

A Practical Roadmap for Provenance Capture and Data Analysis in Spark-based Scientific Workflows

Thaylon Guedes¹, Vítor Silva², Marta Mattoso², Marcos V. N. Bedo³, Daniel de Oliveira¹

¹ Institute of Computing, Fluminense Federal University, Niterói, RJ, Brazil

² Computer Science Department, Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil

³ Fluminense Northwest Institute, Fluminense Federal University, St. A. Pádua, RJ, Brazil

Abstract—Whenever high-performance computing applications meet data-intensive scalable systems, an attractive approach is the use of Apache Spark for the management of scientific workflows. Spark provides several advantages such as being widely supported and granting efficient in-memory data management for large-scale applications. However, Spark still lacks support for data tracking and workflow provenance. Additionally, Spark’s memory management requires accessing all data movements between the workflow activities. Therefore, the running of legacy programs on Spark is interpreted as a “black-box” activity, which prevents the capture and analysis of implicit data movements. Here, we present **SAMBa**, an Apache Spark extension for the gathering of prospective and retrospective provenance and domain data within distributed scientific workflows. Our approach relies on enveloping both RDD structure and data contents at runtime so that (i) RDD-enclosure consumed and produced data are captured and registered by **SAMBa** in a structured way, and (ii) provenance data can be queried during and after the execution of scientific workflows. By following the W3C PROV representation, we model the roles of RDD regarding prospective and retrospective provenance data. Our solution provides mechanisms for the capture and storage of provenance data without jeopardizing Spark’s performance. The provenance retrieval capabilities of our proposal are evaluated in a practical case study, in which data analytics are provided by several **SAMBa** parameterizations.

Index Terms—Provenance, Dataflow, Spark, DISC systems.

I. INTRODUCTION

Scientific workflows rely on large-scale processing to corroborate scientific hypotheses, which requires the execution of data-intensive operations such as data loading, transformations, and aggregations [1]. Although both shell and Python scripts are employed for the implementation of scientific workflows [2], they may be not suitable for executing tasks that require the controlling of parallel processing on high-performance computing environments. Such tasks require native distributed computing protocols, *e.g.*, MPI, as well as efficient serialization protocols for the distribution of objects among computational nodes [3].

Therefore, script-based workflows become tricky to design as it is expected the programmers to implement the management of parallel processing [4], which may limit code and environment and blur the workflow experimental reproducibility [5], [6]. Accordingly, several scientists employ Scientific Workflow Management Systems (SWfMS), such as

Pegasus [7] and Swift/T [8] for the modeling, enacting and monitoring of the execution of large-scale parallel workflows. SWfMSs provide a series of advantages for scientists, including a variety of scheduling strategies, fault-tolerance techniques, and provenance management mechanisms. However, some scientists are migrating their IO-intensive workflows to execute in Data-Intensive Scalable Computing (DISC) systems, such as Apache Spark¹ due to its popularity, high scalability and very large support community of users. Examples are Kira in astronomy [9], Spark-GA [10], GOAT [11], and ADAM² in the bioinformatics domain.

The main advantage of DISC systems is they enhance the workflow execution by exploring in-memory data movement and processing. *Frameworks* as Apache Spark control the execution of each activity regarding distinct data partitions to provide automatic dataflow parallelism between activities. Key Apache Spark data abstractions are the *Resilient Distributed Datasets* (RDDs), which are essentially in-memory collections of immutable and partitioned data instances to be processed in parallel [12], [13]. However, a critical issue that has yet to be addressed on the usage of Spark for the running of scientific workflows is its lack of support for provenance data [14].

Provenance data can be enriched with domain data, *i.e.*, the content of produced data files, to provide (i) partial or full reproducibility of long-lasting workflow executions, and (ii) analytics that describes both the experiment history (searches upon consolidated databases) and the current status (runtime queries) of the scientific workflow execution [15]. Spark provides a *log* for the record of parallel processing activities, but its registers are incipient for reproducibility and require user-provided parsing for the obtaining of execution statistics [16]. On the other hand, the gathering of domain data requires accessing raw data files, which somewhat resembles the problem of scripts’ design [17].

The design of a solution for capturing and storing domain and provenance data to Spark has several challenges [18]. For instance, provenance data from multiple and distributed RDDs must be associated with the in-memory content for the construction of derivation paths, *i.e.*, domain data, but the gathering of provenance data cannot jeopardize the workflow performance. Another limitation when using Spark with

¹<https://spark.apache.org/>

²<https://adam.readthedocs.io/en/latest/>

scientific workflows is the case of black-box programs that communicate with each other through *data files*. In this scenario, Spark is unable to use RDDs for the management of swapped data without flushing them to disk. Therefore, provenance management also demands the capture of data from those produced data files without derailing the entire scientific workflow execution. Although black-box operations might seem not very efficient in Spark, these codes are often highly optimized. Moreover, many workflows are composed by black-box operations mixed with source-code, *i.e.*, white-box, which turns Spark into an attractive alternative.

In this paper, we take advantage of Apache Spark RDD abstraction to design a roadmap and implement a solution we call SAMbA (Spark provenAnce MANagement on RDDs) that defines how to collect prospective and retrospective provenance alongside with domain data from scientific workflows running on DISC systems. Our rationale is based upon the premise that RDDs are a broader concept that can be used in the near future by existing SWfMSs to support in-memory processing and data movement in scientific workflows.

Accordingly, our roadmap consists in the enveloping of both RDD structure and content at runtime so that (i) RDD-enclosure consumed and produced scientific data are stored by SAMbA in a structured way, (ii) swapped domain data amid execution of black-box programs are also stored by SAMbA, and (iii) provenance data can be queried at runtime and after the execution of scientific experiments. All SAMbA data gathering routines are W3C PROV compliant by representing RDDs' content as PROV entities related to data transformation PROV activities. SAMbA generates a portable and easy-to-transform provenance database, which is also a step towards the reproducibility of Spark-based scientific workflows.

SAMBa's implementation includes an interface class for users to define domain data to be extracted and stored. Additionally, SAMbA tackles the optimization of provenance gathering on black-box applications by relying on its own *libfuse*³ in-memory file system – SAMbA-FS, which maps file contents to RDDs at runtime. SAMbA-FS enables SAMbA to benefit from in-memory RDD data without either demanding users to change their code or parsing files of a black-box transformation. As a result, SAMbA-FS transparently intercepts black-box I/O instructions regarding RDD-to-file (and vice-versa) without the expensive data flushing into the disk.

We exploit SAMbA for the capture and query of provenance on a real scientific workflow to illustrate the capabilities of our solution in a practical case study. In particular, we discuss and quantify SAMbA usage to improve runtime and *post-mortem* (after the entire workflow execution) data analytics from SciPhy [19], a bioinformatics workflow implemented in Spark running on a parallel cluster. Accordingly, the contributions of the paper are as follows.

- We present a practical roadmap for capturing provenance in Spark-based scientific workflows, in which all data aspects are managed by SAMbA routines,

- We discuss SAMbA components, where
 - data representation complies to the W3C PROV model,
 - provenance data are enriched with domain data,
 - data and activities within external raw files are captured through SAMbA-FS, and
 - the provenance database can be queried at runtime and after the execution of the workflow.
- We employ SAMbA for managing provenance in a case study of the SciPhy workflow, and results indicate:
 - SAMbA effectively provided runtime and *post-mortem* data analytics regarding prospective, retrospective, and domain data, and
 - our solution did not harm the overall workflow performance on Spark.

The remainder of the paper is organized as follows. Section II presents a background on provenance and describes the features of W3C PROV-compliant data models and analyzes related work on provenance support for Apache Spark. Section III introduces the SAMbA extension to Spark architecture, its abstractions, and implementation. Section IV presents the case study on the SciPhy bioinformatics workflow and discuss the results of SAMbA data analytics and performance. Finally, Section V provides the conclusions.

II. PRELIMINARIES AND RELATED WORK

Provenance is a key concept in workflows [14]. While provenance contributes to workflow reproducibility and data analysis, the provenance data capture process can also add a significant overhead to the workflow execution. In this section, we present the main concepts and challenges in provenance capture in DISC systems such as Apache Spark. We also discuss W3C PROV-compliant data models for the design and capture of provenance data and explain Apache Spark architecture. In particular, here we consider three types of data to be captured as follows.

Prospective provenance data. Prospective provenance data represents the definition of the workflow in terms of data and activity dependencies in the workflow specification [14], [20].

Retrospective provenance data. These data are related to the execution of activities of a given workflow and the information regarding the execution environment for the creation of a final data product [20] [14].

Domain data. Such information is the input and output data produced by the execution of workflow activities, which are specific parameters of interest from broader retrospective data. The advantages and issues of domain data regarding provenance are discussed in the study of [15]. Notice capturing relevant data from an output file is an attractive alternative between coarse (file) and fine (all file contents) grain of provenance information.

³<https://github.com/libfuse/libfuse>

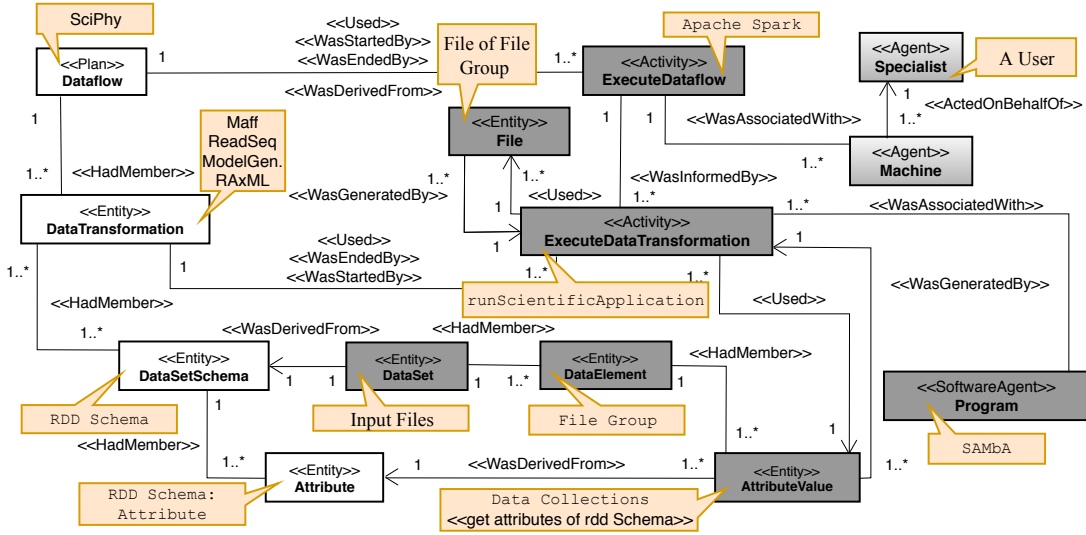


Fig. 1: The PROV-Df data model commented for the SciPhy workflow.

A. PROV-Df Data Model

Prospective and retrospective provenance and domain data can be efficiently modeled by using W3C PROV-compliant models. In particular, we inspect the state-of-the-art PROV-Df [21] features that are suitable for the representation of provenance regarding both flows of files and data elements within scientific workflows running on Apache Spark. W3C PROV-compliant PROV-Df handles prospective provenance data as a chain of data transformations (activities) and their data dependencies, which includes the execution of black-box programs. As for retrospective provenance data, PROV-Df describes the metadata about the execution of each data transformation, and enable the labeling of both intermediate and domain data regarding users' interest. Such a PROV-Df model consists of the specialization of the *Entity*, *Agent*, *Plan*, and *Activities* components of the W3C PROV standard.

Figure 1 illustrates the resulting specialized classes of PROV-Df regarding a real Spark-based workflow we use as a case study in Section IV. The model represents (i) dataflow concepts as white classes in UML diagrams, (ii) dataflow generation with domain data as dark gray classes, and (iii) environment settings as light gray classes. Other stereotypes of UML class diagrams are used for the association of each one of these classes with other PROV-DM elements. Our approach takes advantage of PROV-Df classes for representing in-memory domain data extracted from Spark.

B. Apache Spark

Apache Spark is a large-scale data processing *framework* designed for the optimization of batch and iterative parallel operations on large datasets on Scala and Java. Spark not only enables the running of applications through a chain of MapReduce-like operations [13] but also avoids significant I/O delays of traditional MapReduce routines [13]. Such an advantage is due to the fact the Spark engine knows ahead

of time the distribution of processing across the cluster so that it keeps in-memory data between iterations bypassing the MapReduce-like instructions for the reading of data from file systems and written processed results back to them. Furthermore, instead of just using Map and Reduce, Apache Spark also enables the use of other functional programming abstractions, such as filter, join and collect [12]. A typical sequence of an Apache Spark routine is as follows.

- 1) Building (a set of) RDDs by reading from distributed file systems, caching any (part of) other previous loaded RDD, or parallelizing a collection,
- 2) Processing each element on the RDD partition by using functions that (i) resemble high-level operators, such as filtering, and (ii) avoid external parameters, and
- 3) Reducing the resulting RDDs by using aggregate conditions, as counting, summing, and sorting.

The *framework* resorts to RDDs abstractions for the management of data collections consumed and produced by either operations or data transformations. The rationale is each RDD represents a set of read-only data collections partitioned across the available computational resources [22]. In the scientific workflow context, each data partition of an RDD corresponds to a Dataset Entity of the PROV-Df model. Apache Spark caches RDD contents into available memory of distributed and parallel environments, *e.g.*, clusters or clouds, so that it can be recovered whenever an RDD partition is lost.

Accordingly, Spark can employ a mechanism for collecting and querying the fault tolerance lineage of data. Such a fault-tolerant mechanism analyzes the history of data transformations and the flow of data collections (*i.e.*, data derivation in a specific application) for the rebuilding of lost partitions [12]. Although Spark can be set to *log* some data transformations [12], [16], the framework only targets the lineage of transformations as provenance data, by recording the name of the transformations. Despite the handling of both debugging

TABLE I: Comparison of alternatives for provenance support in Spark.

	Storage			Retrieval		
	Prospective	Retrospective	Domain	Prospective	Retrospective	Domain
Spark log	Text log	Text log	N/A	Web (fixed analytics) and file parsing	Web (fixed analytics) and file parsing	N/A
Newt	Relational DBMS	Relational DBMS	N/A	High-level SQL	High-level SQL	N/A
RAMP	HDFS files	HDFS files	N/A	Hive/Pig	Hive/Pig	N/A
Titian	Lineage RDD	Lineage RDD	N/A	Lineage RDD	Lineage RDD	N/A
BigDebug	Lineage RDD/ log	Lineage RDD/ log	N/A	Break/watch point	Break/watch point	N/A
SAMBA	Columnar and Relational database	Columnar and Relational database	Columnar and Relational database	High-level SQL and CQL	High-level SQL and CQL	High-level SQL and CQL

and transformation identifiers, Spark does not register other provenance-related data and provides neither data analytics nor queries regarding domain data [16]. Such a lack of support for parameterized storage and retrieval of provenance prevents scientific applications to fully benefit from detailed analyses of results during the workflow execution [13].

Moreover, Spark does not manage provenance-related data regarding the execution of black-box programs within scientific workflows. The framework resorts to an operator called *pipe* for transferring the execution control to a black-box program temporarily in such a way scientific workflows that depend on *pipe* operator for the manipulation of raw data files must read and write from standard I/O. An alternative to the *pipe* operator is redesigning (or rewriting) the entire scientific application for using the Spark Standard I/O to RDDs. However, such an approach is either unfeasible (the source code is not available in most cases) or requires the users to pick between (i) the flush of all in-memory data to disk followed by the loading of the obtained results back into main memory, or (ii) the use of a distributed file system (e.g., HDFS), which removes the major advantages of Spark: data locality and in-memory processing.

Unfortunately, existing solutions for endowing Spark with provenance capabilities focus only on specific aspects of data to be stored and retrieved, i.e., they are designed towards a particular provenance type. Thus, the gap we aim to fulfill is the simultaneous gathering of both prospective and retrospective provenance data on Spark-based scientific workflows, as well as domain data associated with black-box programs.

C. Related work in provenance support for Spark

Spark enhances the execution of several types of parallel applications, but the framework has some limitations to support provenance management. Some work provides provenance support for DISC frameworks, as Newt [23] and RAMP [24] systems. Other approaches propose entire isolated applications, such as Titian [25] and BigDebug [26]. As a summary, Table I compares the related approaches regarding their storage and retrieval support for prospective and retrospective provenance and domain data.

Spark Log [16]. Spark includes a simplified prospective and retrospective provenance system, which was designed for monitoring workflow executions. Such a monitoring process can be accessed by either a web interface or a text file. In

the *web* front-end, users may visualize the transformations’ DAG and check the stage of their execution. Although fixed statistics, such as memory consumption and elapsed time can be obtained on the front-end, data analysis require the parsing and processing of text file content.

Newt [23]. Newt aims at using provenance for finding errors in the outcome of workflows. The tool captures provenance data by using code instrumentation, in which users are responsible for setting an identifier for each data instance of the experiment and provide their relationship beforehand. Provenance data are stored into the relational and clustered MySQL DBMS in such a way high-level SQL queries can be issued for determining the tracking of all identifiers involved in experimental parts that culminate in the output error.

RAMP [24]. RAMP (*Reduce And Map Provenance*) was specifically designed for gathering provenance on MapReduce-based tasks on Hadoop. RAMP adopts Hadoop API for gathering provenance data and stores them by using HDFS external files. Therefore, specific tools like Hive [27] or Pig [28] must be used for provenance querying.

Titian [25]. Titian provides provenance support aiming at debugging Spark code. It extends the RDD abstraction through programming models, which allows users to trace backward and forward data. The tool maintains data lineage by means of a newly designed RDD, called LineageRDD, which enables data debugging at runtime focused on fault tolerance recovery on interactive scenarios. Additionally, unlike RAMP or Newt, Titian enables users to apply new transformations on data instead of just “replaying” the running of a scientific workflow.

BigDebug [26]. BigDebug follows a distinct provenance approach and provides early access to data through simulation of breakpoints and on-demand watchpoints. The tool is built on top of Titian by taking advantage of data lineage on RDDs for data debugging. It enables the retrieval of intermediate data from either break or watch points without actually stopping the overall Spark execution.

Whenever scientific workflows invoke third-party codes and applications that read or write raw files, all related solutions are unable to model, store and recover the produced contents because they were not designed for the handling of black-box

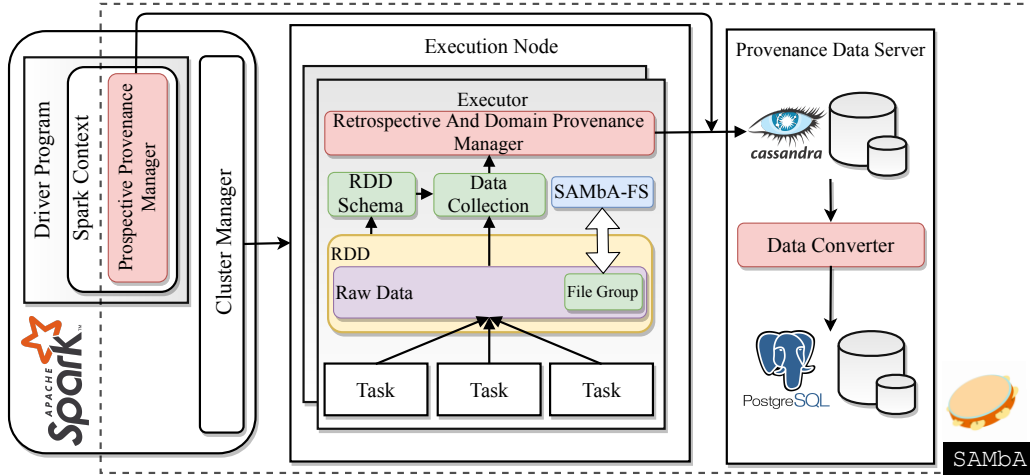


Fig. 2: SAMbA architecture. SAMbA coupled to Apache Spark enables seamless provenance management of scientific workflows.

routes that interchange data and transformations between raw files and RDDs. Moreover, they neither consider nor store the content of produced files, which prevents runtime and *post-mortem* querying of a workflow execution. Such management requires the selection, extraction, and loading of raw files' data elements, *i.e.*, items within a raw data file, into the main memory, which is not a trivial task.

Therefore, to the best of authors' knowledge [7], [25], [29], SAMbA is the first Spark-based approach that handles all three provenance data types in an integrated fashion. In particular, SAMbA follows the guidelines in [15] providing domain data analytics regarding the execution of black-box programs in Spark-based workflows.

III. SAMbA: PROVENANCE SUPPORT ON SPARK

In this section, we present SAMbA⁴, an Apache Spark extension for the gathering of the three provenance data types we discussed in Section II. Our approach relies on enveloping both RDD structure and domain-specific content at runtime so that (i) RDD-enclosure consumed and produced scientific data are stored by SAMbA in a structured way, and (ii) stored provenance data can be queried at runtime and after the execution of scientific experimentations.

In particular, SAMbA design focus on dividing scientific dataflows in such a way the RDD's structures are employed for the capture of all provenance types during the workflow parallel execution. Figure 2 details the Spark components employed in the execution of a regular scientific workflow, whereas the complementary part of Figure 2 presents the symbiotic architecture of SAMbA coupled to Spark.

SAMBa works as follows. First, Spark Context component connects to the Spark Cluster Manager for the setting of parallel executions. As the next step, Spark instantiates *Executors*, which are process-like tasks that effectively execute both (i) workflow computations (activities) and (ii) the storage of processing data within the DISC environment nodes.

Thereafter, each *Executor* can run the application instructions (as well as binary files) that were defined by the scientific workflow specification. The Spark Context component is able to monitors the running of *Executor* instances split across several *Execution* nodes.

Since Spark processes are implemented by *Executor* instances, an essential part of SAMbA design is placed upon each Spark *Executor*. Figure 2 highlights SAMbA adds one processing layer inside each *Executor* (*Retrospective And Domain Provenance Manager*) and also defines a single layer that collaborates with Apache Cluster Manager (*Prospective Provenance Manager*). Both layers are keys for the gathering of prospective and retrospective provenance alongside domain data related to the RDD content and structure. Besides those user-transparent modifications in the execution of Spark-based scientific workflows, SAMbA design also relies on another central feature: the tracking and the storage of data lineage into *Provenance Data Server*.

Provenance data are stepwise stored into the high throughput DBMS Cassandra. Additionally, SAMbA provides a *Data Converter* module to parse-and-modify provenance data from Cassandra into relational-based DBMS PostgreSQL. Such a conversion enables the handling of high-level queries based on either Cassandra CQL or PostgreSQL SQL.

A. Retrospective and Domain Provenance Manager

Figure 2 shows how Retrospective And Domain Provenance Manager interacts with system abstraction *Data Element*, which, in its turn, is related to the RDD *Schema* abstraction. The rationale that motivates the use of the RDD *Schema* abstraction is that the storage of *all* domain data elements consumed and produced by the workflow is unpractical. Therefore, experts running the scientific workflow, who know the particularities of domain data, can define their attributes of interest to be extracted and represented by implementing the RDD *Schema* abstraction.

In practice, SAMbA RDD *Schema* is an interface composed of two methods: `getFieldsNames()` and `splitData()`.

⁴Further details are available at: <https://uffescience.github.io/SAMBa/>

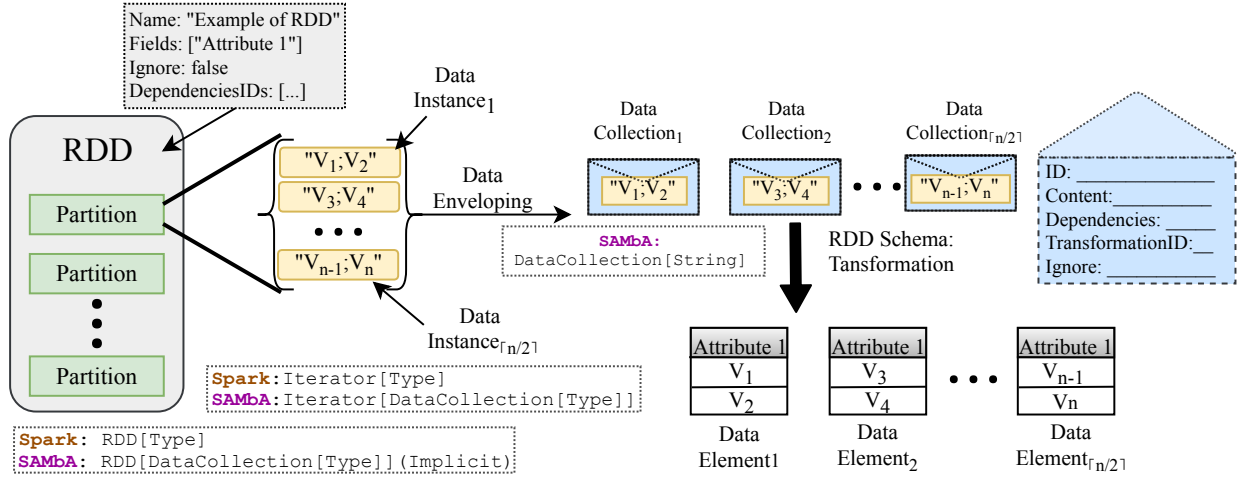


Fig. 3: SAMbA management of domain data for native and black-box transformations. SAMbA retrieves data instances from RDD partitions and envelopes them into a construct that includes content, identification, and dependencies. The RDD Schema-matching of such envelopes enables the extraction of domain data as well as the tracking of their dependencies.

Accordingly, both methods must be implemented to suit each particular scientific workflow. Method `getFieldsNames()` returns the list of names of attributes that are marked as of user interest. Likewise, method `splitData()` enables the extraction of attributes of interest within produced files according to their contents. The method returns a list of arrays, being every array associated with a single data value.

A final abstraction for retrospective and domain provenance management is the *Data Collections*, which is essentially an enveloped version of data instances within the RDD partitions. Each *Data Collection* contains a set of values for a specific attribute in the RDD Schema (Data Element). Figure 3 presents the SAMbA pipeline for the obtaining of *Data Collections* and *Data Elements*. SAMbA envelops the Data Instances of RDD partitions into *Data Collections* composed of the instances' content and their *dependencies*, which identify the previous instances that point to the current one. Such an 'envelope' strategy enables keeping track of data lineage.

The methods on RDD Schema enables SAMbA to transform *Data Collections* into *Data Elements*. Such objects include all information of a *Data Collection*, but in a structured and iterable fashion to enable the use of Spark in-memory iterative routines. Data Elements are tagged with the Collection unique identifier, which is employed for provenance data storage and querying. Therefore, the list of dependencies of a Data Collection is registered in terms of a sequence of unique identifiers and the entire lineage of data transformations is represented as a sequential list.

SAMbA provides default versions of RDD Schemas for the cases the specification of this component is not of users interest. Default RDD Schemas contain the RDD attributes that vary according to the RDD type. For instance, if a *PairRDD* is used, then the RDD Schema contains two attributes, namely *key* and *value*, which are employed for the creation of *Data Collections* composed of key-value pairs. Likewise, a standard

RDD has only one attribute we call *value*. In this case, the values are manipulated as *strings*, which are collected from the `.toString()` methods of primitive Java data types.

The last default RDD Schema of SAMbA is designed for the cases that RDDs include data produced by black-box and external programs. In this case, domain data are kept in main memory because SAMbA optimizes of I/O operations by using its SAMbA-FS component (Section III-C). Therefore, SAMbA may seamlessly apply the `splitData()` over data produced by black-box programs as of in the case of native Apache Spark transformations. The default RDD Schema in this scenario includes three attributes (*file size*, *path*, and *name*) that essentially describe the output files produced by the black-box invocation. Such information is gathered from the File Group abstraction of the SAMbA-FS component. Accordingly, only attributes of interest are stored into the Provenance Data Server.

B. Prospective Provenance Manager

The Prospective Provenance Manager layer gathers data related to transformations that are applied to *Data Collection* during the workflow execution. Although it is not mandatory, users may label each transformation to enhance the semantics of provenance queries on the resulting database. The naming of transformations is coded by calling the native method `setName()` of Spark RDD structure.

SAMbA also enables users to define their transformations of interest to be persisted into the Provenance Data Server. In this case, if users do not want to persist data, they shall invoke method `ignoreIT()` from the SAMbA RDD class structure. Such a functionality enables Prospective Provenance Manager to parameterize disk usage by avoiding the capture of poor semantic transformations, such as in the casting of data types. However, if the user sets to ignore some data, then the provenance trace will not be stored entirely. Consequently,

it may be difficult to debug the trials. Finally, SAMbA also employs the Prospective Provenance Layer for the capture of cluster manager tasks. As in the case of Retrospective and Domain Provenance Manager, such information is also stored into the Provenance Data Server.

C. SAMbA-FS – Mapping File Contents into Main-Memory

SAMbA relies on SAMbA-FS for the handling of external files and, therefore, the optimization of the gathering of domain data on Spark. The keeping of file contents into main-memory enables scientific workflows not only benefit from Spark processing but also enables SAMbA to keep track of file modification and become aware of results produced during the execution of a transformation. SAMbA-FS main abstraction is the *File Group*, which includes the specification of data types from files and maps them to in-memory RDDs.

A *File Group* has three parameters: a set of byte arrays that represent the files' content, a *HashMap* for the carrying of extra information within a *File Group*, e.g., file name, and the list of files themselves. An RDD can be created for a *File Group* by using the *FileGroupTemplate* interface that supports data loading from files into main memory. Two exclusive SAMbA operators are defined for a *File Group* RDD, namely *runCommand()* and *runScientificApplication()*. The former is used for the execution of single native OS commands, while the latter aims at running a sequence of commands whose instructions may contain third-party program invocations. Accordingly, SAMbA mounts the associated *File Group* to a temporary directory by using OS calls and binds the directory content into in-memory data.

The last optimization aspect addressed by SAMbA is the storage of outputs from black-box transformations into an external repository of versioned files. Therefore, besides the capture of domain data by RDD Schema-matching data kept into main memory by SAMbA-FS, users may choose to persist all transformation data into files after the execution of the scientific workflow. In this case, outputs of black-box transformations also become available for raw data querying.

D. Provenance Data Server

SAMbA stores both provenance and domain data into an external DBMS during the execution of Spark-based scientific workflows. SAMbA provides early access to data analytics by storing provenance data, which keeps users updated regarding intermediate results and provenance footprints. The overall idea is providing users with the status of the experiment without waiting for lengthy trials to finish. In our approach, we set the Provenance Data Server to employ the DBMS Cassandra for the storage of provenance data⁵.

Although DBMS Cassandra was the most suitable choice in comparison to other DBMSs regarding *data ingestion*, i.e., the loading of voluminous provenance data produced by scientific workflows in a very short time, its querying language

Cassandra CQL is semantically-limited in comparison to enriched SQL languages, e.g., PL-SQL, supported by relational DBMSs. We address this *trade-off* by providing a *Data Converter* component on SAMbA's architecture (Figure 2). Such a component generates an equivalent PostgreSQL relational schema regarding our Cassandra schema. Accordingly, users may also run *post-mortem* data analyses with enriched SQL statements on relational DBMSs.

IV. EXPERIMENTAL EVALUATION

We evaluate SAMbA for the capture-and-query of provenance regarding a real case scenario of a bioinformatics Spark-based workflow called *SciPhy* [19]. The workflow is essentially composed of a set of external programs for the construction and evaluation of phylogenetic trees used by drug discovery strategies. Given a set of DNAs, RNAs or amino acid sequences, *SciPhy* focuses on the finding of appropriate phylogenetic trees that can provide the best alternatives for the inference of new drug targets.

In our experiments, we inspect inputs from consolidated protozoan genomes databases and infer phylogenetic relationships of potential drug target enzymes found within such genomes. Overall execution parses 197 input data files and generates 4,531 raw data files. The workflow itself is composed of four activities, where each one carried out by the following bioinformatics black-box programs: (a) *Mafft*, (b) *ReadSeq*, (c) *ModelGenerator*, and (d) *RAXML*, respectively. All these applications access data through files.

We exploit all SAMbA modules to illustrate the capabilities of our solution in this practical case study. SAMbA executes *SciPhy* and stores all provenance data types in a PROV-Df-compliant database, whose elements are illustrated in Section II. We executed the *SciPhy* workflow six times and measured SAMbA overall performance regarding provenance data management by following two complementary criteria:

- 1) The computational effort, which includes the time spent for the capture of provenance data in comparison to the time of running the entire *SciPhy* workflow on Apache Spark. We also quantified the impact of the associated settings, such as Git and SAMbA-FS, for the handling of external files,
- 2) Data analytics, which evaluates the SAMbA capabilities for providing both runtime and *post-mortem* provenance queries. We present SAMbA online query results over an interactive *Web* front-end and also evaluate eight SQL provenance queries defined by domain experts of the *SciPhy*. Such queries retrieve all aspects of gathered information, i.e., prospective provenance, retrospective provenance, and domain data.

A. Setting up SAMbA

Since *SciPhy* depends on black-boxes applications for the running of key transformations, SAMbA RDD Schema must contain the attributes of interest from external files to be extracted and stored. Figure 5 shows an example of a RDD Schema for the management of files related to *ReadSeq*

⁵The columnar schema of SAMbA for DBMS Cassandra is available at: <https://gist.github.com/thaylongs/5b2e1cbce7eeb2c8fecca9befa664c89>

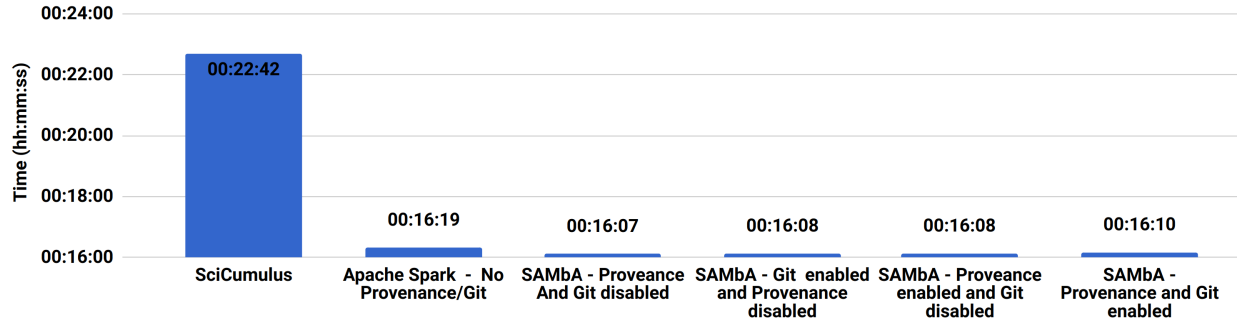


Fig. 4: Average elapsed time of SciPhy workflow running on SciCumulus, Apache Spark and SAMbA.

program. In this case, we select three domain data attributes `FILE_NAME`, `NUM_ALIGNS`, and `LENGTH` to be represented as SAMbA *Data Elements* values. Accordingly, Retrospective and Domain Provenance Manager becomes capable of reading the RDD contents and persist such data of interest into the Provenance Data Server.

Next, we set the SAMbA-FS File Group abstractions for keeping raw files content into main memory. Figure 6 presents an example of a File Group instantiation on SAMbA, in which the files listed in `inputFastaList.txt` are loaded into an Apache Spark RDD. After the setting of RDD Schemas and File Groups, SAMbA runs the SciPhy workflow by invoking third-party programs through the `runScientificApplication()` routine, which enables external applications to run in a managed flow.

B. The Computational Effort

In our first experiment, we measured the time spent by our proposal for the capture of the provenance data types in SciPhy. All workflow executions were performed on LoboC⁶ cluster by consuming the same input dataset. We set the cluster to use 48 processing cores, where 44 cores were employed for

the workflow execution and the remaining 04 cores were put in charge of provenance persistence into DBMS Cassandra.

We compared our solution against two baselines approaches: (i) SciCumulus SWfMS [30] with parallel functionalities, which does not provide in-memory processing, and (ii) standalone Apache Spark, which was employed without any of SAMbA extensions. Figure 4 presents the average elapsed time (standard deviation was less than .5s in all cases) for the execution of SciPhy on SciCumulus, standalone Apache Spark and SAMbA with four distinct settings: (i) both provenance and Git support disabled, (ii) provenance disabled and Git enabled, (iii) provenance enable Git disabled, and (iv) both provenance and Git enabled.

Results show SAMbA outperformed both SciCumulus and standalone Apache Spark for all configurations. In particular, SciCumulus was nearly 40% slower than full set SAMbA (since it writes and reads all data to and from the disk), whereas Apache Spark without provenance support was about 1% slower than SAMbA. Such behavior is due to the SAMbA-FS optimization that avoids I/O operations on disk as much as possible. Accordingly, SAMbA with SAMbA-FS was able to execute the workflow, capture and store provenance data into DBMS Cassandra and external files and still outperform standard Spark execution. Provided optimization aspects, such as the in-memory file system, are properly addressed, such an outcome indicates our Spark-based provenance management solution for scientific applications seems not to jeopardize the workflow performance. Therefore, SAMbA was able to efficiently manage the files' contents regarding black-box transformations at expenses of a little memory overhead, as well as persist extracted and structured information (the *Data Elements*) into external DBMSs.

C. Visual Provenance Analytics

SAMBa implements three dynamic reports for evaluation of scientific workflows at runtime. Reports are available by means of a *web* friendly interface, which exhibits formatted provenance data from DBMS Cassandra in a structured fashion and enables user-interaction. SAMbA *online* data analytics cover the following aspects:

- 1) **Retrospective report:** Presents the start and end time (whenever available) of a given workflow execution.

```
// Schema Implementation for ReadSeq
class ReadSeqSchema extends SingleLineSchema[FileGroup] {

  override def getFieldsNames(): Array[String]
    = Array("FILE_NAME", "NUM_ALIGNS", "LENGTH")
}
```

Fig. 5: A fragment of a SAMbA RDD Schema.

```
// SciPhy experiment using Spark
val fileGroup = new SparkContext(
  new SparkConf().setMaster("local [4]").setAppName("SciPhy")
  .setScriptDir("/workflows/sciphy/ scripts ")
).fileGroup( parserInputFile (" inputFastaList . txt"): _*)// File Group
```

Fig. 6: A fragment of a SAMbA-FS File Group.

⁶Cluster specification: <https://www.nacad.ufrj.br/en/recursos/sgiicex>

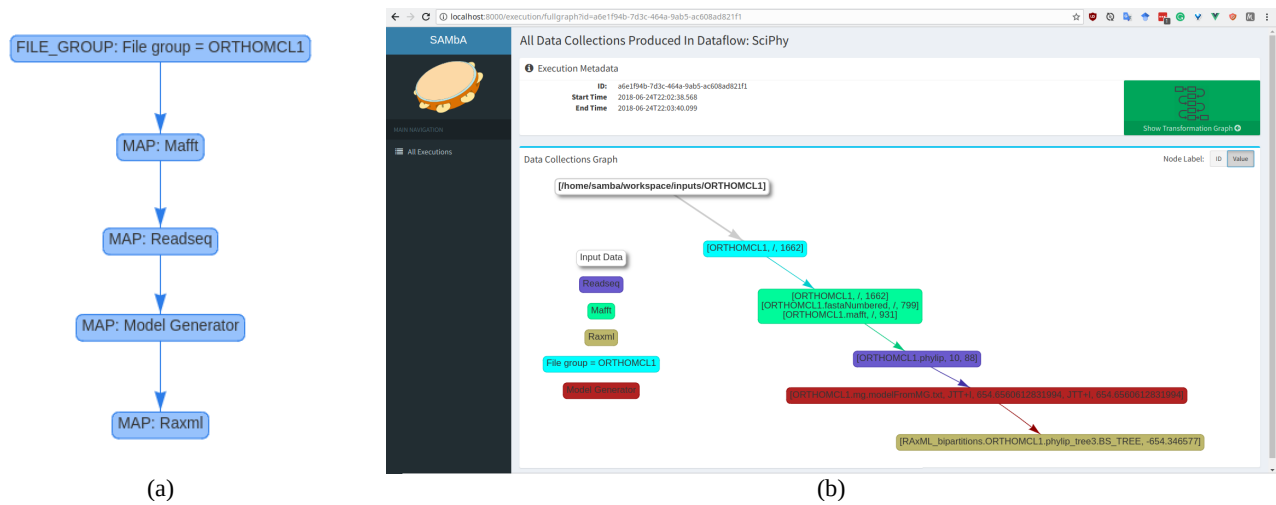


Fig. 7: Dataflow transformations graph for an input set on SciPhy. (a) Transformation flow of a SciPhy execution. (b) Web interface for dataflow visualization of an input set on SciPhy.

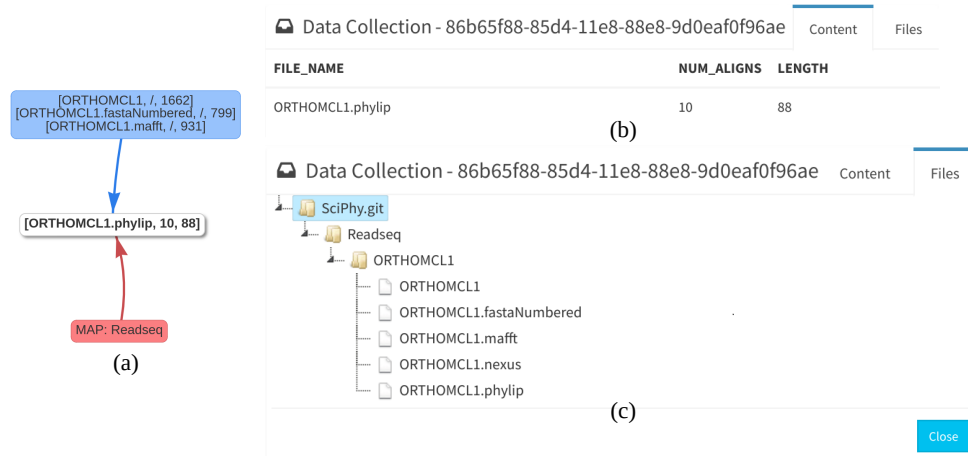


Fig. 8: Data Collections and transformations. (a) Partial transformation dataflow regarding the ReadSeq application. (b) RDD Schema-matching of consumed Data Elements. (c) Output files of the transformation executed by the black-box program.

- 2) Prospective report: Presents the transformation graph of a given workflow execution.
- 3) Domain Data report: Presents information about domain data associated with every activity of the workflow.

Figure 7(a) presents an excerpt of the web interface that contains the transformation graph for a user-selected execution of SciPhy – Prospective report. Such a graph includes the name of the transformation (activity) and its type, *i.e.*, Map, Reduce, etc. Users may expand each transformation based on the Prospective report to access the repository of domain data (raw files) produced by a SciPhy specific execution. Figure 7(b) shows the Retrospective report (top-right) and the transformation flow in details for a given input set (bottom-right), where users can expand and explore domain data associated to each node of the graph – Domain Data report. Users may also navigate through the dataflow regarding specific input sets for the investigation of every transformation as well as visualize their results.

By clicking on a transformation node, a new page is dynamically loaded with the partial dataflow associated with the activity itself – Prospective report. In the interface, users may verify the Data Collection involved in the transformation. Consumed and produced Data Collections are initially collapsed on the interface, as illustrated in the blue and white rectangles of Figure 8(a). The report also shows the Data Elements in a table format for each user-selected Data Collection (Figure 8(b)). Additionally, if the transformation is carried out by a black-box program, the output values are stored on external files that can be accessed through a directory hierarchy – Figure 8(c).

D. Data Analytics – Provenance Queries

In this experiment, we evaluate eight queries defined by SciPhy experts to verify whether SAMbA gathered provenance data are suitable and researchable. The queries were categorized regarding the data aspects they resemble. Figure 9

divides them into sets regarding prospective, retrospective, and domain data-based provenance queries.

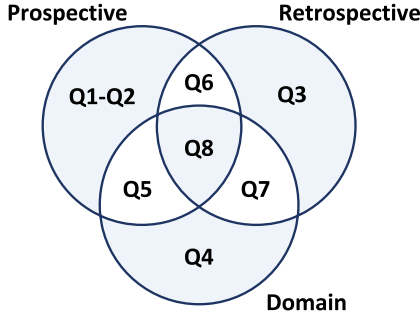


Fig. 9: Query categories and their provenance aspects.

Prospective provenance data queries analyze *SciPhy* dataflow specifications, such as transformations in the workflow and attributes consumed and produced by transformations, their associated schema and data dependencies. Retrospective provenance data queries investigate dataflow paths, such as the history of data processing activities, regarding fixed and specific executions of the *SciPhy* workflow. Finally, domain data queries investigate *SciPhy* domain data. We opt to keep data of interest stored in both raw files and DBMS Cassandra so that *SAMbA* also persisted the information of raw files into the Provenance Data Server.

TABLE II: Description of the eight expert-provided queries issued on *SAMbA* Provenance Data Server.

Query	Description
Q1	Retrieves all data transformations of the specific execution.
Q2	Retrieves the name and type of all attributes regarding a specific output dataset.
Q3	Retrieves all files generated within the last two hours.
Q4	Retrieves the phylip name and the number of alignments for the execution of <i>ModelGenerator</i> program, whose results reached a value for the first obtained model greater than 1,500.
Q5	Retrieves all generated phylogenetic trees, <i>i.e.</i> , data elements in dataset <i>ds_raxml</i> , after the execution of <i>SciPhy</i> .
Q6	Retrieves the initial and final time of a <i>SciPhy</i> runnings.
Q7	Retrieves the elapsed time of <i>SciPhy</i> workflow executions that generates, at least, four phylogenetic trees in which the minimum bootstrap is higher than 90%.
Q8	Retrieves the algorithm used in the <i>Model Generator</i> transformation that generates phylogenetic trees with average bootstrap greater than 70%.

Provenance data were converted and stored into a *relational schema* in such a way we can execute the expert-provided queries as *post-mortem* searches. In this context, we were able to evaluate the combination of prospective, retrospective, and domain data as generic high-level SQL statements to enrich data analytics. Table II describes the eight issued queries following the categorization of Figure 9. The average size of PostgreSQL relational *schema* obtained from the *SAMbA* Provenance Data Manager was around 2.1MB. Figure 10 details the average time spent for the execution of the most representative PostgreSQL queries (regarding filtering and join selective predicates) after ten repetitions without caching.

Query **Q6** was employed as a baseline because it executes a full table scan regarding a relation that contains all workflow executions. In the experiments, Queries **Q2** (more selective) and **Q4** (less selective) were up to 42.04% and 236.69% slower than Query **Q6**, respectively.

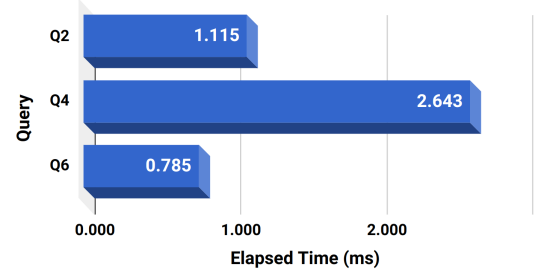


Fig. 10: Average time of representative PostgreSQL queries.

V. CONCLUSIONS AND FUTURE WORK

Apache Spark is a widely adopted DISC system for big data analytics and its in-memory processing, large installation base, and support has attracted IO intensive scientific workflows to benefit from data parallelism, scheduling strategies and fault tolerance mechanisms. However, Apache Spark is still limited in supporting data provenance and analysis. *SAMbA* contributes to current approaches in providing provenance to Spark by capturing prospective, retrospective, and domain data provenance during the workflow execution. In addition, *SAMbA* also extracts raw data selected from files by an efficient file system named *SAMbA-FS*.

We modeled the gathered provenance data by using the *PROV-Df* model. Moreover, *SAMbA* supports provenance management by properly handling both the structure and content of Spark RDDs. Experts can instantiate RDD Schemas for the extraction of data elements produced by black-boxes transformations. *SAMbA* uses an in-memory file system that improves I/O operations demanded by black-box programs of scientific workflows and efficiently provides users with runtime querying and *post-mortem* data analytics.

We evaluate *SAMbA* in a practical scientific workflow on bioinformatics domain called *SciPhy* and results indicated: (i) *SAMbA* did not jeopardize standard Apache Spark performance, and (ii) *SAMbA* efficiently provided runtime and *post-mortem* data analytics.

We believe that *SAMbA* can make the use of Apache Spark even more attractive for the classes of IO-intensive scientific workflows. Future work includes the evaluation of *SAMbA* with different scientific workflows, such as Montage [31]. We also plan to investigate how to collect provenance from Data Frames as many applications are moving toward to them.

ACKNOWLEDGMENTS

Authors would like to thank CAPES (Thaylon Guedes' scholarship grant), CNPq (grant 308966/2015-5) and FAPERJ (grant E-26/203.215/2016). Authors would like to thank Kary Ocana for her support on the *SciPhy* workflow.

REFERENCES

- [1] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "Scientific Workflow Scheduling with Provenance Data in a Multisite Cloud," *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, vol. 33, pp. 80–112, 2017.
- [2] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "noWorkflow: a Tool for Collecting, Analyzing, and Managing Provenance from Python Scripts," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1841–1844, 2017.
- [3] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using Python," *Advances in Water Resources*, vol. 34, no. 9, pp. 1124–1139, 2011.
- [4] S. Jha, J. Qiu, A. Luckow, P. Mantha, and G. C. Fox, "A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures," in *2014 IEEE International Congress on Big Data*. IEEE, 2014, pp. 645–652.
- [5] F. S. Chirigati, D. E. Shasha, and J. Freire, "ReproZip: Using Provenance to Support Computational Reproducibility," in *Presented as part of the 5th USENIX Workshop on the Theory and Practice of Provenance*. Lombard, IL: USENIX, 2013, pp. 1–4.
- [6] L. Murta, V. Braganholo, F. Chirigati, D. Koop, and J. Freire, "noWorkflow: Capturing and Analyzing Provenance of Scripts," in *International Provenance and Annotation Workshop*. Springer, 2014, pp. 71–83.
- [7] E. Deelman *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.
- [8] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. L. Lusk, and I. T. Foster, "Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing," in *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013, Delft, Netherlands, May 13-16, 2013*, 2013, pp. 95–102.
- [9] Z. Zhang, K. Barbary, F. A. Nothaft, E. R. Sparks, O. Zahn, M. J. Franklin, D. A. Patterson, and S. Perlmutter, "Kira: Processing Astronomy Imagery Using Big Data Technology," *IEEE Transactions on Big Data*, pp. 1–14, 2018.
- [10] H. Mushtaq, F. Liu, C. Costa, G. Liu, P. Hofstee, and Z. Al-Ars, "SparkGA: A Spark Framework for Cost Effective, Fast and Accurate DNA Analysis at Scale," in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. New York, NY, USA: ACM, 2017, pp. 148–157.
- [11] B. S. Kanzki, V. Dupuy, C. Urvoy, F. Belghait, A. April, F. Harvey, F.-C. Marois-Blanchet, M. S. Phillips, J. Tremblay, and P. Hamet, "GOAT: Genetic Output Analysis Tool: An Open Source GWAS and Genomic Region Visualization Tool," in *Proceedings of the 6th International Conference on Digital Health Conference*. New York, NY, USA: ACM, 2016, pp. 55–59.
- [12] 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," pp. 1–14, 2012.
- [13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," *HotCloud*, vol. 10, no. 1-7, p. 95, 2010.
- [14] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for Computational Tasks: A Survey," *Computing in Science & Engineering*, vol. 10, no. 3, pp. 11–21, May 2008.
- [15] D. de Oliveira, V. S. Sousa, and M. Mattoso, "How Much Domain Data Should Be in Provenance Databases?" in *7th USENIX Workshop on the Theory and Practice of Provenance, Edinburgh, Scotland, UK, July 8-9, 2015*, 2015, pp. 8–14.
- [16] M. Armbrust, T. Das, A. Davidson, A. Ghodsi, A. Or, J. Rosen, I. Stoica, P. Wendell, R. Xin, and M. Zaharia, "Scaling Spark in the Real World: Performance and Usability," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1840–1843, 2015.
- [17] S. Blanas, K. Wu, S. Byna, B. Dong, and A. Shoshani, "Parallel Data Analysis Directly on Scientific File Formats," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 2014, pp. 385–396.
- [18] J. Wang, D. Crawl, S. Purawat, M. Nguyen, and I. Altintas, "Big data provenance: Challenges, state of the art and opportunities," in *2015 IEEE International Conference on Big Data (Big Data)*, Oct 2015, pp. 2509–2516.
- [19] K. A. Ocaña, D. de Oliveira, E. Ogasawara, A. M. Dávila, A. A. Lima, and M. Mattoso, "SciPhy: A Cloud-based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes," in *Brazilian Symposium on Bioinformatics*. Springer, 2011, pp. 66–70.
- [20] T. McPhillips *et al.*, "YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts," *International Journal of Digital Curation*, vol. 10, no. 1, pp. 298–313, may 2015.
- [21] V. Silva, J. Leite, J. J. Camata, D. De Oliveira, A. L. Coutinho, P. Valduriez, and M. Mattoso, "Raw data queries during data-intensive parallel workflow execution," *Future Generation Computer Systems*, vol. 75, pp. 402–422, 2017.
- [22] 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*. Berkeley, CA, USA: USENIX Association, 2012, pp. 1–14.
- [23] D. Logothetis, S. De, and K. Yocum, "Scalable Lineage Capture for Debugging DISC Analytics," in *Proceedings of the 4th Annual Symposium on Cloud Computing*. ACM, 2013, pp. 17 – 32.
- [24] R. Ikeda, H. Park, and J. Widom, "Provenance for Generalized Map and Reduce Workflow," pp. 1–11, 2011.
- [25] M. Interlandi, K. Shah, S. D. Tetali, M. A. Gulzar, S. Yoo, M. Kim, T. Millstein, and T. Condie, "Titan: Data Provenance Support in Spark," *Proceedings of the VLDB Endowment*, vol. 9, no. 3, pp. 216–227, 2015.
- [26] M. A. Gulzar, M. Interlandi, S. Yoo, S. D. Tetali, T. Condie, T. Millstein, and M. Kim, "BigDebug: Debugging Primitives for Interactive Big Data Processing in Spark," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 784–795.
- [27] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive A Petabyte Scale Data Warehouse Using Hadoop," in *International Conference on Data Engineering*, March 2010, pp. 996–1005.
- [28] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. ACM, 2008, pp. 1099–1110.
- [29] D. Barseghian, I. Altintas, M. B. Jones, D. Crawl, N. Potter, J. Gallagher, P. Cornillon, M. Schildhauer, E. T. Borer, E. W. Seabloom *et al.*, "Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis," *Ecological Informatics*, vol. 5, no. 1, pp. 42–50, 2010.
- [30] D. de Oliveira, E. Ogasawara, F. Baião, and M. Mattoso, "SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows," in *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, July 2010, pp. 378–385.
- [31] J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. Prince *et al.*, "Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking," *International Journal of Computational Science and Engineering*, vol. 4, no. 2, pp. 73–87, 2009.