2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)

Visualization and Adaptive Subsetting of Earth Science Data in HDFS

A Novel Data Analysis Strategy with Hadoop and Spark

Xi Yang*, Si Liu*, Kun Feng*, Shujia Zhou† and Xian-He Sun*
*Department of Computer Science, Illinois Institute of Technology, Chicago, USA
{xyang34, sliu89, kfeng1}@hawk.iit.edu, sun@iit.edu
† Northrop Grumman Information Technology, McLean, VA
shujia.zhou@ngc.com

***Abstract*—Data analytics becomes increasingly important in big data applications. Adaptively subsetting large amounts of data to extract the interesting events such as the centers of hurricane or thunderstorm, statistically analyzing and visualizing the subset data, is an effective way to analyze ever-growing data. This is particularly crucial for analyzing Earth Science data, such as extreme weather. The Hadoop ecosystem (i.e., HDFS, MapReduce, Hive) provides a cost-efficient big data management environment and is being explored for analyzing big Earth Science data.**

Our study investigates the potential of a MapReduce-like paradigm to perform statistical calculations, and utilizes the calculated results to subset as well as visualize data in a scalable and efficient way. RHadoop and SparkR are deployed to enable R to access and process data in parallel with Hadoop and Spark, respectively. The regular R libraries and tools are utilized to create and manipulate images. Statistical calculations, such as maximum and average variable values, are carried with R or SQL. We have developed a strategy to conduct query and visualization within one phase, and thus significantly improve the overall performance in a scalable way. The technical challenges and limitations of both Hadoop and Spark platforms for R are also discussed.

***Keywords*—Visualization; R; MapReduce; Hadoop; Spark**

I. Introduction

Ever-increasing High-Performance Computing (HPC) capabilities greatly accelerate scientific discovery. For example, higher-resolution Earth Science (e.g., climate and weather) simulation can be performed with a longer period of time. Consequently, simulation output data can be easily over TeraBytes, which poses a significant challenge for conventional data analysis (e.g., visualization, diagnosis, and subsetting) tools based on a single node computer [1].

Earth Science researchers typically use visualization of whole simulation domain to identify the interested events such as the centers of hurricanes or thunderstorms. However, those events are dynamic. Hence, an efficient and scalable analysis platform for adaptively subsetting data out of a huge amount of data is highly desirable.

In the past few years, MapReduce [2] has been successful in dealing with big data problems and Hadoop MapReduce framework [3] is the most popular big data ecosystem. It features easy programming, transparent parallelism, and fault tolerance on commodity machines. The in-memory computational engine, Spark [4] [5], alleviates expensive disk I/O for storing intermediates result, significantly improves the performance, especially for interactive and iterative computations. Spark provides rich APIs, including MapReduce, for efficient programming. Nowadays, in the so-called 'post-Hadoop' era, the Hadoop Distributed File System (HDFS) [6] [7] is still very powerful to support big data processing in a cost-efficient way, managing massive data in the Hadoop data lake, and is the most popular storage solution to the Apache Big Data Stack (ABDS) [8]. Furthermore, the MapReduce paradigm is also developed and applied to HPC [9] [10] and interactive and real time problems. It has been widely adopted in scientific researches, such as data mining, graphic processing, and genetic analysis.

There are several researches exploring MapReduce paradigm on image plotting [11] and animation generation. Moreover, to address the big data challenges, a hybrid programming model was proposed to potentially exploits the merits of multiple programming models [12].

The current implementation of MapReduce is tightly-coupled with key-value pair processing in terms of programming APIs, transparent parallelism support, and optimized I/O system. Consequently, it cannot be directly and efficiently applied for image plotting. Earth science researchers often use R [13] for data analysis and visualization. However, R is not designed to exploit parallelism and data locality. The extended R interfaces of the Hadoop ecosystem, RHadoop [14] and SparkR [15], lack efficient strategies to parallelize the R analytic workloads, especially for adaptively subsetting.

This paper investigates how R can utilize a MapReduce-like strategy to analyze data in a scalable way, especially for Earth Science data. It identifies and addresses several challenges in utilizing MapReduce for data diagnosis and visualization. The contributions include:

- We demonstrate how to encapsulate R image plotting function into MapReduce paradigm and transparently and adequately align tasks to data.

- We identify and prevent both inefficient I/O and redundant computation for R visualization. Moreover, we integrate data query and visualization within the same procedure to improve the efficiency of diagnosis and data subsetting.
- Our data analytics solutions are built on top of Hadoop and Spark frameworks. They have been tested and evaluated with the NASA cloud-resolving model simulation data [1]. The experimental results show the effectiveness and efficiency of our solution.

The organization of this paper is as below. The related work is discussed in Section II and the motivation for this study is presented in Section III. The challenges in utilizing R and MapReduce for visualization are addressed in Section IV. Our proposed methods are evaluated in Section V and summary is in Section VI.

II. Related Work

Big Earth Science data stored in HDFS desires a scalable distributed visualization tool. R [13] is a powerful visualization and statistical analysis tool. Although IDL [16] is a powerful commercial visualization tool, it is not easy to be parallellized. In our previous study, we visualize the HDFS data in the formats of both CSV and NetCDF with IDL on a single node [1]. Opass [17] enables a MPI-based visualization application, ParaView [18], to access and visualize scientific data resided in HDFS. It optimizes workload balance and data locality. On the contrast, MapReduce application, greedily pursuing data locality and parallelism, is an alternative and promising solution, particularly for multi-tenant Hadoop environment that highly tolerates embarrassingly data accesses, and transparently alleviates the common speculative tasks [19].

MapReduce has been widely adopted in distributed big data analytics and applicable to parallel visualization. However, users have to implement complex and dedicated visualization algorithms integrated to a MapReduce model. For example, there is a study [20] that utilize MapReduce to conduct display algorithms including mesh rendering and isosurface extraction. In this study, we accomplished parallel distributed visualization in RHadoop [14] and SparkR [15], R interfaces of two open-source MapReduce implementations Hadoop and Spark, respectively. RHadoop is a collection of multiple R packages that allow users to manage and analyze data in R language with Hadoop. The recently released SparkR package supports programming in R on top of the Spark framework.

Hadoop-GIS [21] and SpatialHadoop [22] provide indexing to efficiently process spatial data in HDFS. Hadoop-GIS is a high-performance spatial data warehousing system to address spatial queries and high computational complex queries. SpatialHadoop uses SpatialRecordReader to read multiple records for further processing, whereas our approach bypasses the RecordReader and directly reads and processes the data. HadoopViz [11] is a Hadoop-based visualization platform for visualizing spatial data. It processes spatial data via its Mapper to read the data logically grouped for visualization on the indexed data. However, it cannot be integrated with R programming on Hadoop and Spark platforms at this time. Our approach combines query and image plotting within one MapReduce-like procedure and implements the strategy in both Hadoop and Spark platforms.

III. Motivation

R is a popular programming language for statistical data analysis and visualization. Image plotting in R often requires loading the entire dataset into memory. This is because image plotting is a two-phase procedure, data loading and processing. Though the capacity of memory has been increasing, it is still challenging to cache all the input data for creating images. Using a fat node, which has a large size of memory and a large number of cores, seems to be a good solution to overcome the memory capacity issue. However, this approach may result in considerable data movement in the cluster as the centralized visualization node needs to aggregate all the target data distributed across the storage nodes. Furthermore, it does not fully explore the parallelism of image plotting in a distributed environment. For a large amount of data stored in HDFS environment, it is more efficient to move computation than data to avoid network traffic and latency. Therefore, visualizing a large amount of data requires a solution efficiently utilizing both a memory and computation. The MapReduce paradigm divides data into sub-datasets and conducts independent tasks on those sub-datasets in parallel in the Map phase and further processes the intermediate results in the Reduce phase. This could be a good solution for visualizing a large amount of data.

Our study investigates how to use R for visualization and adaptive subsetting in Hadoop and Spark platforms. In particular, we explore how to use RHadoop and SparkR to efficiently visualize and adaptively subset Earth Science data. This is due to the fact that in Earth Science data analysis, especially for extreme weather events such as hurricanes and thunderstorms, only the areas near the extreme events such as the centers of hurricanes and thunderstorms are more important for detailed analysis. Therefore, it is highly desirable to develop a system for visualizing and adaptively subsetting data in an efficient as well as scalable way.

IV. Design and Implementation

A. Solution Overview

Figure 1 presents the proposed parallel visualization strategy in MapReduce paradigm. A visualization job consists of multiple visualization tasks running in parallel. An existing visualization function is wrapped into a Map task such that each task processes a file and generates a visualized frame. The timestamp and the generated image compose a key-value pair. Moreover, the statistical information can be collected from each data frame, and encapsulated together with the plotted image in the same phase. Visualized frames with the same timestamp rather than the original data are shuffled onto one node for further manipulation such as combination. In a cloud resolving model simulation, for instance, a surface layer data (e.g., rainfall) as an input file has about 3 GB in the CSV

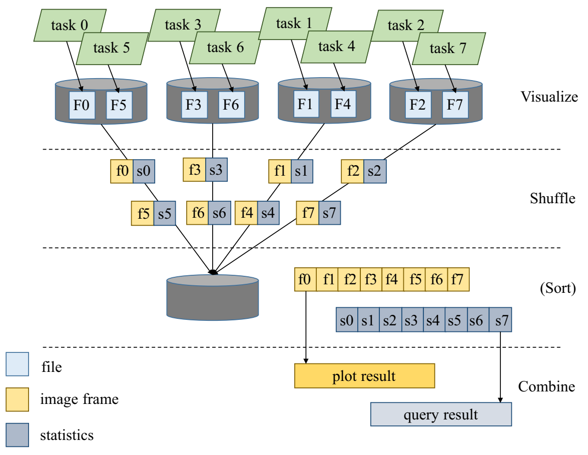


Fig. 1: Illustration of parallel visualization strategy. A visualization job consists of multiple visualization tasks. Each task processes a file and generates a visualized frame. Frames in one job will be shuffled onto one node. After sorting and combining, the frames will be output in a continuous stream.

Algorithm:

```
1   map (k, .){
2       id ← parse split reference for target data
3       data frame ← read target data
4       statistics ← query on the data frame
5       image frame ← plot (data frame)
6       value ← {image frame, statistics}
7       return <id, value >
8   }
    # shuffled key-value pairs are already sorted by internal sort
9   reduce (k', v'){
10      for every N frames, plot or combine as a result
11  }
```

Fig. 2: Visualizing and statistically analyzing data with the MapReduce paradigm.

format, and the corresponding plotted frame is only about 6 MB with the 1200×1200 resolution, which is significantly smaller than the original data size. Thus, the network traffic is greatly reduced as it greedily takes advantage of data locality. After sorting and combining at the Reduce phase, the frames are output in a continuous stream.

Figure 2 presents the pseudo code for visualization and statistical analysis. Line 2 reads and parses the split reference in order to learn the data range that the current task needs to process. Next, line 3 is to directly read from HDFS and line 4 is to conduct a query on the data frame. The result is converted into a matrix for image plotting. Each generated frame has a unique id, which is parsed from line 2. The mechanism of such split parsing will be discussed in next subsection. All sorted key-value pairs within a group of keys are shuffled onto a Reducer for a further combination according to their keys. Reduce function combines frames for every certain number of frames as the expected data layout. The Reduce phase starts after the internal sort phase is completed. Thus, the ids generated in the Map phase have an alphabetically ascending order. For example, all the images generated from file 1, such as 1-1, 1-2, 1-3, 1-4, are reduced by one Reduce task. Thus, there is no need to implement a customized partitioner (e.g., hash function in MapReduce to assign related intermediate results to Reducer).

Figure 3 shows the partial results of visualizing the NASA cloud-resolving weather simulation output. As shown in Figure 3(a), image frames are collected and merged into a collage image. An animation of those images can also be created. In the same procedure, the statistic information such as the maximum and average rainfall can be obtained and used in the subsequent adaptively subsetting. Figure 3(b) shows a red square centered at the maximum rainfall value.

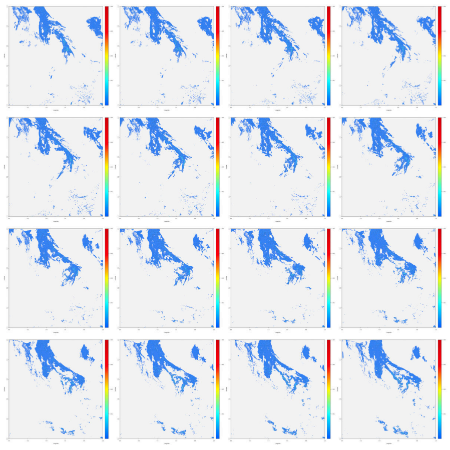
In short, we firstly use Hadoop MapReduce for distributed computing (e.g., find the maximum value of rainfall) on the target data in HDFS, and then use R for image plotting on each node to achieve parallelism. Table I lists the main R packages that we used for visualization in both Hadoop and Spark. In the later section, we will apply such a visualization strategy onto Spark platform in order to enhance performance and overcome the limitation of reading data from Hive tables. In addition, we will integrate simple SQL query, diagnosis and visualization within one procedure to achieve efficiency.

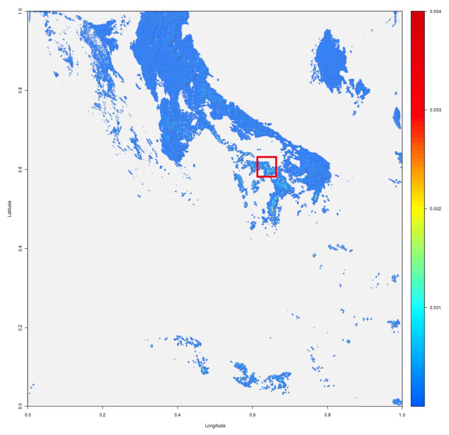
TABLE I: Main Packages

Package	Version	Description
R	3.2.2	
rhdfs	1.0.8	Connect with HDFS
rmr2	3.3.1	R in Hadoop MapReduce
rhive	2.0-2.0	Connect with Hive
Cairo	1.5-9	Graphics library for creating bitmap
doParallel	1.0.10	Provides a parallel backend method
SparkR	1.6.1	Integrated package in Spark1.6
sqldf	0.4.10	SQL on data frame

B. *Input Format*

A Comma Separated Values (CSV) [23] file stores data in plain text. Usually, records are separated as lines in CSV file. It is a popular format for scientific data. It can be recognized by Hive [24] or Impala [25], for query operations and other scientific data processing tasks. RHadoop provides a read function `hdfs.line.reader()` for reading CSV data in HDFS line by line. Due to the inefficient read method, the latency is considerably long. Besides the poor I/O performance, the major cost for processing CSV data is the preprocessing phase that converts a CSV file into a matrix for image plotting record by record. Alternatively, Our solution is to first read all the target CSV data into memory as it is in a large I/O request, and then converts CSV data to a matrix. Such an I/O efficient approach significantly reduces the overhead in processing CSV files. The scientific simulation data in a CSV file often have a regular and predictable pattern in size. For example, a group of 1.56 million records (lines) is used for one image in our current test data sets. Hence, an application can deploy multiple concurrent tasks to plot different aligned images in parallel with the knowledge of target data sizes and ranges. As a result, an image with a large number of pixels can be created in a fine-grained and memory-efficient manner. Hence, both our RHadoop and SparkR implementations adopt such a strategy.


(a) Collage images (From 9 AM, 2014-04-28 to 12 AM, 2014-04-29)



(b) An image with a highlighted area for an interested event (the heaviest rainfall)

Fig. 3: Visualization of rainfall in a cloud resolving model simulation with R and MapReduce. (a) 16 images are combined into one frame. (b) The interested event, the heaviest rainfall, is identified and highlighted.

C. RHadoop Implementation

1) I/O and Computation Optimization: We firstly utilize Hadoop MapReduce for the visualization applications. A conventional MapReduce application processes data records in key-value pairs. Hadoop MapReduce optimizes I/O by reading key-value pairs in a dedicated and consecutive loop. Hence, computation in Map function and I/O can be overlapped [26]. However, visualization applications in R require to load all the needed data before plotting images. In this case, computation and I/O can hardly overlap. Furthermore, RecordReader reads input in key-value pairs which generates frequent seek or scan operations to determine the boundary of input units. Such a procedure introduces numerous expensive memory copy and redundant I/O operations. Therefore, we have to customize this visualization procedure to improve performance. The task reads the target data directly from HDFS rather than from the embodied RecordReader in the Map function. Consequently, I/O efficiency could increase as a result of those two operations: 1) avoid frequent scan operations for sync-markers and 2) use one large read request rather than many small read requests issued by RecordReader. To realize these operations, the visualization task is required to know the target data through the specification of the path and the offset range of dataset.

2) Alignment: The conventional Hadoop MapReduce aligns the input data and ensures data integrity by RecordReader's predefined sync-markers. However, in a visualization applications, multiple continuous lines or a specific range of data are the input for a single frame. That means such an atomic input unit is determined by the logical data range rather than arbitrary sync-markers (such as 'new-liner' or 'tab'). Hence, there is a lack of a valid sync-markers to align the input units. As RHadoop is tightly-coupled with Hadoop implementation, we bypass RecordReader to read the target data. The split alignment depends on predetermined data range. In short, the loaded data are exactly the input for plotting a single image frame. This requires to rebuild the mapping between the logical view and the physical data since the conventional MapReduce assigns Map tasks based on splits, the size of which is the same as the size of blocks of an input file by default. We adopt a similar strategy in Spark platform to align input for tasks.

D. SparkR Implementation

Hadoop naturally supports the MapReduce paradigm. However, the recently released SparkR (Spark version 1.6.1), which is built on top of Spark, emphasizes high-level user interfaces to provide integrated data computation method. SparkR discards low-level interfaces (i.e., the public interfaces such as Map and Reduce). Therefore, we cannot apply the above approach on a visualization application that improves performance through I/O and data alignment. Spark is based on Resilient Distributed Dataset [27] (RDD) to achieve memory based fault tolerance, and recently introduced DataFrame for better integration with data frame based computations. Data within the Spark ecosystem are represented as RDD or DataFrame, flowing from one job or component to the other. A data flow efficiently within the ecosystem. However, only after conducting the `collect` operation, DataFrames will be collected and decoded into a non-Spark-specific data format, on which other programming models, such as R data frame, can operate. Furthermore, the `collect` operation will only be performed by the centralized driver on the hosting node, which greatly sacrifices parallelism and introduces tremendous network traffic for later non-typical Spark tasks.

For example, a user can easily and efficiently obtain the SQL results, however, the subsequent operations afterward on that resultant dataset will trigger the `collect` operation at the driver of the hosting node. That causes data shuffling and limits parallelism. It is commonly recognized that collecting a large amount of data is a bad practice and should be avoided.

Hence, we distribute the references to tasks rather than directly input data. The application follows the same procedure as Hadoop implementation: reads data directly from HDFS using the hint which indicates the data range processed by a specific task. Thus, the input data will be read as a regular R data frame rather than loaded into RDD or Spark DataFrame. In this way, the subsequent image plotting operates on those data items. We implement MapReduce in Spark for image plotting. It consists of three stages: (1) `SparkR:::parallelize` to distribute input references, (2) `SparkR:::lapply` to conduct Map phase, and (3) collect results from Map phases.

E. Image Combination

The image combination phase of visualization performs in the Reduce phase of Hadoop MapReduce. Before the Reduce function starts, all key-value pairs are sorted according to specific keys. The intermediate images for a combined image are loaded into memory before the combination procedure with Montage [28], a popular image combining tool.

1) Collage: Image collage can help a user to view the differences among multiple images and find the interested events. For a large image (e.g., ultra-high resolution), it is efficient to first create sub-images and then combine them into one. Since the PNG format supports image combination, we use the PNG format in our image processing. All values (images) associate with the same key are grouped and combined into one image. In the combination phase, relevant sub-images are merged. Since the keys are already sorted, sub-images are stored consecutively and can be merged as specified.

2) Animation: An animation or movie helps to visualize time series events such as thunderstorms. Our implementation generates images in either gif or HTML format.

3) Combine image in parallel: We find that the image combination phase can be time-consuming with Montage. The total image combination time is proportional to the number of images. Therefore, we develop two parallel image combination methods to improve the performance.

The first method is a MapReduce-like scale-out strategy, utilizing parallel computing power in a distributed environment. It first stores the image into its target directory in HDFS according to a certain attribute such as the layer id. After that, a wave of tasks is launched. Each task reads multiple images from its corresponding directory, and combines images. For example, all images of one layer with different timestamps are stored in a directory with respect to that layer. In this way, we can efficiently create an animation or collage image for each specific layer in parallel. Finally all the processed images are collected onto one node. The intermediate images are temporarily dumped into HDFS so that the following image combination can retrieve the data. This approach introduces disk I/O overhead. Since the image combination phase is a computing intensive operation, overall performance could gain through parallelism if one node does not have sufficient computer resource.

The second method is a multi-threaded solution that adopts 'for-each' function provided by 'doParallel' R package [29] to parallelize workload. It exploits multi-core computing resource at the node with the driver. It collects all the plotted images onto that node (the driver resided node) for the final image combination so as to avoid extra I/O and across-node communication.

F. Hive Tables

Hive table provides a user-friendly way of accessing and processing data through SQL-like query. However, we find that it is not effective to use R to create images with Hive query.

1) Plotting images from the query results with RHive: The RHive package connects R instance with Hive and enables Hive query in R language. However, we find that the performance of obtaining the data, which can be plotted with R, via Hive query for a large dataset is very poor. If the resultant dataset is larger than 2 MB, the data will be stored back into HDFS as indicated in its source code. Even though we force to store the data into memory, the performance is still very poor. This is due to the fact that the R code needs to parse Hive resultant object in the manner of attribute by attribute and line by line. In addition, the Hive resultant object cannot be exported to a local file system because the create-table-as-select (CTAS) operation for an external table is forbidden in Hive [30]. Hence, plotting image from the query results via RHive is not effective.

2) Plotting images from the query results with SparkR: SparkR in the version of Spark-1.6 integrated SparkSQL [31], which provides interaction with Hive and conducts query onto Hive tables in R language. The recently introduced Spark DataFrame APIs [5] provide efficient and user-friendly operations on the tabular datasets. We find that the query results are in a Spark DataFrame. As discussed above, converting such a DataFrame to an R data frame needs the 'collect' operation at the driver hosting node. However, R image plotting function needs to operate on such a matrix format at the executor hosting node, which can be different from the driver hosting node. As a result, this data conversion procedure could significantly degrade parallel performance of image plotting.

G. An Integrated Strategy of Query and Image Plotting

Since plotting an image needs to load entire target data frame and performing statistical calculation via query also need to read and filter on those data items, we propose to integrate these two operations in the same phase to minimize. the cost of reading data.

We use R to perform the interested statistical calculations, and use 'sqldf' [32] package to conduct simple SQL query directly on R data frames. After that, the calculation results and the generated images are encapsulated and shuffled to the corresponding nodes for further processing. At the end of data processing, the application collects global calculation results and combines images. This approach can be used to efficiently identify some interested events in the large data frame, such as those areas with the temperature or the rainfall above a predefined threshold and indicate them in the final image.

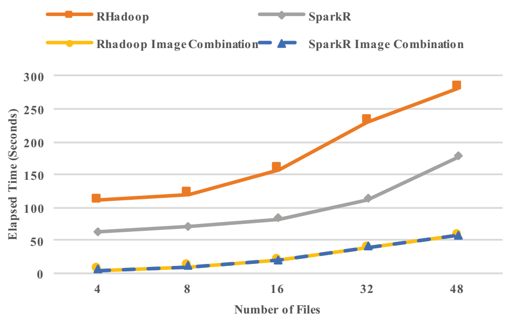


Fig. 4: Performance of implementations with RHadoop and SparkR. The first layer (surface) of a rainfall data file in the CSV format is visualized on 8 nodes at HEC cluster. The number of input files varies from 4 to 48.

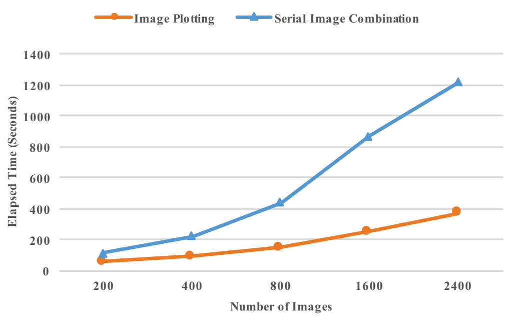


Fig. 5: Performance of implementation with SparkR. All 50 layers of a rainfall data file in the CSV format are visualized on 8 nodes at Chameleon Cluster. The implementation consists of two phases: distributed image plotting and centralized serial image combination operation.

Those areas are also able to subset for detailed analysis. We list a few queries with typical statistical calculations in Table II and will evaluate their performance in Section V.

We can also support tiled image plotting. Each task only plots a tile of a frame and all tiles are stitched together after aggregation. The universal coordinate mapped from longitude and latitude in all tasks keeps the coordinates of highlighted areas are consistent across all tiles.

V. Evaluation

In this section, we will evaluate the performance of Hadoop and Spark implementations. First, we evaluate the image plotting, which consists of two phases: parallel image plotting and combination. Secondly, we examine the performance of query as well as that of integrated query and image plotting. Thirdly, we evaluate scalability and discuss the implementation tradeoffs, including two scaling methods for image combination.

A. Environmental Setup

We conduct our experiments at Chameleon cluster of TACC and HEC cluster at IIT, respectively. Each Chameleon node is equipped with 48 2.67GHz Intel Xeon CPU cores, 128 GB memory, and a 250 GB 7200 RPM ST9250610NS SATA hard drive. Each HEC node is equipped with two 2.3GHz Opteron quad-core processors, 8 GB memory, and a 250 GB 7200 RPM ST32502NSSUN SATA hard drive. We use eight slave/worker nodes by default.

We use the cloud-resolving weather simulation output from a NU-WRF model with a 1250×1250×50 grid of 4km resolution and 48-hour simulation time. Its original output is in NetCDF. Each file has a variable and a time stamp. Its data format has five columns: timestamp, layer, latitude, longitude, and variable. We transform one NetCDF file into one CSV file, which has 3,299,293,700 bytes. In this paper, we will report the experiment results with one variable, rain (QR), in 48 CSV files with about 147.5GB. We configure 1200×1200 as the default resolution for plotting images.

B. Hadoop and Spark Implementations

First, we evaluate the performance of implementation on Hadoop and Spark platforms at HEC cluster. The first layer (surface) of rain files is visualized. As Figure 4 shows, both implementations obtain good scalability as the total number of input files increases. Although Hadoop implementation keeps the plotted images in memory, the platform transparently materializes those intermediate data into local file system at both Map output phase and the end of shuffle phase of Reduce phase. Since Spark implementation eliminates I/O of intermediate data, overall performance is improved. On the HEC platform, our Spark implementation is approximately 2× faster than the Hadoop one. Therefore, we focus on evaluating Spark implementation in the following experiments. Secondly, we evaluate the performance of image combination phase for collaged images and animations. We found that their performances are almost the same. Thus, we use animation generation for the image combination phase function in the subsequent performance evaluation. Because both Hadoop and Spark combine images in memory using the same function, they have almost the same elapsed time in this phase.

C. Image Plotting Phase

The two phases of the Spark implementation, parallel image plotting and centralized serial image combination, are evaluated and their performances are as shown in Figure 5. The parallelized image plotting obtains linear scalability as workloads increase. It takes 373 seconds to plot 2400 images with 1200×1200 pixels from 147.5 GB HDFS-resident data. The elapsed time of the serial image combination operation is proportional to the number of images, which motivates us to parallelize the image combination phase.

We evaluate the scalability in terms of scale up and scale out at Chameleon cluster and at HEC cluster, respectively. Figure 6 shows the scalability of the Spark implementation through varying the number of executor cores on each node, from 12 to 48 cores, as well as varying the number of input files from 4 to 48. In particular, we observed a super linear speedup as the number of files increases and the number of cores increases. We believe that this is due to two facts: (1) full utilization of computation power and (2) good balance of workload. Figure 7 shows the linear scalability of the Spark

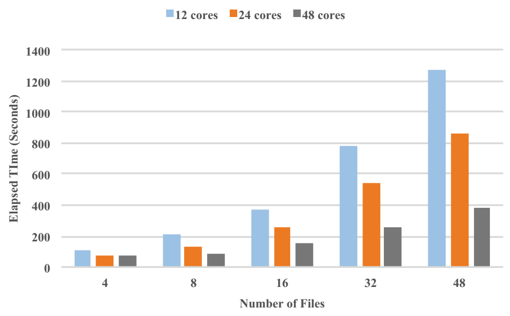


Fig. 6: Performance of image plotting phase as a function of the number of configured cores per node as well as the number of files. The experiments were performed on 8-node Spark platform at Chameleon cluster.

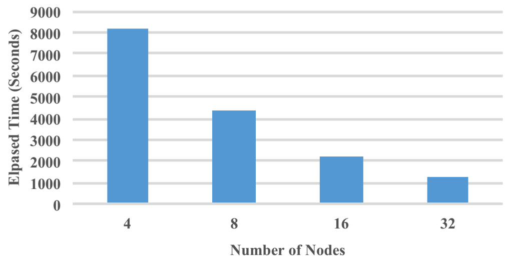


Fig. 7: Performance of image plotting phase as a function of the number of nodes on Spark platform at HEC cluster.

implementation in HEC cluster with varied the number of nodes.

D. Image Combination Phase

Figure 8 presents the combination-phase performance improvement of both the distributed parallel method and the single-node parallel method over the baseline of serial image combination. Both parallel methods efficiently reduce the elapsed time from 1200 seconds to 100 seconds. In this set of experiments, the single node parallel method always outperforms the distributed parallel method. It is because all 48 cores on the single node are utilized to explore parallelism and data locality.

We also evaluate the impact of the resolution of plotted images to overall performance. The number of cores is set to 12 and the total number of input files is set to 32. As Figure 9

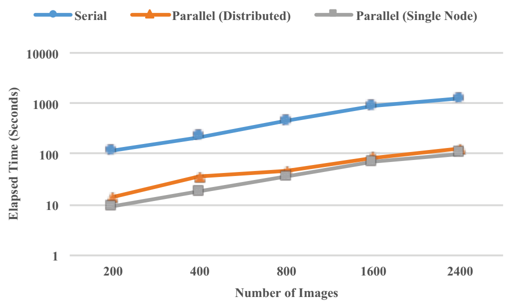


Fig. 8: Performance comparison among centralized serial image combination method (distributed 8-node), and parallel image combination method (single-node), at Chameleon cluster.

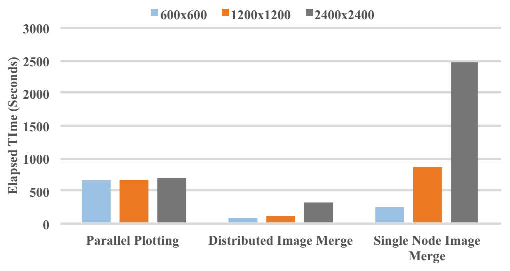


Fig. 9: The impact of image resolutions to the performances of different phases at Chameleon cluster. The resolution varies from 600×600 to 2400×2400.

shows, increasing the resolution of images has little impact on the parallel plotting phase but significantly increases the time of the image combination phase.

How to further improve performance is worth further exploring. Firstly, the performance would be enhanced by using a scale-up fat node for the image combination phase. However, contrary to the performance trends shown in Figure 8, the distributed method always outperforms the single node method. This is because computation power is the bottleneck. Thus, the performance could be improved by exploring parallelism from distributed computing power at the image combination phase. Secondly, such an image combination function can be implemented in various alternative software tools. When such elapsed time of such an image combination task is long, we find that adopting a distributed parallel processing strategy can improve the performance.

E. Query and Adaptively Subsetting

Figure 10 shows the performance of the query-only job, the image-plotting-only job, and the integrated-query-and-image-plotting job. As we vary the number of input files, the elapsed time of the integrated method is a slightly higher than the image plotting. The 'Condition' takes a longer time since it collects more subset data. Overall, these experiments confirm that integrating query with image plotting is an efficient approach to improve I/O and computation efficiency.

VI. Conclusions

This paper proposes an efficient and scalable method to use R and MapReduce on both Hadoop and Spark platforms to visualize, diagnose, and subset data. We transparently parallelize existing R visualization and diagnosis (e.g., statistical calculations) codes and integrate them with SQL query in one procedure, and consequently enable our method to be user-friendly, scalable, and efficient. A user can utilize statistical calculation results to find the interested events to visualize and adaptively subset data and visualize within one job. In addition, we discuss several technical challenges of using RHadoop and SparkR to realize our method, especially in I/O, accessing and manipulating data in Hive tables, and SparkR data frame. The experimental results with Earth Science data show our proposed method is scalable and efficient.

TABLE II: Simple SQL query statements

Labels in Figure 10	SQL statements
Top 1	select * from dataframe where value == (select max(value) from dataframe) limit 1
Top 10	select * from dataframe desc order by value limit 10
Condition	select * from dataframe where value > 0.005
Avg	select avg(value) from dataframe

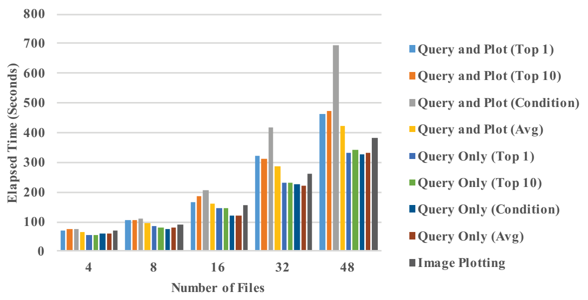


Fig. 10: Elapsed time of query as a function of the number of files at Chameleon cluster. Top 1 and Top 10 mean to collect all the rows (records), with the maximum value and the top 10 values of each layer (frame) in each table, respectively. Condition means to collect all the rows (records) above a certain value (e.g., 0.005). Avg means to collect all the average values from each layer (frame) in each table.

Acknowledgment

Zhou would like to thank the support from NASA AIST 2014 grant. Yang, Liu, Feng and Sun are partially supported by NASA AIST 2014 grant and by NSF under NSF grants CNS-0751200, CNS-1162540, and CNS-1526887.

References

[1] S. Zhou, X. Yang, X. Li, T. Matsui, S. Liu, X.-H. Sun, and W.-K. Tao, "A Hadoop-Based Visualization and Diagnosis Framework for Earth Science Data," in *Proceedings of IEEE International Conference on Big Data (IEEE BigData 2015)*, 2015.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," in *Proceedings of USENIX Symposium on Operating Systems Design and Implementation*, 2004.

[3] Apache Software Fundation. (2014, Mar.) Apache Hadoop Project. [Online]. Available: http://hadoop.apache.org

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," *HotCloud*, 2010.

[5] The Apache Software Foundation, "Spark," https://spark.apache.org.

[6] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache Hadoop Yarn: Yet Another Resource Negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013.

[7] X. Yang, C. Feng, Z. Xu, and X.-H. Sun, "Dominoes: Speculative Repair in Erasure Coded Hadoop System," in *Proceedings of The 22nd annual IEEE International Conference on High Performance Computing (HiPC 2015)*. IEEE, Dec. 2015.

[8] A. Luckow, K. Kennedy, F. Manhardt, E. Djerekarov, B. Vorster, and A. Apon, "Automotive Big Data: Applications, Workloads and Infrastructures," in *Proceedings of IEEE International Conference on Big Data (IEEE BigData 2015)*. IEEE, 2015.

[9] 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*. ACM, 2015.

[10] H. Sim, Y. Kim, S. S. Vazhkudai, D. Tiwari, A. Anwar, A. R. Butt, and L. Ramakrishnan, "AnalyzeThis: an Analysis Workflow-aware Storage System," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015.

[11] A. Eldawy, M. Mokbel, and C. Jonathan, "HadoopViz: A MapReduce Framework for Extensible Visualization of Big Spatial Data," in *Proceedings of the 32nd IEEE International Conference on Data Engineering (ICDE)*, 2016.

[12] P. Wang, H. Jiang, X. Liu, and J. Han, "Towards Hybrid Programming in Big Data," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[13] The R Foundation, "R-project," https://www.r-project.org.

[14] A. Piccolboni, "RHadoop Project," https://github.com/RevolutionAnalytics/RHadoop.

[15] The Apache Software Foundation, "SparkR," https://spark.apache.org/docs/1.6.0/sparkr.html.

[16] Exelis Visual Information Solutions, "IDL project," http://www.exelisvis.com/ProductsServices/IDL.aspx.

[17] J. Yin, J. Wang, J. Zhou, T. Lukasiewicz, D. Huang, and J. Zhang, "Opass: Analysis and optimization of parallel data access on distributed file systems," in *Proceedings of 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2015.

[18] KitwarePublic, "ParaView," http://www.paraview.org.

[19] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*. USENIX Association, 2008.

[20] H. T. Vo, J. Bronson, B. Summa, J. L. D. Comba, J. Freire, B. Howe, V. Pascucci, and C. T. Silva, "Parallel visualization on large clusters using MapReduce," in *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*. IEEE, 2011.

[21] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz, "Hadoop GIS: a High-performance Spatial Data Warehousing System over MapReduce," *Proceedings of the VLDB Endowment*, 2013.

[22] A. Eldawy and M. F. Mokbel, "SpatialHadoop: A MapReduce Framework for Spatial Data," in *Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE)*. IEEE, 2015.

[23] Shafranovich, Y., "Common format and MIME type for Comma-Separated Values (CSV) Files," http://tools.ietf.org/html/rfc4180.html.

[24] The Apache Software Foundation, "Apache Hive Project," https://hive.apache.org.

[25] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs *et al.*, "Impala: A Modern, Open-source SQL Engine for Hadoop," in *Proceedings of the Conference on Innovative Data Systems Research (CIDR'15)*, 2015.

[26] B. Feng, X. Yang, K. Feng, Y. Yin, and X.-H. Sun, "IOSIG+: on the Role of I/O Tracing and Analysis for Hadoop Systems," in *Proceedings of IEEE International Conference on Cluster Computing*, 2015.

[27] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012.

[28] ImageMagick Studio, "Montage," http://www.imagemagick.org/script/montage.php.

[29] Calaway, R. and Weston, S., "doParallel: Foreach Parallel Adaptor for the 'parallel' Package," https://cran.r-project.org/web/packages/doParallel/index.html.

[30] Zhang, Ning, "CTAS support external table (create external table as select)," https://issues.apache.org/jira/browse/HIVE-891.

[31] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark SQL: Relational data processing in Spark," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 2015.

[32] Grothendieck, G., "sqldf: Perform SQL Selects on R Data Frames," https://cran.r-project.org/web/packages/sqldf.