret these data properly for processing. Consequently, users often resort to a cumbersome approach to using these systems for processing their data. First, they have to convert their data stored in self-describing format, e.g., NetCDF, to plain-text format using tools such as *ncdump* [14]. After conversion, the file needs to be further translated to a multi-column table format such as CSV that is supported by a big-data system such as Hadoop.

This approach is not only cumbersome to users but also incurs substantial time and space overhead. First, the *ncdump* conversion increases the output file size to about three times of the original NetCDF data size. The further translation to Hive accepted format, adding dimension information to each row, adds an additional two to three times of space overhead. Hence, the space usage of this approach is bloated up to at least six times of the original NetCDF data size. Because these data need to be replicated at least three times on the big-data system, the absolute space usage can be prohibitive for large scientific data sets. Second, it takes time, storage, and network I/O bandwidth to load the data into HDFS. Because the data size is increased multi-fold by the conversion, this overhead is also increased substantially. Finally, as the data size gets bloated up by the conversion, it requires more map tasks to process the data, and more time to perform I/O tasks on the data, which in the end causes the data processing to consume much more resources and time.

Therefore, there is an urgent need to enable commonly used big-data systems to support scientific data formats. While there are more advanced data formats (e.g., Orc [8] and Parquet [9]) in big-data systems, they cannot be directly used to store scientific data that come in a self-describing format. SciHadoop is a related project which stores scientific data on HDFS and implements its custom array query language using MapReduce jobs [11]. But it assumes that MapReduce is not aware of the physical placement of the data blocks and the data partition is done using the logical view of the data. This causes serious performance issues because a data block that is assumed to be local by a map task may not be entirely local, and the paper proposed several techniques to address these issues. In contrast, in our approach we build the support for scientific data directly in Hadoop where MapReduce is always aware of the physical distribution of the data and does not have the problems in SciHadoop. Moreover, our approach allows scientific users to conveniently use existing big-data tools to work with scientific data. Users can program MapReduce jobs to process scientific data in the same manner as other types of data. Users can also use popular high-level data services such as Hive to process the scientific data using simple queries, where the handling of data is completely transparent to the users. Therefore, to the best of our knowledge, this work is the first to provide native support of scientific data in widely used big-data systems. The rest of this paper presents the details of our approach.

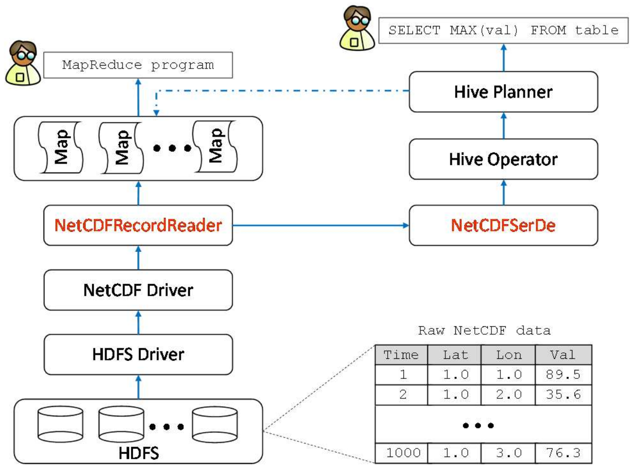

Figure 1. Architecture of the proposed approach for enabling scientific data storage and processing in big-data systems

III. Design and Implementation

A. Enabling Hadoop to Support Scientific Data

Hadoop employs parallel map and reduce tasks to complete a large job. Each map task gets a split of the input data and performs the computation preferably on the node where the data locates. There is an *InputFormat* for each type of input file format, which handles the splitting of input data. Afterwards, the splits are given to map tasks which each retrieves its assigned split using a *RecordReader*.

To support NetCDF data, our design is to introduce new *NetCDFInputFormat* and *NetCDFRecordReader* APIs in Hadoop for MapReduce programs to use for processing the NetCDF data stored on HDFS. Figure 1 illustrates the overall architecture of our approach. To implement these new APIs, we exploit the standard NetCDF library to implement a *NetCDF Driver*, which can understand the internal data structure of the NetCDF data and access the data according to the structure. *NetCDFRecordReader* uses this *NetCDF Driver* to read the records from the NetCDF data, where each record corresponds to a row in the multi-dimensional array data, e.g., the temperatures of different locations (*latitude*, *longitude*) at a specific time. In this way, the extended Hadoop can support all the data structures that are supported by NetCDF.

To ensure good performance of the map tasks that process NetCDF data, *NetCDFInputFormat* needs to split the input data based on the physical distribution of the data so that each map task can get a split that is locally stored for processing. To achieve this, *NetCDFInputFormat* uses the *NetCDF Driver* to find out the offsets of the records and compare them to HDFS block boundaries. With this logical record to physical block mapping, *NetCDFInputFormat* can ensure that the records in a split largely falls under the same HDFS block so that the map task that is assigned the split can find most of its data locally.

Finally, our approach also employs an optimization for processing large NetCDF dataset. Big-data systems process large volumes of data in bulk, and are thus particularly sensitive to I/O efficiency in its design. Therefore, rather than reading a variable value at a time from the multi-dimensional array stored in NetCDF, the *NetCDFRecordReader* uses a single read operation to retrieves a number of variables, e.g., an entire row of values from the multi-dimensional array. Experiments have confirmed that this optimization can speed up the data processing by up to 500 times compared to reading one valuable value at a time.

B. *Enable Hive to Support Scientific Data*

Hive is a data warehousing solution which is built on top of the Hadoop framework [1]. Users can query data stored in HDFS using SQL-style declarative language. Hive would process and generate a plan, which include a set of MapReduce jobs to be executed on Hadoop. Hive allows scientists to conveniently query the data without having to handcraft the MapReduce application which is a lot more difficult and time-consuming. For example, a geoscientist can use a simple query such as "`SELECT MAX(temperature)FROM table`" to find out the highest temperature of a dataset without writing a single line of code. Therefore, we also extend Hive to directly support the use of NetCDF data and provide *transparent* support to scientific users who wish to use queries to process big data.

Every query being submitted by users would be transferred into an execution plan by *Hive planner*, which represents a set of map and reduce tasks that need to be executed in a specific order. The plan consists of multiple nodes which are connected by directed edges. Each node represents a map or reduce task which is responsible to execute a specific *Hive Operator* on the input data. Hive has a set of operators to be executed on the input data, such as filter, group by, join, etc. *Hive driver* receives this plan and submits map and reduce tasks to Hadoop. These tasks use two important sets of classes in order to process data, *InputFormat* and *SerDe*. *InputFormat* is responsible for reading data and passing key-value pairs to the map function. *SerDe* stands for SerializerDeserializer, which transforms the output of *RecordReader* into a column oriented format which the Hive operators can use for processing. Each map task uses *InputFormat* to retrieve the input, uses the corresponding *SerDe* to transform the data, and then executes the specified operator on the data. Therefore, to enable Hive to handle NetCDF data stored on HDFS, we need to create a new *SerDe* for NetCDF.

This *NetCDFSerDe* needs to convert every variable value from the multi-dimensional array stored in NetCDF into a row for Hive, so that users can query based on the different dimensions of the value, e.g., temperature, latitude, longitude, and time for data in a 3-dimensional array. However, as mentioned earlier, our *NetCDFRecordReader* produces a bulk of values at a time for better performance. To support this optimization, we change the architecture of map tasks so that each map task is able to use *NetCDFRecordReader* to retrieve a bulk of values in one shot and invoke *NetCDFSerDe* to transform all these values into rows of the table for the Hive operators to process. Consequently, the performance of Hive queries can also be much improved when processing NetCDF data.

	SQL Query
Query 1	`SELECT AVG(val) FROM TABLE`
Query 2	`SELECT MAX(val) FROM TABLE`
Query 3	`SELECT SUM(val) FROM TABLE`
Query 4	`SELECT SUM(val)FROM TABLE WHERE lat > 50.0`

Table 1. Benchmark queries

IV. EVALUATION

A. *Setup*

The experimental evaluation was done on a cluster of nine nodes, each with two six-core 2.4 GHz AMD Opteron CPUs, 32GB of RAM, and two 500GB 7.2K RPM SAS disks, interconnected by a Gigabit Ethernet switch. All the nodes run the Debian 4.3.5-4 Linux with the 3.2.20-amd64 kernel and use EXT3 as the local file system. The evaluation uses Hadoop 2.5.2 and Apache Hive 1.2.0. One node serves as the NameNode and the others as DataNodes. HDFS block size is defaulted to 128MB, and replication level is three. Each Hadoop map task's resource usage is set to 1 CPU core and 1024 MB memory.

We compare the performance of our approach to the traditional CSV-based approach which converts the NetCDF data to CSV format before storing and processing the data using Hadoop. The dataset represents typical geoscience data which contains a set of temperatures of certain locations (latitude and longitude) at certain times. We consider four commonly used queries as the benchmarks, which are listed in Table 1. *Query* 1, 2, and 3 gets the average, maximum, and sum of the entire dataset, respectively. Query 4 gets the sum of a subset of the data.

B. *Query Performance*

The first set of experiments compare the query execution time of our proposed approach (*Proposed*) to the traditional CSV-based approach (*CSV*). We consider all the four different queries for processing a NetCDF dataset of different sizes, from 2GB to 100GB. Note that in the CSV approach, the NetCDF data need to be converted into CSV format first. In these experiments, we assume the data are already converted and loaded into HDFS. We will evaluate the overhead of data conversion and loading in next subsections.

Figure 2 shows the performance comparison for *Query 1*. The results show that our approach is faster than the CSV approach by 265% for 2GB input, 402% for 20GB input, 906% for 50GB input, and 983% for 100GB input. Similarly, our approach also substantially outperforms the CSV approach for the other queries, by up to 12-fold, as shown in Figures 2, 3, and 4. We attribute this significant improvement to two factors: first, the CSV approach greatly increases the size of data which

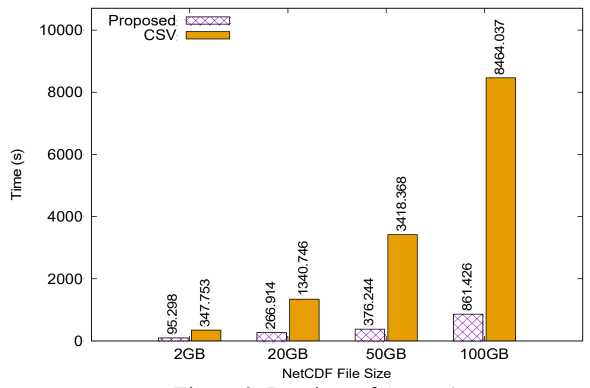

Figure 2. Runtime of Query 1

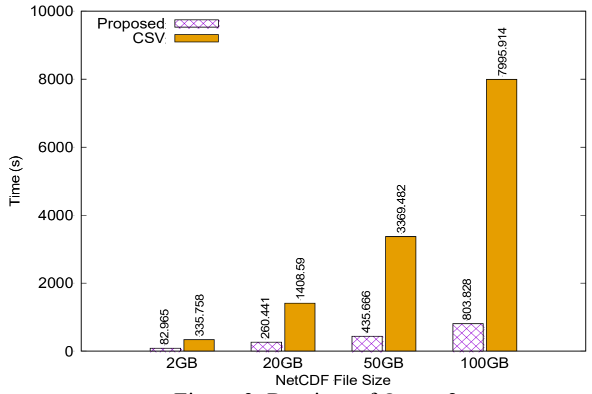

Figure 3. Runtime of Query 2

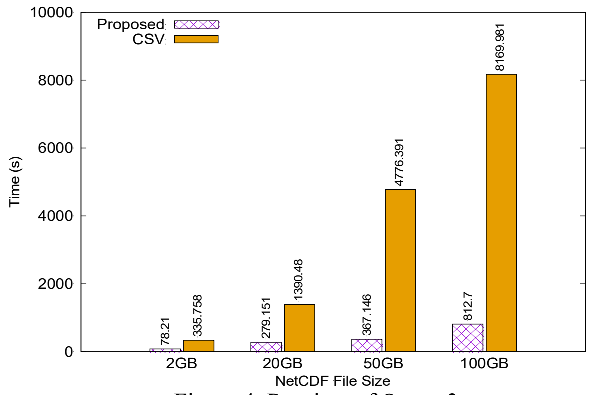

Figure 4. Runtime of Query 3

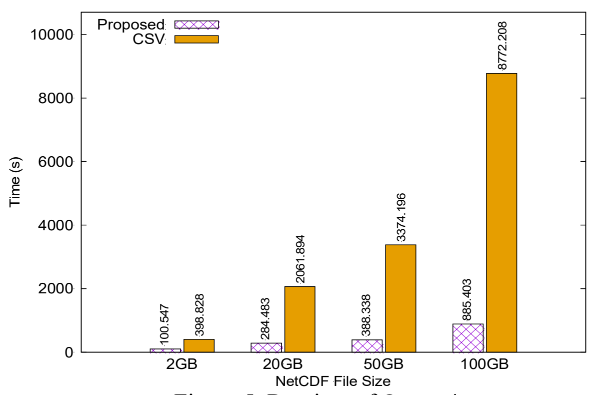

Figure 5. Runtime of Query 4

requires more I/Os and more map tasks to process, both increasing the time required for executing a query; second, the optimization that we made to process records in batch in both Hadoop and Hive also makes our approach much more efficient than the traditional approach. In comparison, in the CSV approach, Hive processes only one line of the CSV data at a time.

C. Conversion from NetCDF to CSV

Another significant source of overhead of the CSV approach is the time required to convert the NetCDF data into the CSV format. Figure 5 shows the time that it takes to convert NetCDF data with different sizes into CSV file format, which is about 9 minutes, 86 minutes, 4 hours, and 7 hours for 2 GB, 20 GB, 50 GB, and 100 GB respectively. In all three cases the conversion time is even more than the query execution time.

D. Importing Data into HDFS

Larger dataset requires longer time when imported into HDFS. Figure 6 compares the time required to import the NetCDF data and the converted CSV data into HDFS. By importing the raw NetCDF file into the HDFS instead of using CSV, we are able to reduce the copy time extensively, by over 80% for the different input sizes.

E. Total Processing

Putting everything together, the end to end time to process a NetCDF dataset is the sum of the conversion time (only for the CSV approach), the importing time, and the query execution time:

Total Processing Time = Conversion Time + Importing Time + Execution Time

Figure 8 compares the total processing time for Query 2 in our approach and the CSV approach with different input sizes. Overall, our approach is faster than the CSV approach by 817.99%, 1598.31%, 2001.23%, and 2042.28% for the NetCDF data of 2GB, 20GB, 50GB and 100GB, respectively. It also achieves the same level of improvement for the other queries.

F. Space Overhead

Storage space consumption on the big-data system is another aspect to compare these two different approaches. Because our proposed approach works directly on the raw NetCDF data, it does not incur any space overhead. In contrast, the CSV approach requires about 524.9% more space after converting NetCDF data to CSV format, e.g., 2GB NetCDF file consumes around 12GB space after conversion. Therefore, our approach also saves space usage substantially. Because data need to be replicated at least three times on HDFS for reliability, the CSV approach would quickly run out of space for larger datasets which in comparison can still be supported by our approach.

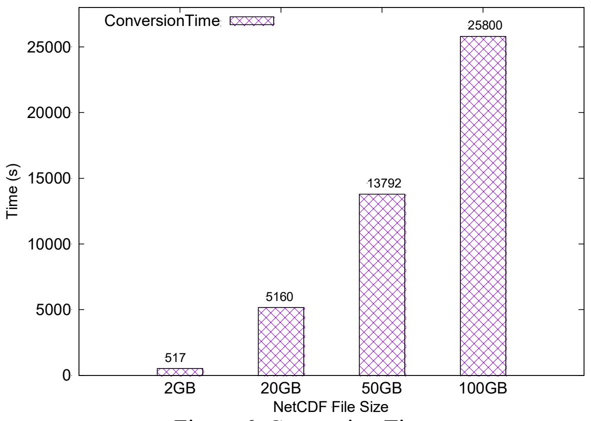

Figure 6. Conversion Time

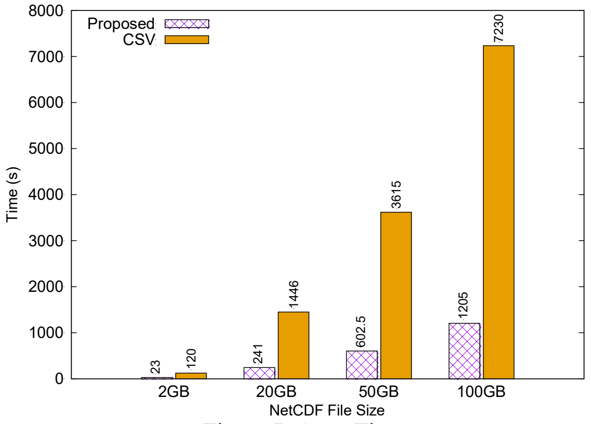

Figure 7. Copy Time

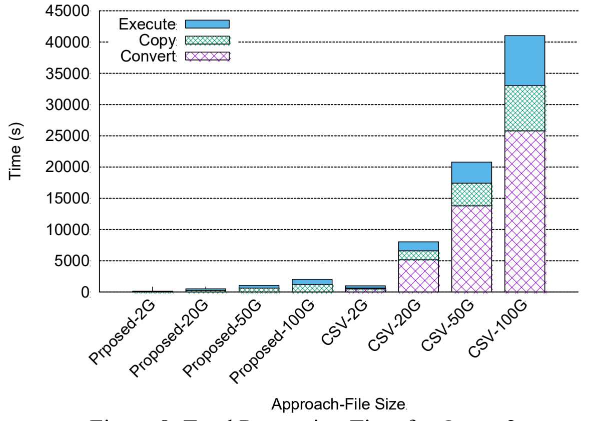

Figure 8. Total Processing Time for Query 2

V. Conclusions and Future Work

This paper presents an approach to enabling big-data systems to support the storage and processing of scientific data. It bridges an important gap between the self-describing data commonly used by scientists for data distribution and sharing and the big-data systems which are increasingly important to scientific productivity. Based on this approach, we have extended two important and widely used big-data systems, Hadoop and Hive, to support scientific data. Users can write MapReduce programs using convenient new APIs to process NetCDF data stored HDFS. They can also use the extended Hive to transparently process NetCDF data using standard queries. Our experiment results obtained from typical queries on a geoscience dataset show that this new approach substantially outperforms the traditional CSV-based approach.

While in this paper we focused on enabling Hadoop and Hive to support scientific data, our approach is also applicable to other emerging big-data systems such as Spark [15]. Based on our current results, we believe that it requires only similar extensions in Spark to support the storage and processing of scientific data stored on HDFS, which will be considered in our future work. In addition, we will also consider the support for the latest self-describing formats such as NetCDF4 and HDF5. Although our approach is also applicable to these new formats, the implementation may be more involved and require additional implementation efforts due to the more complex internal structure of these formats. Finally, we will study data formats optimized for big-data processing (e.g., Orc [8] and Parquet [9]) and explore the possibility of improving scientific data formats such as NetCDF and HDF for big-data systems.

VI. Acknowledgement

This research is sponsored by the National Science Foundation CAREER award CNS-1253944, the Department of Defense award W911NF-13-1-0157, and a gift from VMware Inc. S. Zhou is supported by NASA AIST 14 project.

References

[1] Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” In Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation – Volume 6, OSDI’04, page 10, Berkeley, CA, USA, 2004.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google File System,” In ACM SIGOPS Operating Systems Review, 2003, vol. 37, pp. 29–43.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” in IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.

[4] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, “Bigtable: A Distributed Storage System for Structured Data,” In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation – Volume 7, OSDI’06, pages 15–15, Berkeley, CA, USA, 2006.

[5] HBase. http://hbase.apache.org.

[6] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy, “Hive: a Warehousing Solution over a Map-Reduce Framework,” In Proceedings of VLDB Endow. 2, 2 (August 2009), 1626-1629.

[7] R. Rew and G. Davis, “The Unidata netCDF: Software for Scientific Data Access, ” Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography and Hydrology, Anaheim, CA, February 1990.

[8] Introduction to HDF5, https://www.hdfgroup.org/HDF5/doc/H5.intro.html

[9] ORC Files. https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC

[10] Apache Parquet: https://parquet.apache.org

[11] Joe B. Buck, Noah Watkins, Jeff LeFevre, Kleoni Ioannidou, Carlos Maltzahn, Neoklis Polyzotis, and Scott Brandt, “SciHadoop: Array-Based Query Processing in Hadoop”, In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11). ACM, New York, NY, USA, Article 66 , 11 pages.

[12] Apache Parquet, https://parquet.apache.org/

[13] Earth System Grid, https://www.earthsystemgrid.org/about/overview.htm

[14] NetCDF Utilities, https://www.unidata.ucar.edu/software/netcdf/docs/netcdf/NetCDF-Utilities.html#NetCDF-Utilities

[15] Apache Spark, http://spark.apache.org/