A Data Management Scheme for Micro-Level Modular Computation-intensive Programs in Big Data Platforms *

Debasish Chakroborti[0000−0002−1597−8162], Banani Roy, Amit Mondal, Golam Mostaeen, Ralph Deters, Chanchal Roy, and Kevin Schneider

Department of Computer Science, University of Saskatchewan
{debasish.chakroborti, banani.roy, amit.mondal, golam.mostaeen, ralph.deters, chanchal.roy, kevin.schneider}@usask.ca

Abstract. Big-data analytics or systems developed with parallel distributed processing frameworks (e.g., Hadoop and Spark) are becoming popular for finding important insights from a huge amount of heterogeneous data (e.g., image, text and sensor data). These systems offer a wide range of tools and connect them to form workflows for processing Big Data. Independent schemes from different studies for managing programs and data of workflows have been already proposed by many researchers and most of the systems have been presented with data or metadata management. However, to the best of our knowledge, no study particularly discusses the performance implications of utilizing intermediate states of data and programs generated at various execution steps of a workflow in distributed platforms. In order to address the shortcomings, we propose a scheme of Big-data management for micro-level modular computation-intensive programs in a Spark and Hadoop based platform. In this paper, we investigate whether management of the intermediate states can speed up the execution of an image processing pipeline consisting of various image processing tools/APIs in Hadoop Distributed File System (HDFS) while ensuring appropriate reusability and error monitoring. From our experiments, we obtained prominent results, e.g., we have reported that with the intermediate data management, we can gain up to 87% computation time for an image processing job.

Keywords: Data management · Modular programming · Computation-intensive program · Big data · Distributed environment · Intermediate state.

1 Introduction

With the rapid advancement of Big Data platforms, software systems [2], [3], [5], [9], [16] are being developed to provide an interactive environment for large-scale data analysis to the end users in the area of scientific research, business,

* This work is supported in part by the Canada First Research Excellence Fund (CFREF) under the Global Institute for Food Security (GIFS)

government, and journalism. Big Data platforms such as Hadoop, Spark, Google Data-flow, and so on provide us a high-level abstract interface for implementing distributed-cluster processing of data. Recently, many researchers [4], [5], [6], [7] have focused on developing architectures and frameworks for large-scale data analysis tools utilizing these platforms. Most of these architectural frameworks are adopting workows or pipelines [8], [9], [12] for data analysis to provide flexible job creation environment. Workflows, in general, connect a sequence of interoperating tools to accomplish a task. For example, in order to find features from images of a large crop field, different image processing tools such as *image registration, stitching, segmentation, clustering and feature finding tools* are needed to be connected in order to form a workflow. In such workflow management systems, modularization is important to support scalability and heterogeneity. Thus, a wide range of tools is incorporated where each tool is treated as an independent process or module or service and can be implemented using different programming languages. Traditionally, researchers and data analysts require to run same workflows frequently with different algorithms and models that causes overwhelming efforts even with the moderate size of data by running all of the modules in the workflows. Utilization dimensions of those modules can be increased by storing their outcomes as intermediate states in such systems. In other words the possibility to use the transformed data in later stages to reduce computation time, to enhance reusability and to ensure error recovery can be increased by a proper data management.

Modular programming is a software design approach that draws attention to uncoupling the functionalities of a script towards independent, interchangeable, reusable units [30]. On the other hand, data modularity is always anticipated for quick and flexible access to a dataset. In this paper, the intermediate states of data are being recognized as modular data which are outcomes of modular programs. The dynamic management of datasets is another ground to have intermediate states for a huge amount of data [25]. Some other common benefits of intermediate data are sustainability, scalability, quick construction and cost savings [25], [26], [27], [28]. In many circumstances of a data-intensive distributed application, image processing is an inevitable part where contemporary technologies of image analysis are pertinent for processing a large set of image data. However, special care is needed for both image data and program to process a huge amount of data with such emerging technologies. To make users' tasks more realistic in real time for Big Data analytics in image processing, intermediate modular data states can be an appreciable settlement to design and observe pipelines or workflows. In image-based tasks, the common execution operations (such as pre-processing, transformation, model fitting, and analysis) can be wrapped up as modules, and their outputs can be used for both observation and post-processing. Another important aspect of Big Data analytics is to store lots of data in a way that can be accessed in a low-cost [17], thus data management with the reusable intermediate data and program states might be a great deal to restore program state by reflecting the lower cost.

A distributed image processing system requires huge storage and data processing time for high-resolution large files. For such a system, stored intermediate states can be fruitful rather than transmitting raw files for a paralleled task. Spark itself generates some intermediate states of data in its processing engine to reduce IO cost [18]. Hence, in our modular programming paradigm, if we introduce a mechanism of storing intermediate states and manage them, we can reuse our data from various modules without computation which eventually minimizes total cost of processing. Many researchers and organizations are now pointing up on long time data processing with distributed parallel environments for low cost and convenient system to handle a large amount of data [19]. Following the trends, a machine/deep learning-based algorithm with heterogeneous data is another emerging practice to analyze a large amount of data for agricultural and other data-intensive tasks [20]. Considering the current technologies and trends, our scheme would be a contemporary package to analyze a large amount of image data.

Although a few studies [25], [26], [27], [28], [31] focused on propagated results in the intermediate states in Big Data management, to the best of our knowledge none reflected the performance implications of reusability of intermediate data and model in distributed platforms. Moreover, these studies do not show how the intermediate states across various image processing workflows can be reused. To fill the gap, in this paper we investigated whether the intermediates states generated from modular programs can speed up the overall execution time of workflows in Hadoop/Spark based distributed processing units by assuring enhanced reusability and error recovery. Here our idea was that loading intermediate states from HDFS (Hadoop Distributed File System) might take longer than generating them in memory during the execution of a workflow. In order to figure out the actual situations, we developed a web-based interactive environment where users can easily access intermediate states and associated them across various workflows, e.g., some outcomes of the image segmentation pipeline are reusable to the image clustering and registration pipelines. We found that even though it takes some time to load intermediate states from an HDFS based storage, a workflow can be executed faster in a system of handled intermediate states in persistent memory than a system of no intermediate states in persistent memory. Our finding is described in Figure 5 for various workflows. In the figure, we illustrate if we can skip some modules with the help of stored intermediate states, we can get better performance than the case of without storing intermediate states. Details of the performance and re-usability have been discussed in the section 5. The rest of the paper is organized as follows. Section 2 discusses the related work, Section 3 presents our proposed scheme of intermediate data management, Section 4 describes our experiment setup, Section 5 compares our proposed scheme with the usual technique of intermediate data handling by using various image progressing pipelines, Section 6 describes some valuable findings towards data management for modular programming, and finally, Section 7 concludes the paper.

2 Related Work

A number of studies [1], [5], [7], [15] have investigated recent techniques of filesystem to process Big Data and some studies [6], [10], [12], [13], [14] were presented with application of data or image processing in such file systems with metadata management. Similarly, various databases, algorithms, tools, technologies, and storage systems are acknowledged in a number of studies while solving the problem of big data processing for image analysis and plant-based research in distributed environments. Studies related to phenotyping, image processing and intermediate states in distributed environments are considered to compare with our work, and some of them are presented below in three subsections.

2.1 Study on file systems of Big Data processing

Blomer [1] investigates various file systems of computer systems and compared them with parallel distributed file systems. Benefits of both distributed and core file systems of computer systems are discussed. As well as, a distributed file system as a layer of the main file system has been addressed with common facilities and problems. In their work, data and metadata managements key points are also discussed for map-reduce based problems and fault tolerance concept. Luyen et al. [5] experiment on different data storage systems with a large amount of data for phenotyping investigation. Their work on a number of heterogeneous database systems and technologies such as MongoDB, RDF, SPARQL, and so on. explores possibilities of interoperability and performance. Similar to the above studies, Donvito et al. [15] discuss and compare various popular distributed storage technologies such as HDFS, Ceph, and GlusterFS. Wang et al. [7] propose a technique of metadata replication in Hadoop based parallel distributed framework for failure recovery. By emphasizing metadata replications in slave nodes for reliable data access in big data ecosystem, the proposed metadata replication technique works in three stages such as initialization, replication and failover recovery. Similarly, Li et al. [10] propose a log-based metadata replication and recovery model for metadata management in a distributed environment for high performance, balanced load, reliability, and scalability. In their work, caching and prefetching technologies are used for the metadata to enhance performance. While above studies focus on metadata management in various file systems, our study is fundamentally different with an investigation of a scheme of intermediate data management for distributed file systems considering reusability, error recovery and execution at a lower cost.

2.2 Big Data in terms of Image Processing

Minervini et al. [2] discuss different challenges in phenotyping by addressing the current limitations in image processing technologies, where collaboration is emphasized from diverse research group to overcome the challenges. Although collaboration is an important part of a scientific workflow management system (SWfMS) and we are proposing a scheme for a SWfMS, but the main focus in this study is consideration of intermediate states with micro-level modular computational-intensive programs while managing workflows in a distributed environment for increasing efficiency. Some other studies on phenotyping concentrate on the costs of processing and appropriate usage of current technologies to reduce the costs such as Minervini et al. [11] propose a context-aware based capturing technique for JPEG images that will be processed by distributed receivers in a Phenotyping environment. Singh et al. [20] provide a comprehensive study on ML tools in phenotyping area for appropriate and correct application of them in plant research. Coppens et al. [14] emphasize data management for automated phenotyping and explore the image metadata potentiality for synergism (Usage of metadata with main features). Sensor's metadata of imaging technique could be used by appropriate filtering for the synergism is also addressed in their work. Smith et al. [6] address the importance of metadata in Big Data ecosystems to migrate and share data or code from one environment to another. Data consolidation, analysis, discovery, error, and integration also have been scrutinized from metadata usage perspective in the Big Data environment. Pineda-Morales et al. [13] emphasize on both file metadata and task metadata to propose a model for filtering hot metadata. After categorizing and analyzing both hot and cold metadata, they recommend for distributing hot metadata (frequently accessed) and centralizing cold metadata in a system of data management. Although the above works show the behavior and direction of metadata, none of them analyze the processed data or intermediate data as metadata for efficient management. In our work, modular outcomes of computational intensive programs are analyzed, and a scheme is proposed to store those intermediate outcomes for efficiency in a SWfMS.

2.3 Study on Intermediate-data

Becker et al. [25] survey on current Big Data usage and emphasize on intermediate data behavior for iterative analysis at every stage of processing to minimize the overall response time with an algorithmic perspective. Besides, intermediate data can be saved in persistent storage at checkpoints was considered for error recovery. Same as Heit et al. [31] explained the nature of intermediate data while giving a statistical model for Big Data as well as intermediate data should be independent was reported in their work. Intermediate data management is considered by Tudoran et al. [26], where they pointed out storing of intermediate-data for rollback purposes from the application level. Intermediate data for checkpoints analysis are also considered by Mistrik and Bahsoon in their work [27]. Likewise, how intermediate data are dispatched in parallel distributed systems such as Spark, Hadoop is discussed by Han and Hong [28] in their book by bringing together and analyzing various areas of Big Data processing.

Although, some of the above works were discussed with the fault-tolerant concept using intermediate states, none of them discussed the possibility and necessity of storing intermediate data for Big data analytics in a distributed environment. Reviewing the above works, we realized that in a distributed environment efficient data access mechanism is a big concern and time implications for both data storing and loading are needed to be observed in a distributed environment. Focusing these, we investigate the possibility of a data management mechanism by storing intermediate states with the help of modularized programs for enhancing processing time, reusability and error recovery.

3 Proposed Method

Modular programming for computation-intensive tasks is already a proven technology for reusability of code [21], [22], [23], [24], but for large raw data transaction and loading from distributed systems, we cannot fully use the proficiency of modularity. On the other hand, processed data require less memory, transaction and loading time. Having both these knowledge, we store processed intermediate data with other program data and want to explore three research questions in a SWfMS, such as

- **RQ1:** Does the system support data reusability that ultimately reduces time and efforts?
- **RQ2:** Does it have any error recovery mechanism that could construct workflows quicker if errors occur?
- **RQ3:** and How the system is supporting fast processing for frequently used workflows?

Answers to these three research questions have been presented in the section 5. To use both of these intermediate data and modular programs, and explore the usability of modular outcomes in our full system a modularized data access mechanism is introduced. We have an interactive web interface, which is used for accessing datasets of various modules input and output. A dedicated distributed data server where images and intermediate data are stored was used in our system for parallel processing. From the web interface, image processing workflows can be built, and modules of the workflows can be submitted to a distributed processing master node as jobs, by this way distributed image data are processed with specified modules from the workflows.

Two major types of metadata of image processing environment are file-structure-metadata and result-metadata [13]. In our system, we managed both kinds of metadata using APIs that retrieve and store information on a dedicated Hadoop server. This dedicated server is designed to facilitate various parallel processing from different modules of pipelines or workflows. When a SHIPPIs (Spark High-throughput Image Processing Pipeline) module needed to be executed for a task, available data or intermediate data states are suggested from a web interface with the help of the metadata and HDFS APIs (e.g., Figure 1). SHIPPI is our designed library that provides cloud-based APIs for the modular functionality of image processing [30]. Details of the SHIPPI has been described in the experimental setup section, i.e., Section 4. Our system at first tries to find suitable datasets for a modular task with task information and information from the metadata server, suggested datasets are presented on a web interface to be selected for a particular task from our distributed storage system, i.e., HDFS.

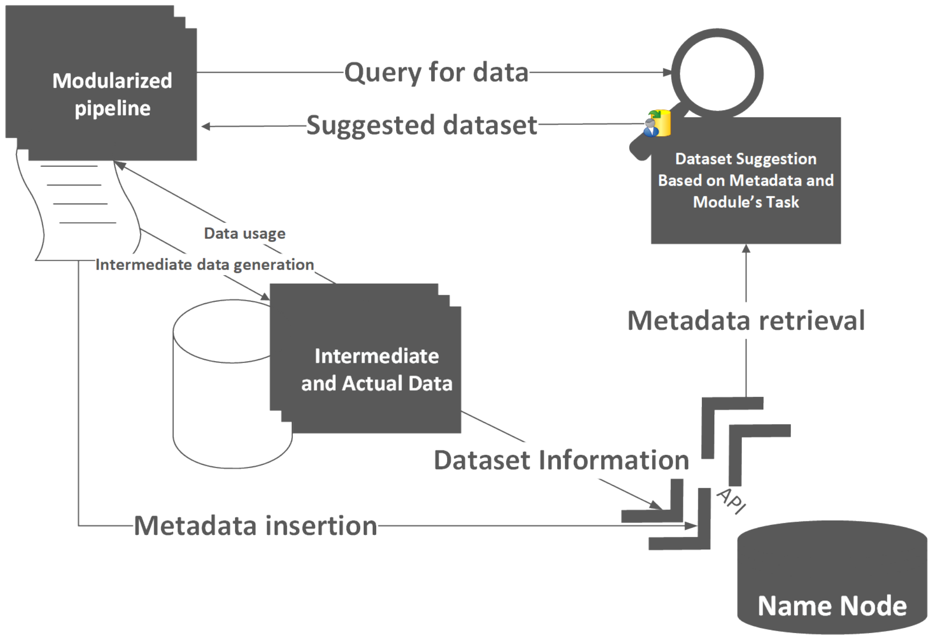


Fig. 1. Proposed data management scheme.

Suggested datasets can be an intermediate or raw data, but the intermediate states will always have a higher priority to be selected if available. Another important task of our system is to keep track of metadata and intermediate data. After every execution, there are a few outputs from a single module such as analysis results, intermediate data, and so on. Intermediate data are stored in a distributed file system with actual data, and analysis results are stored in a server as metadata for future use.

How the proposed scheme of data management by modular programs and intermediate data can introduce reusability, error recovery and faster processing in a Scientific Workflow Management System (SWfMS) is described with simple workflows (Figure 2). In the figure, four workflows are illustrated with modules and intermediate data. All black and grey marked processes (Ps) and intermediated data (IDs) represent more than the one-time existence of them in the four workflows and possibility to be reused of them. This reusability of them means they are likely to be reused later in either for the same workflow with different tools or the other workflows in a SWfMS. Different intensities of black and grey colours represent the same modules or intermediate states from different workflows in different groups. A particular group is formed for the same operations on the same dataset or for the intermediate states that are being generated for the same operation of module sequences on them with same input dataset. In workflow 1, we have four modules (P1, P3, P5, and P7). These modules perform operations sequentially on dataset D1 and produce three intermediate

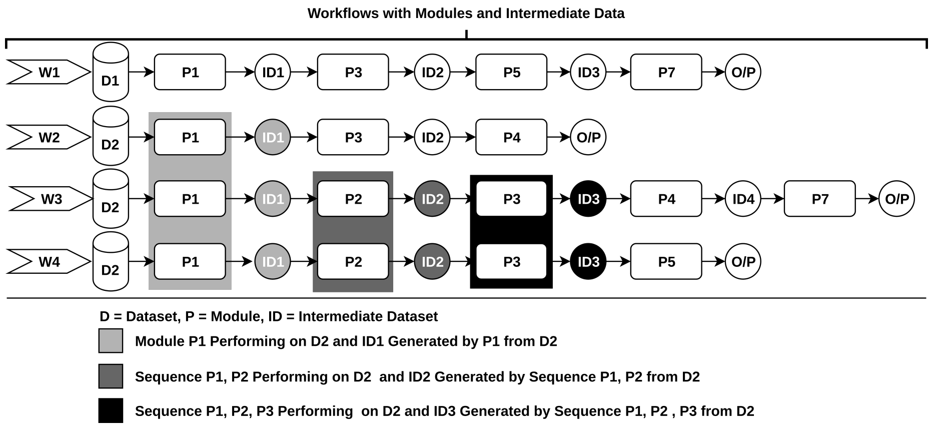

Workflows with Modules and Intermediate Data
W1
D1
P1
ID1
P3
ID2
P5
ID3
P7
O/P
W2
D2
P1
ID1
P3
ID2
P4
O/P
W3
D2
P1
ID1
P2
ID2
P3
ID3
P4
ID4
P7
O/P
W4
D2
P1
ID1
P2
ID2
P3
ID3
P5
O/P
D = Dataset, P = Module, ID = Intermediate Dataset
Module P1 Performing on D2 and ID1 Generated by P1 from D2
Sequence P1, P2 Performing on D2 and ID2 Generated by Sequence P1, P2 from D2
Sequence P1, P2, P3 Performing on D2 and ID3 Generated by Sequence P1, P2 , P3 from D2

Fig. 2. Reusable intermediate data in workflows.

datasets (ID1, ID2, and ID3), where the last outcome is desired output (O/P). All workflows have some processing modules to perform operations on specific datasets or intermediate datasets for generating desired intermediate outcomes or final results. The intermediate states are usually used by intermediate modules sequentially (N. B. For simplicity we are considering only sequential module processing in workflows) in workflows. Workflow 2 has three modules (P1, P3, and P4) that generate two intermediate datasets (ID1 and ID2). Workflow 3 and 4 have five and four processing modules with four and three intermediate stages respectively. Last three workflows in the figure are working on the same dataset D2, and a few modules such as P1, P2, and P3 are frequently used for building those workflows. This scenario opens the possibility of reusability of intermediate states, as it is common in a SWfMS that same modules can be used in different workflows. As well as, the same sequence of operations on the same or different dataset can be occurred in various workflows by supporting the possibility of program reusability. In the workflow 2, an operation by module P1 is performed on dataset D2 and generates intermediate dataset ID1. This operation and the outcome (ID1) are the same in the other three workflows for the same module sequence and same dataset. Same sequence $P1 \rightarrow P2 \rightarrow P3$ in workflow 3 also occurs in workflow 4 with the same input data set D2. Thus, ID3 (intermediate dataset) from workflow 3 can be directly used in workflow 4 for skipping the first three module operation to analyze with only last module or increase the performance. To introduce this type of efficiency and performance enhancement in a system of workflow management, modules outcome is needed to be stored with appropriate access mechanism. Furthermore, if failures occur in any workflow, stored intermediate states (IDs) can be used to recover the workflow by addressing the issues only in faulty modules. In a distributed environment, data are partitioned and replicated on different nodes, and the transition time of data is not as simple as it is a single file system. Data transfer time from slave nodes to master nodes is needed to be considered for both storing and retrieving. So, for storing and retrieving intermediate data from such distributed systems, experiments are necessary regarding transition and computation time. Our experimental section (Section 5) is designed in that way to investigate the actual scenario of I/O operation and computation time for the micro level modular computation-intensive programs in a distributed environment and explore the possibility of reusability.

4 Experimental Setup

In our experiments, for evaluating a full system performance with the proposed scheme, a web platform is used, a web interface is designed with current web technologies and Flask (A Python micro-framework) web framework. From the web interface, SHIPPIs APIs are called to execute a job in an OpenStack based Spark-Hadoop cluster (e.g., five-node, 40 cores, and total 210 GB RAM). It provides cloud-based APIs for the modular functionality of image-processing pipeline and automatically parallelizes different modules of a program. Each API call is modularized into four sections such as conversion, estimation, model-fitting, analysis-transform. Figure 4, shows the block diagram of SHIPPI, which represents the relation among different libraries and modules of the framework. In our testing environment, HDFS is used to store our image data for parallel processing. To store intermediate states, we used the Pickle library of Python, and we designed three image processing pipelines - Segmentation, Clustering and Leaves Recognition following our proposed model discussed in Section 3. Every pipeline was tested at least five times and an average of their execution time was recorded. For testing purposes, we mainly used three datasets - *Flavia* [29], *2KCanola*, *4KCanola*. Each of those datasets has more than 2000 images, where *Flavia* is a known dataset used for leaves recognition system, and *2KCanola* and *4KCanola* datasets are from the P2IRC (Plant Phenotyping and Imaging Research Centre) project of University of Saskatchewan used for various purposes of image analysis. The three pipelines are illustrated in Figure 3, where a single

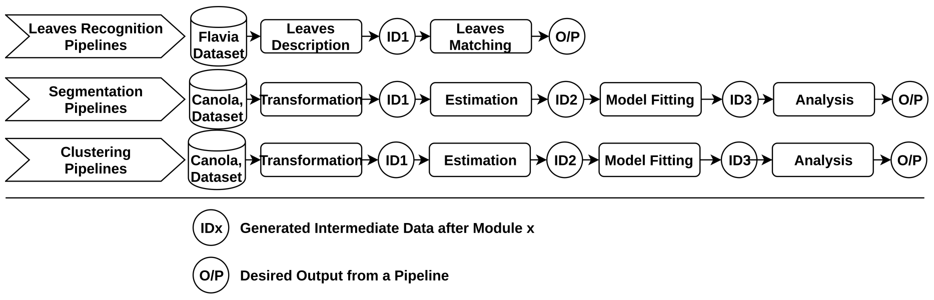


Fig. 3. Pipeline building with image processing tools.

job such as leaves recognition, segmentation or clustering is built by their respective modularized parts. The first module (Leaves Description) of the leaves recognition pipelines extracts features and prepares data for the next module leaves matching. The next module (Leaves Matching), which is the final module in the leaves recognition pipelines does the job of comparing those features and reports a classification result of leaves. Similarly, image segmentation and clustering pipelines are modularized by four common modules such as *transformation, estimation, model fitting, and analysis.* Transformation is used for mainly colour conversion, estimation is used for feature extraction, model fitting is used for training an algorithm and setting parameters, and the final module analysis is used for testing and result preparation. Pipelines can be built by selecting modules and parameters for submitting a desired task in the distributed environment on the web interface, where each module works as a job in the distributed environment and their outcomes are gathered from a master node and distributed file system for further analysis.

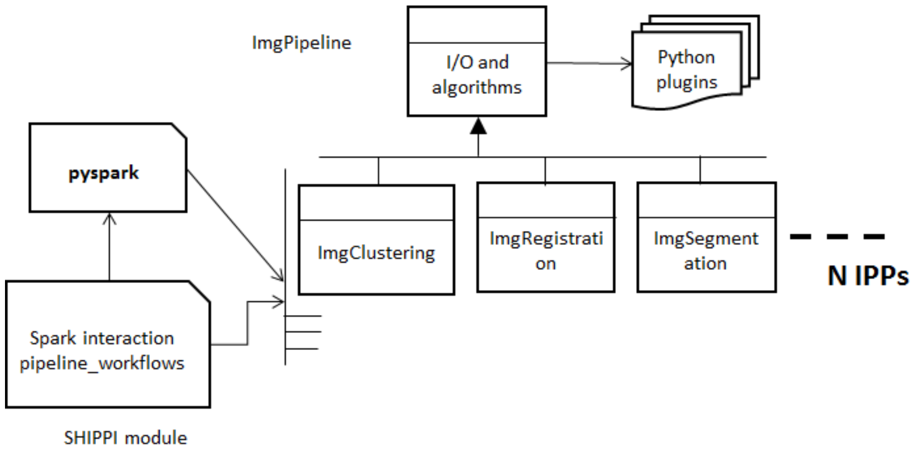


Fig. 4. Block diagram of components of SHIPPI.

5 Results and Evaluation

We developed a Spark High-throughput Image Processing Pipeline framework known as SHIPPI, which is an image processing pipeline library to abstract map reduce, data loading, and low-level complexity of image operations. SHIPPI was written in Python 2.7 and Spark 2.0.1, four modules of our modular program paradigm of a pipeline using SHIPPI can be modified based on image processing operations concerning the desired purpose. Furthermore, these modules are parallel with the help of PySpark library, and each output of a module is used in the next module. In a spark-based solution all data are in main memory, so to reuse those processed data, we store them as intermediate states in Hadoop distributed file system from those modules that give reusable data. This system not only designed for reusability but also for error recovery (for the higher failure rate of a long-running program or pipeline with a huge amount of data). Thus, if we store intermediate data, there is a chance for us to recover a system at low cost.

Table 1. Time gain for the reusable intermediate data

Pipeline Tool	**Step 1**	**Step 2**	**Gain**
Leaves Description	1163.7 sec + 35.7 sec	-	-
Leaves Matching	Can be skipped	175.9 sec	1199.5 sec

Total eight distinct modules of the three image processing pipelines have been implemented with current technologies, and all of them are designed for both without intermediate and with intermediate data concept. Both transformation and estimation modules in the pipelines use the same technology, and they are similar in the last two pipelines. Figure 5 is the execution-time comparison graph of both of the concepts for the three pipelines in distributed and ubuntu file systems. Although saving intermediate states is a memory overhead task for all of the pipelines in the figure, but there are possibilities to use stored data and skip some processing steps for executing a pipeline at low cost. Skipping procedure eventually increases the flexibility and reusability to analyze fractions of pipelines in low cost. Pipelines with skipped modules by using intermediate data are presented with the improved performance in the figure. Another major concern of our investigation was to explore the trade-off scenario between computation time and data loading/storing time of modules in distributed systems. For investigating this, previously we assumed that a data and metadata transition time among slave and master nodes would be more than a module computation time in a distributed environment. But the experimental results for the pipelines of skipped modules show improved performance and contradict with our assumption regarding time implication of data transfer. Hence, a system of workflow management with distributed processing unit is capable of handling intermediate states, and a scheme is possible to introduce in such systems for reusability, error recovery and performance enhancement. We organize our experiment into three sections to answer the research questions of our system's usability. All of the three pipelines of image processing are considered in each section for their respective illustrations. Below are the three main subsections of our experiments.

5.1 Experiment for reusability

A collaborative SWfMS is used for design and job submission of the image processing pipelines to a distributed parallel processing system. Figure 6 and 7 show

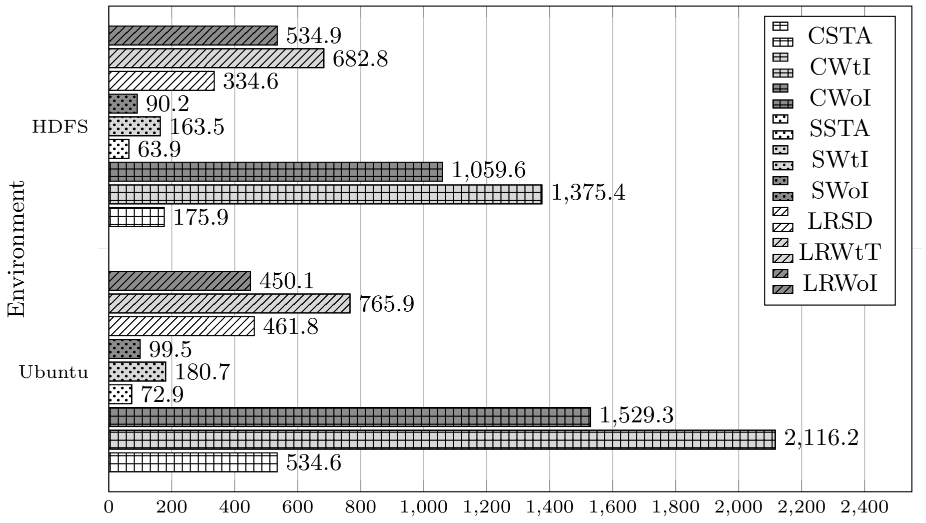
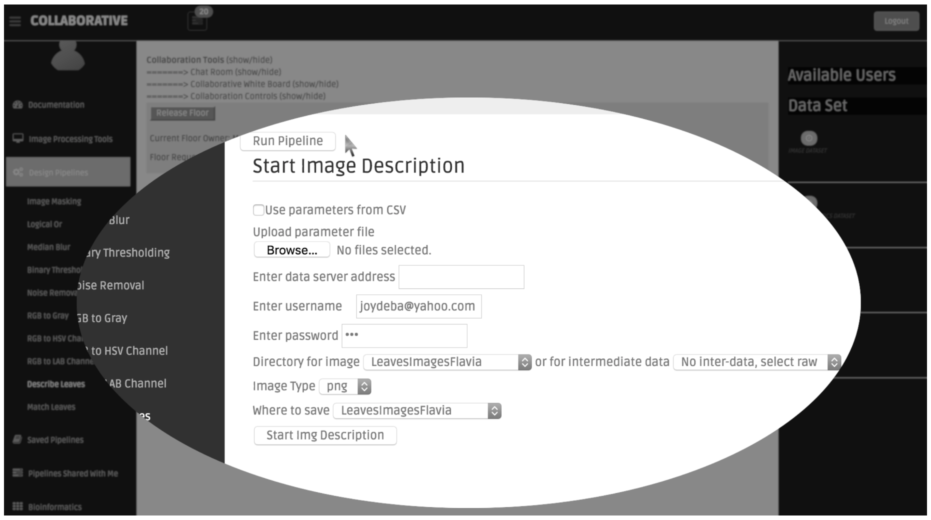


Fig. 5. Performance comparison of workflows with-intermediate and without-intermediate data

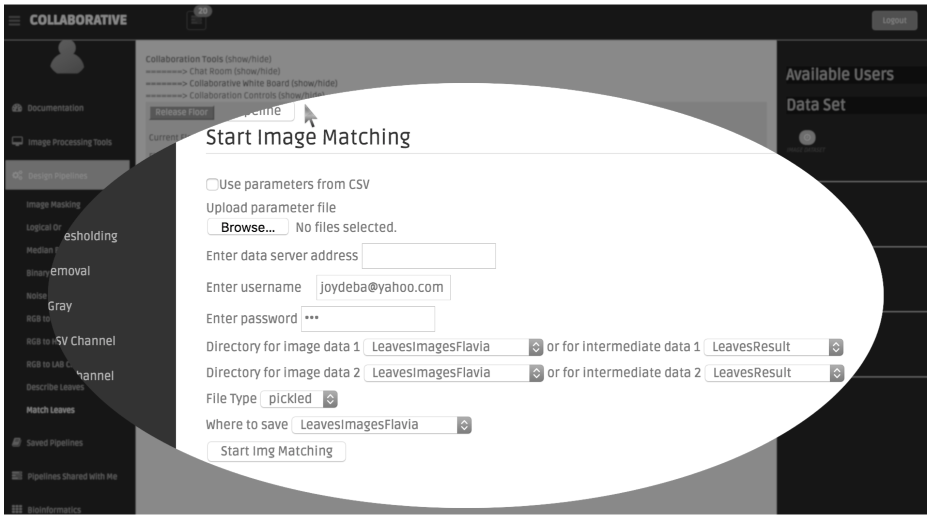


Collaboration Tools (show/hide)
=======> Chat Room (show/hide)
=======> Collaborative White Board (show/hide)
=======> Collaboration Controls (show/hide)

Start Image Description

☐ Use parameters from CSV
Upload parameter file
Browse... No files selected.
Enter data server address
Enter username joydeba@yahoo.com
Enter password •••
Directory for image LeavesImagesFlavia or for intermediate data No inter-data, select raw
Image Type png
Where to save LeavesImagesFlavia
Start Img Description

Fig. 6. Image description module in our interactive workflow composition environment.



Start Image Matching

☐ Use parameters from CSV

Upload parameter file

Browse... No files selected.

Enter data server address

Enter username joydeba@yahoo.com

Enter password •••

Directory for image data 1 LeavesImagesFlavia or for intermediate data 1 LeavesResult

Directory for image data 2 LeavesImagesFlavia or for intermediate data 2 LeavesResult

File Type pickled

Where to save LeavesImagesFlavia

Start Img Matching

Fig. 7. Image matching module in our interactive workflow composition environment.

the configuration interface for the image recognition pipelines, which is implemented with SHIPPIs four-phase modular architecture where only two modules are considered for simplicity. Intermediate state from the first module of this pipeline can be reusable to image matching modules. So, an outcome of the image description stage is important for an image matching technique and needed to be stored for reuse. In Figure 7 of the image matching technique, there is an option to use intermediate states for the reuse. Possibility to use intermediate states is the reusability model we are using in our system, an outcome of a module can be used in other modules if available and appropriate for those modules. Consider another situation, when image files are huge in size for high resolution; then pipelines take a large portion of total execution time for only data transferring and loading in a distributed parallel processing system. Thus, stored features or descriptors as intermediate states can help to reduce the transfer time up to a certain level. From table 1, we can see that module at step 1 (Image Descriptor) required more than 1199.4 seconds to accomplish the feature extraction part. In a pipeline design, we can skip this step if we can use intermediate states and can gain up to 1199.4 seconds.

5.2 Experiment for error recovery

While a pipeline is running in a cluster, it is common that all the steps/modules may not be succeeded and the pipeline may fail to produce the desired output. But in a system of workflow management stored intermediate states can be helpful to recover a workflow execution by applying appropriate module configuration. If we store intermediate data and necessary parameters for modules and if there is an error in a module operation; still, data and configuration up to the previous module can be used for stored intermediate data and parameters. So, configuring the failed module with the stored parameters and intermediate data, next run of the pipeline will only take time for the failed modules. For example, in our case (Figure 8), image recognition pipeline is divided into two modules using SHIPPI library, the second stage sometimes fails for too many comparisons and I/O operations, but using intermediate states we can quickly re-run the second stage by only reconsidering the error issues in that module. Similarly, both segmentation and clustering pipelines are error-prone at the model fitting stage for lots of computation, in that case, up to two stages of computation can be saved if we can reuse stored data with appropriate parameters, and this is possible from our web interface. Besides, in a distributed environment when computation and data are enormous, the failure rate is high for various tasks of a job due to memory overhead and access limitations of different libraries. So, our proposed approach can be a feasible solution in such type of computation-intensive tasks to avoid errors or failures.

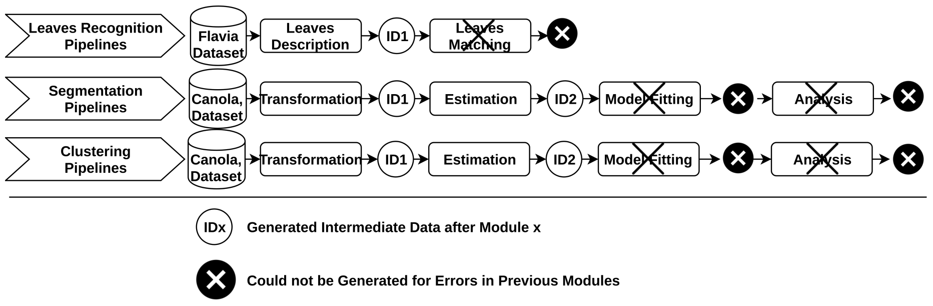


Fig. 8. Common error occurrences in workflow execution.

5.3 Experiment for fast processing

Both distributed and ubuntu file systems are considered for the execution time experiments of our scheme, where pipeline design is considered with both with-intermediate and without-intermediate data. Table 1, Figure 5 and Figure 9 are presented with those execution time experiments. Examining the table, we can say that for the intermediate states, possibilities of skipping execution steps can make pipeline execution time shorter than the normal execution. The Figure 5 illustrates the execution time comparisons and performances for the three image processing pipelines in our SWfMS. These image processing pipelines are categorized by three techniques of workflow building for two file systems. The techniques are - execution of workflows by storing intermediate data, by not storing intermediate data and by skipping some module operations. So, there are nine experiments in each file system that makes total eighteen execution scenarios in our system for both file systems. Short forms of the above techniques are given below.

– Leaves Recognition workflow without storing intermediate data (LRWoI)
– Leaves Recognition workflow with storing intermediate data (LRWtI)
– Leaves Recognition workflow by skipping descriptor operation (LRSD)
– Segmentation workflow without storing intermediate data (SWoI)
– Segmentation workflow with storing intermediate data (SWtI)
– Segmentation workflow by skipping transformation and analysis (SSTA)
– Clustering workflow without storing intermediate data (CWoI)
– Clustering workflow with storing intermediate data (CWtI)
– Clustering workflow by skipping transformation and analysis (CSTA)

The grided dark-grey, grey, and white bars in the Figure 5 representing leaves recognition system performance in two different file systems. If we skip the descriptor calculation part using intermediate state, it performs better, the grided white bars in the figure presents skip of the descriptor part and performances. Same goes for the clustering and segmentation pipelines, dotted and lined bars in the figure represent segmentation and clustering algorithms performances respectively, and white bars of these patterns represent the skipping of modules and improved performance. Furthermore, from the Figure 5, we can have an idea of quantitative values of performance for three different pipelines where grey colours represent all modules computation time of pipelines with or without intermediate states, and white colours represent the computation time of the pipelines that can skip some modules operation for the availability of intermediate states. So, it can be interpreted from the chart that, if we can skip some module operations for the intermediate states, system or pipeline can be executed at a low cost in both of the file systems. However, there are some delays to store data in a storage system, but for the reusability, a system can really work fast for already processed data. In a SWfMS, amount of data is increased frequently for processing of various workflows. So, without proper data management, volume and velocity of big data cannot be addressed efficiently. As well as, different types of data exist in a SWfMS without proper annotation and association variety of big data processing cannot be granted. All of the three Vs of big data can be appropriately handled in a system of scientific workflow management by using a data-centric component management. Data-centric component development for large-scale data analysis is a current trend to reduce error rates and increase usability, and our system can provide such kind of facilities using stored intermediate data.

6 Discussion and Lesson Learned

From our experimental studies, we figured out that although loading intermediate states from HDFS takes some extra time, the solution and possibility of using intermediate states shows better performance than loading raw large image files for the execution of image processing workflows. In Figure 9, normal execution to modules skipping performances have been illustrated for all of the three image processing pipelines. For each pipeline, gain by skipping modules is directly proportional to their normal execution time, as much as the computation time increases, the gain is increased for the time too. This increment directly portraits that the proposed scheme is more effective for computationally intensive tasks than the average workload of pipelines.

We also experienced that discovering data reusability across various image processing tasks are challenging, and significant analysis is required in this regard. In this situation, breaking up an image processing task into micro-level tasks and offering APIs for each task with the micro-level modular programming concept is a viable solution. We learned in a system of modular programming a user has more control over a task configuration such as if a user wishes, certain steps can be used, or some can be skipped with respect to the required functionalities and data availability. Some other common benefits of modular program design that we experienced are - easy bug detection, less computation error, easy program handling, and modular testing. Although we noticed this kind of modularity hampers performance, with the appropriate reusability of intermediates states, it is possible to recover the loss. In order to figure out matching intermediate states across various image processing tasks, applying machine learning algorithms might be another solution. In this case, we need to tag intermediate states with metadata to associate them with the tasks they might be used. A system should automatically suggest users for using intermediate states during compositions of workflows.

While we are working in our cluster, we gathered some valuable knowledge of a distributed processing framework such as for parallel processing the cores of different nodes are busy with different tasks of a specific job. So, if anyone wants to store any processed data, it needs to happen on a master node. To store any states, some data transition time from different nodes will be added to the storing procedure, which eventually increases a program's execution time. Another common problem of a distributed system is memory overhead exception. If a master node does not have enough main memory, programs cannot execute operations on a large dataset. For example, in our case, 30GB RAM of the master node could not process 4K images, later which was increased to 90GB and then 4K images were processed with a clustering algorithm. Another problem of a Spark based distributed system is, for rapid file system access program it throws IO exceptions or SSH library file loading exceptions. Although there are various challenges to process a large dataset in a distributed environment but to facilitate the real-time experience for Big Data, distributed processing environments are inevitable.

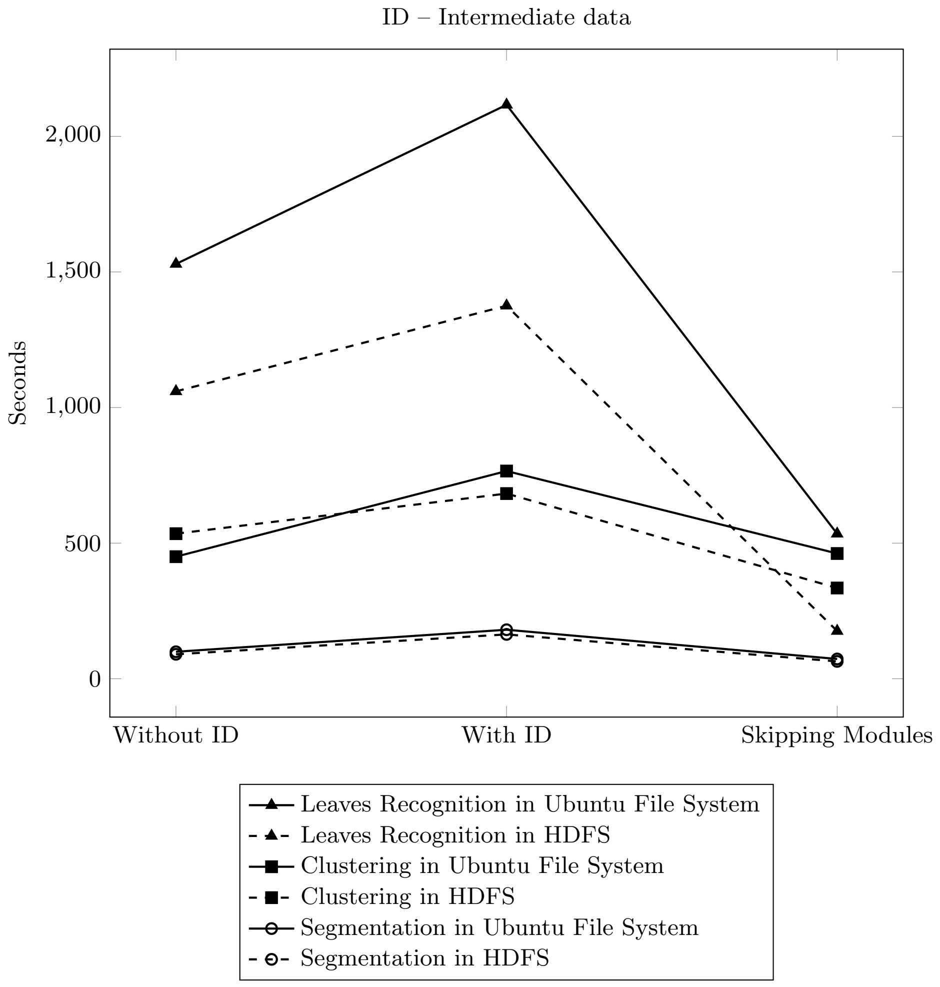


Fig. 9. Performance comparison in both environments

7 Conclusion

We devised a data management scheme for our image processing pipeline framework, where we considered to store intermediate data and program state to reuse processed data and recover a pipeline failure with the data. When a dataset has a large number of images and associate programs need long computational time, there is a chance of high error rate. In spite of that, existing models have not concentrated on an organization of intermediate states except raw data and metadata. Considering the above statements, we proposed a model of intermediate state and showed that using intermediate state, it is easy to recover failures quicker and increase reusability in a SWfMS. In addition, intermediate states can be used in other pipelines to reduce the time cost, which eventually increases data reusability. To provide end users an interactive environment for processing Big Data in a parallel distributed framework and at the same time give them reusability with more control is a challenging job. Only a modularization of a program or task might not be a feasible solution in many cases. So, our model with data modularization or intermediate data state at every stage of a modular program will give a user more control of usability. Above all, our study contributes to Volume, Velocity and Variety creation in Big image processing. In the current practice of workflow or pipeline design, high computation capability resources may not facilitate users if users do not have more control over data usability. Hence, using the proposed model of intermediate states of data, resource utilization can be increased up to a certain level. Our experimental results were presented with such utilization, which makes possible the practical use of our system.

References

1. J. Blomer, *Experiences on File Systems: Which is the best file system for you?*, Journal of Physics: Conference Series. 664(4), p.042004, 2015.
2. M. Minervini, H. Scharr, and S. Tsaftaris, *Image Analysis: The New Bottleneck in Plant Phenotyping [Applications Corner]*, IEEE Signal Processing Magazine. 32(4), pp.126-131, 2015.
3. A. Walter, F. Liebisch and A. Hund, *Plant phenotyping: from bean weighing to image analysis*, Plant Methods. 11(1), p.14, 2015.
4. B. Depardon, G. L. Mahec, C. Seguin, *Analysis of Six Distributed File Systems*, [Research Report]. pp.44, hal-00789086, 2013.
5. L. N. Luyen, A. Tireau, A. Venkatesan, P. Neveu, and P. Larmande, *Development of a knowledge system for Big Data: Case study to plant phenotyping data*, In Proceedings of the 6th International Conference on Web Intelligence. ACM, New York, NY, USA, Article 27, 9 pages, 2016.
6. K. Smith, L. Seligman, A. Rosenthal, C. Kurcz, M. Greer, C. Macheret, M. Sexton, and A. Eckstein, *"Big Metadata": The Need for Principled Metadata Management in Big Data Ecosystems*, In Proceedings of Workshop on Data analytics in the Cloud. ACM, New York, NY, USA, Article 13, 4 pages, 2014.
7. F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, *Hadoop high availability through metadata replication*, In Proceedings of the first international workshop on Cloud data management. ACM, New York, 37-44, 2009.

8. B. Roy, A. K. Mondal, C. K. Roy, K. A. Schneider and K. Wazed, *Towards a Reference Architecture for Cloud-Based Plant Genotyping and Phenotyping Analysis Frameworks*, IEEE International Conference on Software Architecture (ICSA), Gothenburg. pp. 41-50. 2017.
9. E. Skidmore, S. Kim, S. Kuchimanchi, S. Singaram, N. Merchant, and D. Stanzione, *iPlant atmosphere: a gateway to cloud infrastructure for the plant sciences*, In Proceedings of the 2011 ACM workshop on Gateway computing environments. ACM, New York, NY, USA, 59-64, 2011.
10. B. Li, Y. He and K. Xu, *Distributed metadata management scheme in cloud computing*, 6th International Conference on Pervasive Computing and Applications. Port Elizabeth, pp. 32-38, 2011.
11. M. Minervini and S. A. Tsaftaris, *Application-aware image compression for low cost and distributed plant phenotyping*, 18th International Conference on Digital Signal Processing (DSP). Fira, pp. 1-6, 2013.
12. L. Pineda-Morales, A. Costan and G. Antoniu, *Towards Multi-site Metadata Management for Geographically Distributed Cloud Workflows*, IEEE International Conference on Cluster Computing. Chicago, IL, pp. 294-303, 2015.
13. L. Pineda-Morales, J. Liu, A. Costan, E. Pacitti, G. Antoniu, P. Valduriez, and M. Mattoso, *Managing hot metadata for scientific workflows on multisite clouds*, IEEE International Conference on Big Data (Big Data). Washington, DC, 2016, pp. 390-397, 2016.
14. F. Coppens, N. Wuyts, D. Inz, and S. Dhondt, *Unlocking the potential of plant phenotyping data through integration and data-driven approaches*, Current Opinion in Systems Biology. 4, pp.58-63, 2017.
15. G. Donvito, G. Marzulli and D. Diacono, *Testing of several distributed file-systems (HDFS, Ceph and GlusterFS) for supporting the HEP experiments analysis*, Journal of Physics: Conference Series. 513(4), p.04, 2014.
16. M. Kim, J. Choi and J. Yoon, *Development of the Big Data Management System on National Virtual Power Plant*, 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC). Krakow, pp. 100-107, 2015.
17. A. S. Kaseb, A. Mohan and Y. H. Lu, *Cloud Resource Management for Image and Video Analysis of Big Data from Network Cameras*, JInternational Conference on Cloud Computing and Big Data (CCBD). Shanghai, pp. 287-294, 2015.
18. X. Yang, S. Liu, K. Feng, S. Zhou and X. H. Sun, *Visualization and Adaptive Subsetting of Earth Science Data in HDFS: A Novel Data Analysis Strategy with Hadoop and Spark*, IEEE International Conferences on Big Data and Cloud Computing (BDCloud). Atlanta, GA, pp. 89-96, 2016.
19. S. K. Prasad et al., *Parallel Processing over Spatial-Temporal Datasets from Geo, Bio, Climate and Social Science Communities: A Research Roadmap*, IEEE International Congress on Big Data (BigData Congress). Honolulu, HI, pp. 232-250, 2017.
20. A. Singh, B. Ganapathysubramanian, A. K. Singh, S. Sarkar, *Machine Learning for High-Throughput Stress Phenotyping in Plants*, Trends in Plant Science. Volume 21, Issue 2, Pages 110-124, ISSN 1360-1385, 2016.
21. A. M. uti, M. V. D. Brand, T. Verhoeff, *Exploration of modularity and reusability of domain-specific languages: an expression DSL in MetaMod*, Computer Languages, Systems and Structures. Volume 51, Pages 48-70, ISSN 1477-8424.
22. N. Bezzo, J. Park, A. King, P. Gebhard, R. Ivanov and I. Lee, *Demo abstract: ROSLab A modular programming environment for robotic applications*, ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS). Berlin, pp. 214-214, 2014.

23. W. Sun, X. Wang, X. Sun, *Ac 2012-3155: using modular programming strategy to prac- tice computer programming: a case study*, American Society for Engineering Education. 2012.
24. F. Desprez and P. F. Dutot, *Euro-Par 2016: Parallel Processing Workshops*, Euro-Par 2016 International Workshops. Grenoble, France, August 24-26, Lecture Notes in Computer Science, 2016.
25. T. Becker, J. Cavanillas, E. Curry, W. Wahlster, *Big Data Usage, New Horizons for a Data-Driven Economy*, Springer. Cham, 2016.
26. R. Tudoran, B. Nicolae, G. Brasche, *Data Multiverse: The Uncertainty Challenge of Future Big Data Analytics*, Semantic Keyword-Based Search on Structured Data Sources. Lecture Notes in Computer Science, vol 10151, Springer, Cham, 2017.
27. I. Mistrik and R. Bahsoon, *Software Architecture for Big Data and the Cloud*. ISBN 9780128054673, Jun 12, 2017.
28. Z. Han and M. Hong, *Signal Processing and Networking for Big Data Applications*. Cambridge: Cambridge University Press, Apr 27, 2017.
29. S. G. Wu, F. S. Bao, E. Y. Xu, Y. X. Wang, Y. F. Chang and C. L. Shiang, *A Leaf Recognition Algorithm for Plant classification Using Probabilistic Neural Network*, IEEE 7th International Symposium on Signal Processing and Information Technology. Dec., Cario, Egypt, 2007.
30. A. K. Mondal, B. Roy C. K. Roy, K. A. Schneider, *Micro-level Modularity of Computaion-intensive Programs in Big Data Platforms: A Case Study with Image Data*, Technical Report, University of Saskatchewan. Canada.
31. J. Heit, J. Liu and M. Shah, *An architecture for the deployment of statistical models for the big data era*, IEEE International Conference on Big Data. pp. 1377-1384. Washington, DC, 2016.