Algebraic Dataflows for Big Data Analysis*

Jonas Dias[1], Eduardo Ogasawara[1,2], Daniel de Oliveira[3], Fabio Porto[4], Patrick Valduriez[5] , Marta Mattoso[1]
[1]Federal University of Rio de Janeiro - COPPE/UFRJ, [2]CEFET/RJ, [3]Fluminense Federal University - UFF
[4]LNCC National Laboratory for Scientific Computing, Brazil
[5]INRIA and LIRMM, France
{jonasdias, marta}@cos.ufrj.br, eogasawara@cefet-rj.br, danielcmo@ic.uff.br
fporto@lncc.br, Patrick.Valduriez@inria.fr

Abstract— **Analyzing big data requires the support of dataflows with many activities to extract and explore relevant information from the data. Recent approaches such as Pig Latin propose a high-level language to model such dataflows. However, the dataflow execution is typically delegated to a MapReduce implementation such as Hadoop, which does not follow an algebraic approach, thus it cannot take advantage of the optimization opportunities of PigLatin algebra. In this paper, we propose an approach for big data analysis based on algebraic workflows, which yields optimization and parallel execution of activities and supports user steering using provenance queries. We illustrate how a big data processing dataflow can be modeled using the algebra. Through an experimental evaluation using real datasets and the execution of the dataflow with Chiron, an engine that supports our algebra, we show that our approach yields performance gains of up to 19.6% using algebraic optimizations in the dataflow and up to 39.1% of time saved on a user steering scenario.**

Keywords: big data; dataflow; algebraic workflow; performance evaluation.

I. INTRODUCTION

Big data analyses are critical in business and scientific data processing. Analyses over big datasets may depict more accurate results regarding Web data, users behavior, and challenging scientific researches [1]. They involve the execution of many activities, such as: programs to collect and extract data for analysis, data cleaning procedures, annotations, data aggregations, core programs to perform analyses and tools to visualize and interpret the results. These activities are typically organized in a dataflow and may use traditional programs for information retrieval, data mining, computational mechanics or novel approaches developed by scientists.

Many users currently use MapReduce [2] implementations to perform big data analysis in a High Performance Computing (HPC) environment (typically clusters). The only thing they have to concern about is to code Map and Reduce functions while the framework provides data parallelism. Hadoop† is a very popular open source implementation of MapReduce. However, it does not provide features to model complex dataflows [3]. Dataflows are commonplace in both business analytics and scientific experimentation, where they are typically called scientific workflows.

One approach to develop dataflows for big data analysis is Pig Latin [3], which combines MapReduce and SQL programming styles and provides native operations to load, process, filter, group, and store data in Hadoop. Pig, the implementation of Pig Latin, considers the optimization of the dataflow logic [4] including the optimized placement of filter operations, as well as collapsing two filters into a single Map operation. These optimizations are suitable for the native Pig Latin operations (like filter and group). However, when the user (that can be a scientist or a business user) needs to use customized code, Pig Latin requires using User Defined Functions (UDF), which hide the behavior of the customized program [5]. When it comes to task dispatching and resource allocation, Pig relies on Hadoop to handle MapReduce tasks. Unfortunately, the Hadoop execution plan does not take advantage of the declarative features of Pig Latin, providing little room for dataflow runtime optimization. There are several efforts aiming at improving Hadoop performance [6–8]. However, they focus on improving the way Hadoop works, changing the way data is stored and accessed and tuning Hadoop parameters. Since Hadoop is not aware of the entire dataflow and the history of executions, it cannot optimize the execution, by taking advantage of data movements and task execution behavior.

We believe that concepts from well-established distributed query optimization in Relational Database Management Systems (RDBMS) [9] may well improve big data processing. RDBMS typically exploit their knowledge regarding the behavior and implementation of relational algebra operators, as well as database statistics. This knowledge allows for the RDBMS to know how a given operation consumes and produces data, thus making it possible to create equivalent query execution plans and estimate the cardinality of intermediate results. Then, based on a cost function, the RDBMS can pick the best plan and adjust it at runtime.

Compared with RDBMS, current dataflow processing approaches lack information regarding the behavior of the activities they are executing and the statistics of their execution. For the execution statistics, we can benefit from data provenance techniques for scientific workflows [10]. Several studies [11–14] point to the importance of bringing provenance to MapReduce execution frameworks. Provenance, as long as it is available for querying during runtime, allows for the analysis of partial results, thus leveraging the dynam-

* Work partially funded by CNPq, CAPES, FAPERJ and INRIA (SwfP2Pcloud and Hoscar projects) and performed (for P. Valduriez) in the context of the Computational Biology Institute (www.ibc-montpellier.fr).

† http://hadoop.apache.org/

ic steering of workflows by users [15]. Dynamic steering of workflows is associated to features that allow for changes and adjustments in the workflow during its execution so the workflow engine dynamically adapts the execution to the current configuration or scenario. By steering workflows, scientists may change parameters such as filtering criteria during the execution of the workflow, *e.g.* to reflect better results for the experiments [16]. Based on provenance, it is also possible to estimate the execution time of an activity, the amount of data produced by a Map function or the aggregation factor of a Reduce function, for example. Prior to dataflow execution, Hadoop cannot know whether the Map function programmed by the user always produces a single output for each input or several outputs for each input. This lack of data statistics typically yields dataflows with a static execution strategy.

To tackle this lack of data statistics, some approaches [17] use static code analysis to optimize the dataflow based on the behavior of the activities. Other approaches require the development of the dataflow through a well defined API [18] so the knowledge over data behavior is captured by the interpreter. Both approaches may be the best options for applications such as text searching or novel web mining approaches that use recently developed Java code that can be easily used as an UDF.

In this paper, we discuss how algebraic workflows may support big data analysis enabling more flexible execution approaches through provenance data. Our workflow algebra [19] is inspired by the relational algebra for databases and provides a uniform dataflow representation that adds metadata to dataflow activities that leverage data statistics. In the algebraic approach, data processed by activities is modeled as relations while activities are mapped to algebraic operators. This approach is suitable for big data workflows (*e.g.* bioinformatics ones [20,21] and oil and gas [19,22]). The algebraic representation brings many opportunities for workflow optimization through algebraic transformations and leverages the management of data parallelism through different execution strategies. Furthermore, the execution model for the workflow algebra is tightly coupled to the provenance database, enabling real-time provenance. Querying provenance at runtime allows for adaptive scheduling based on provenance statistics [21] and other features such as user steering [16]. Such features avoid a black-box execution and may produce faster and more efficient workflows. We show performance comparisons between different execution strategies and different execution plans for the same workflow, obtained through algebraic optimizations. Our experimental evaluation shows up to 19.6% of performance improvements. In addition, our experiments show up to 39.1% of time saved on a user steering scenario.

This paper is organized as follows. Section II introduces our approach with a motivating example. Section III briefly describes the algebraic approach for data-centric workflows, its execution model and real-time provenance. Section IV presents our experimental evaluation and Section V concludes.

II. MOTIVATING EXAMPLE

To illustrate how a workflow for big data analysis can be modeled using our algebra [19], a dataflow that searches for trends in scientific publications was chosen. This dataflow is called BuzzFlow, and it uses data collected from bibliography databases such as DBLP[‡] and PubMed[§], to show trends and their correlations in the published papers during the last 20 years. BuzzFlow is an example of a general big data analysis dataflow and will be consistently used in the remainder of the paper as proof of concept.

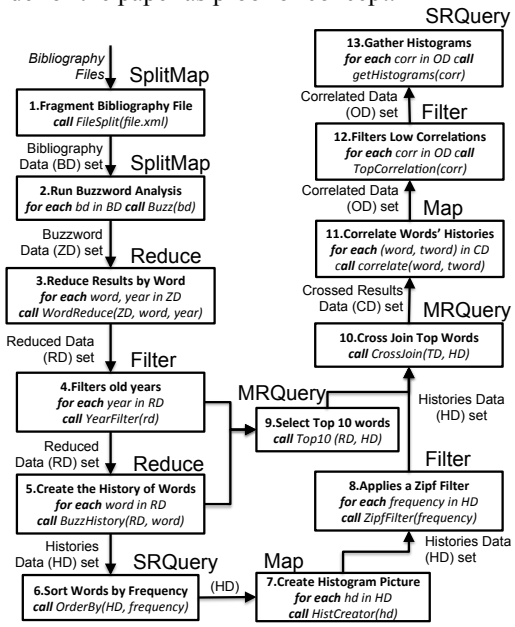

Figure 1. BuzzFlow dataflow

BuzzFlow is composed of 13 activities as shown in Figure 1. The boxes represent the activities of the workflow and the arrows represent the datasets that flow between them. Activity 1 splits the input file into fragments to enable parallel processing of the dataset. Activity 2 produces a list of words and its frequency on the year, *i.e.* it produces a set of output data that contains the year, the word and its frequency. Activity 3 aggregates the frequency value for each year and word. Activity 4 removes every year before 1992. Then, activity 5 aggregates the output relation by *Word* and writes, in a history file, the frequency that the word had on every year since 1992. Activity 5 also stores the total frequency of the word to be sorted in descending order by activity 6. For each word in the results, activity 7 creates a histogram of the data contained in the history file, plotting the frequency of the word along the years. The results obtained so far follows a Zipf distribution. Thus, activity 8 selects the interesting information based on the Zipf's law and decides whether a given input tuple should continue in the dataflow or not. Activity 9 selects ten of the most frequent terms on recent years (2011 and 2012). The activity selects the history file of those terms produced in activity 5. Next, activity 10

[‡] http://www.informatik.uni-trier.de/~ley/db/

[§] http://www.ncbi.nlm.nih.gov/pmc/

makes a cross product between the outputs of activities 8 and 9. The idea is to correlate the "hot terms" of the recent years with the buzzwords of the last two decades. Thus activity 10 produces the input data set for activity 11. Each input of activity 11 contains a "hot topic" of a recent year (*tword*) and a buzzword from the past two decades (*word*). It compares their histories and calculates the correlation between the terms. Activity 12 filters only the pairs (*tword*, *word*) with higher correlations (above 95%). Then, activity 13 selects the histograms for these pairs.

This example shows that a meaningful big data analysis requires a good number of activities and each activity has a particular behavior regarding how it consumes and produces data. Some activities consume a single input and produce many. Some activities only modify the data, *e.g.* activity 6 or 7, while others may filter the data out, *e.g.* activities 4 or 8. Some activities also group data into a single output, *e.g.* activities 3 or 5. Without understanding the behavior of the activities, it may be hard to estimate data cardinality prior the execution of the dataflow. Thus, our algebraic workflow approach is appropriate for big data analysis. Furthermore, provenance data may provide tremendous help to scientists by allowing for their analysis during runtime and steering of their workflows.

III. Algebraic Big Data Processing

In relational databases, relational algebra is the main support for query execution and optimization. We believe it is important to review and possibly inherit successful features from the well-established relational algebra for big data processing. Pig Latin already uses characteristics from relational databases. In an RDBMS, algebraic operators (like select, project, join, union) have precise semantics but extensions done through UDF do not, which requires complex optimization strategies such as predicate migration [23]. With our workflow algebra, the main properties of relational algebra are kept while supporting general-purpose compiled programs and scripts without relying on UDF.

Our algebraic approach for data-centric workflows [19] associates an algebraic operator (ϕ) to activities of the dataflow. An algebraic operator expresses the behavior of an activity with respect to how it consumes the input data and produces the output data. For an output relation T that follows a schema $\mathfrak{I}$ we write $T \leftarrow \phi(Y, \overline{\omega}, \overline{R})$. We say that $\overline{R} = \{R_1, R_2, \ldots, R_n\}$ is the set of input relations with schemas $\{\mathcal{R}_i, \ldots, \mathcal{R}_n\}$, $\overline{\omega} = \{\omega_1, \omega_2, \ldots, \omega_m\}$ is the set of additional operands and $Y <\{\mathcal{R}_i, \ldots, \mathcal{R}_j\}, \mathfrak{I}_i, UC>$ is the dataflow activity that executes a given computational unit (*UC*) to produce an output relation compatible with schema $\mathfrak{I}$. Observe, however, that a UC may be a legacy or third-party software not amenable to the relational representation of the data.

Activities always consume and produce relations, *i.e.* sets of tuples. For a single input tuple, *Map* activities produce a single output tuple whereas *SplitMap* may produce many. Reduce aggregates tuples into a single one based on a grouping attribute G_a. Filter decides if an input tuple must go to the output or not. *SRQuery* and *MRQuery* execute relational algebra expressions: *SRQuery* for single relation queries (*i.e.* selection and/or projection) while *MRQuery* involves multiple relations (cross product or join). For details please refer to [19]. The algebraic operators associated to BuzzFlow activities are shown on Figure 1.

Although relations represent data, the content of the files are not inserted into relations. The relations store only metadata with the references to files with binary or unstructured data. However, some important parameters and results may be extracted from files to be inserted in a relation to enable queries over experiment data results in provenance.

A. Execution Model

The parallel execution model of the algebraic approach for workflows is based on the activation concept. A*ctivation* is the finest unit of data that is necessary to execute an instance of a given activity [19]. It includes the minimum set of tuples from the input relation that is necessary to execute the activation based on the operator that is associated to the activity. For example, for a *Map* operator, only one tuple per activation is necessary. However, for a *Reduce*, the activation may contain a horizontal fragment of the input relation. The activation also contains information about the *UC* and the reference to the output tuple or relation. The execution of a given activation is composed of three steps: data instrumentation, program invocation and result extraction [19]. The instrumentation step extracts data from the input relation to build command lines to be invoked. The program invocation should execute the command line on different nodes/processors of the execution environment, such as a cluster or a cloud. As the activations finish, the results are extracted and inserted in the output relation. Although data instrumentation is an automated process, result extraction may need the development of an *extract* component. Since the activity *Correlate* instantiates a general compiled program or script (correlate), the output can be domain-specific. The extract component is executed per activation and is responsible to extract desired results in the form of a tuple or relation.

As long as input data is available for an activity, activations are produced and ready to execute. However, the execution model allows for the scheduling and execution of activations following four different strategies. The strategies are based on two scheduling policies and two distribution strategies. The *blocking* policy (*i.e. First-Activity-First* [19]) says that all the activations of a given activity are executed prior the execution of any activation of further activities. If an activity *B* consumes data produced by an activity *A*, then *B* only executes after all activations of *A* are finished. Alternatively, there is the *pipeline* policy (*i.e. First-Tuple-First* [19]). If an activity *C* consumes data from *B*, which consumes data from *A*, the policy executes an activation of *A* followed by an activation of *B* and then an activation of *C*. There are also two distribution strategies: the *Static* strategy sends bags of activations for each computing resource that requests tasks to process. The *Dynamic* strategy sends a single activation per request. The combination of these characteristics yields the four execution strategies: *Static-*

Blocking, *Static-Pipeline*, *Dynamic-Blocking* and *Dynamic-Pipeline* (please refer to [19] for more details).

Although we propose a novel approach for workflow design and execution, we believe it is complementary to other approaches such as Pig, as each approach has its specific features that are better for a given set of applications. For example, in Figure 2 we show a hypothetical Pig dataflow and the respective execution plan in Hadoop (Hadoop Jobs). But instead of running jobs in Hadoop, Pig could simply make a call to Chiron, our workflow engine that implements the proposed algebra (Figure 2 shows a translation of the Pig workflow to algebra). In this comparison, Hadoop executes three static MapReduce jobs while the algebraic approach is able to stipulate different execution strategies (static-blocking, dynamic-pipeline and dynamic-blocking) for different fragments of the workflow. The choice for the best execution strategy may be related to intermediate data cardinality and provenance statistics. The possibility to choose between different execution strategies brings flexibility and better performance.

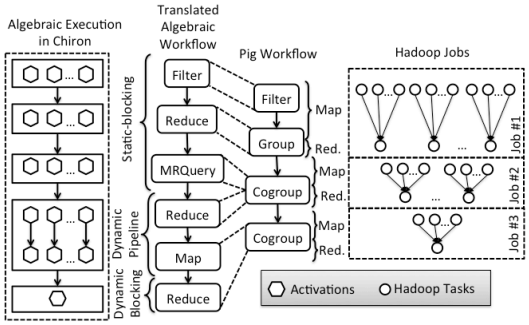

Figure 2. Comparison between execution strategies.

B. Real-time Provenance Support

Since the algebraic approach for workflows is an extension of relational algebra, it is possible to use a database to manage the workflow relations. Thus, it is easier to integrate all information of a workflow within the provenance database. Chiron workflow engine uses provenance data to manage the whole workflow execution, and for instance, decide what to execute next based on the previous activity execution. Consequently, the provenance database reflects the execution history in a structured way, and becomes available to scientists for querying at runtime.

Real-time provenance support allows scientists to monitor the workflow, perform preliminary analysis on the experiment while it is running and take actions regarding execution. For example, let us consider a scientist is executing BuzzFlow in the cloud. By querying the provenance database, she knows that activity *ZipfFilter* is running and several activations already finished, thus there are results available. She queries the output relation of *ZipfFilter* and checks the path to several histogram files in the cloud storage. The scientist can now stage out the histogram files and analyze them. If she finds that the filter is allowing irrelevant data to pass, she can decide to make the filter more rigorous by changing the parameters of the command line program. If the scientist changes the parameter in the provenance database, all remaining activations will use the new parameter values. Additionally, the scientist can reset the status of the *ZipfFilter* activity, so it will be executed all over again with the new parameters.

Scientists can also look for errors in the workflow and check possible causes using provenance data. Depending on the error, it may be possible to fix it by changing a misconfigured parameter or activity. All such capabilities are related to steering of workflow executions by the user [15]. Steering of workflows involves several complex issues in scientific workflows execution that are related to the best approach to monitor the execution, the support to runtime analysis and the dynamic changes in the workflow execution. Although different tools can be developed to support steering of workflows, we believe real-time provenance support is one of the main features that can fully support it.

IV. Experimental Evaluation

In this section, we show how the algebraic approach may support big data experiments. The workflow execution engine can take advantage of the algebra to compute different execution strategies that improve the efficiency of the workflow. Furthermore, the scientist can query the experiment and fine-tune the execution to produce better results. During the study, we consider the four different execution strategies presented in Section III.A (static-blocking, static-pipeline, dynamic-blocking and dynamic-pipeline). The executions of BuzzFlow were performed on an SGI Altix ICE 8200 cluster, with a total of 32 nodes (and 32 × 8 cores) of 2.66GHz Intel Xeon 5355 8-core processors. The workflow was executed using Chiron[**], a parallel workflow engine that implements our algebra. Each execution of BuzzFlow generated 70,811 parallel activations with more than 600,000 tuples in the relations of the workflow for the DBLP bibliography.

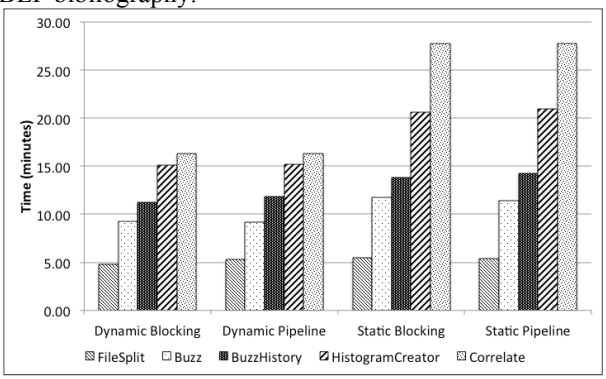

Figure 3. Execution time for the four different strategies.

BuzzFlow was executed using the four available execution strategies. We show the execution time of the most compute-intensive activities of the dataflow in Figure 3. We can see that the execution strategy choice impacts significantly the dataflow performance. The worst strategy for the workflow (static-pipeline) takes 79.7 minutes to complete while the best strategy (dynamic-blocking) takes 56.6

[**] http://chironengine.sf.net

minutes - a performance difference of up to 29.1%. The dynamic dispatching strategies outperform the static ones. Since most activations of the dataflow are short-term and the experiments are executed in a cluster with private and high-speed InfiniBand network connection, dispatching individual activations as the compute nodes request them tends to be more efficient because it minimizes the overall idle time of available resources. The best choice for the execution strategy may vary depending on the workflow and the execution environment. The static approach may be better on environments with higher latency, for example.

The results in Figure 3 do not show significant performance differences between the blocking and pipeline strategies. To provide a better view of the differences between these execution strategies, we applied an algebraic optimization (AT1) in BuzzFlow. AT1 makes Correlate and HistogramCreator activities execute one after the other. Both activities are time-consuming and are associated to the Map operator. Thus, the dataflow engine may benefit from pipelining the activations. We execute AT1 with both dynamic-blocking and dynamic-pipeline strategies as depicted in Figure 4. Although AT1 improves the usage of the pipeline strategy, it increases the cardinality of intermediate results, which affects the performance of the workflow negatively. Alternatively, another algebraic optimization (AT2) places ZipfFilter before HistogramCreator because ZipfFilter decreases the cardinality of data in approximately 90%. Thus, a better alternative for the workflow is to anticipate the ZipfFilter activity. We execute AT2 only with the dynamic-pipeline strategy. For both AT1 and AT2, we performed manual optimizations. Figure 4 shows the time comparison between the original BuzzFlow and the algebraically transformed versions AT1 and AT2.

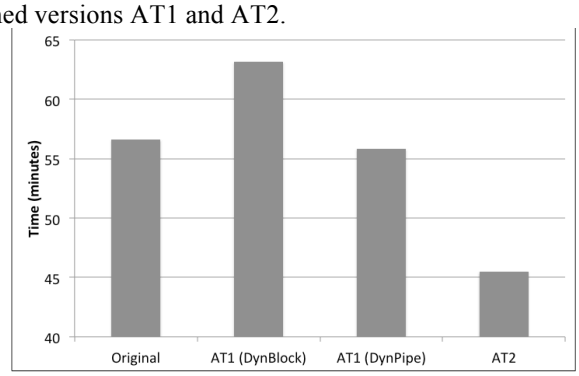

Figure 4. Performance of different BuzzFlow designs

The results show the impact on performance of different ways of modeling the dataflow using the algebraic approach for big data processing. AT2 is a better choice since it reduces the cardinality of intermediate results. The way the activities are chained into the dataflow and the choice of a given execution strategy can impact the execution performance. These optimizations are related to the logic of the dataflow. However, scientists usually need to fine-tune the dataflow to obtain more accurate results while they are experimenting with data. For instance, in the BuzzFlow example, the ZipfFilter activity can be adjusted to filter more or less data depending on the requirements of the experiment. The filter adjustment, in this case, is related to the best trade-off between precision and recall that scientists need. ZipfFilter is originally configured to filter out around 90% of the data. Scientists may analyse preliminary results at runtime and decide to change the parameter of ZipfFilter to increase the filtering level to 95%, for example. Increasing the filter speeds up the execution of the remainder of the workflow because it reduces the cardinality of intermediate data for the remaining activities.

In order to change a parameter of the workflow, scientists typically abort current workflow execution and re-execute the workflow with the new configuration (with the new parameter value). This *manual* approach is very simple, but it is hard to maintain and not very efficient because the activities of the workflow that are not affected by the change do not need to be re-executed. We believe scientists should be able steer the workflow execution by making changes while the workflow is running. Thus, the workflow engine should be sensitive to the parameter values and dynamically adjust the execution accordingly. This *steering* approach may be more efficient as shown in Figure 5. In the figure, we show the comparison between the manual and steering approaches applied to both the original BuzzFlow and the enhanced AT2 optimization. The manual approach includes: the time to run the workflow until the activity *ZipfFilter* produces its results; the overhead between the partial data analysis and the workflow restart; and the time to run the workflow again with the new parameter configured for the filter. Alternatively, the steering approach includes: the time to run the workflow until the activity *ZipfFilter* produces its results; the overhead time to change the parameter at runtime; and the time to run the remainder of the workflow with the new filter configuration.

The steering approach is expected to be more efficient than the manual approach since it saves the re-execution of several activities of the workflow. In this case, we save up to 39.8% of execution time by steering the original BuzzFlow. The AT2 transformation takes less advantage of user steering (30.6% of improvement) because the *ZipfFilter* activity is anticipated, thus fewer activities are re-executed in the manual approach. Additionally, more activities are executed after the changes.

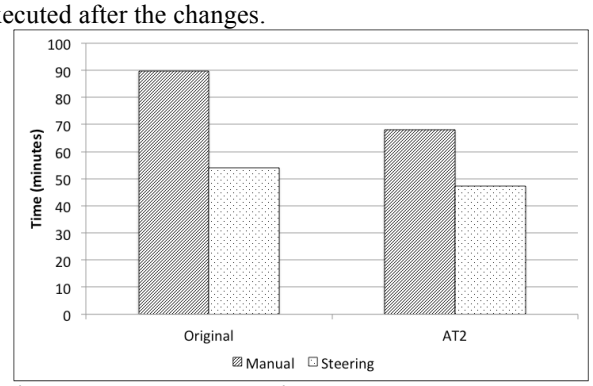

Figure 5. Manual re-execution approach compared to user steering.

This experimental evaluation highlights the flexibility of the algebraic approach to manage dataflows. Using algebraic optimizations and choosing the best execution strategy for the workflow improves the execution performance signifi-

cantly. Furthermore, considering the experimental process where scientists refine a dataflow to produce more accurate results, the real-time provenance leverages user steering of the dataflow. Analyzing partial data and performing changes at runtime can save execution time, thus resulting in more efficient experiments.

V. CONCLUSION

In this paper, we presented an algebraic approach for big data processing. It extends concepts from dataflow languages and parallel databases to model workflows that can be executed in parallel. The algebra provides data uniformity and algebraic operations that are associated to activities, which can invoke general-purpose programs, including legacy code and third-party software. The workflow algebra offers different execution strategies for the workflow and provides opportunities for workflow optimization using algebraic transformations. Since the workflow algebra is an extension of relational algebra, it is straightforward to use the provenance database to store the workflow relations, so that scientists can query provenance and experiment data at runtime. We illustrated how a general dataflow (called BuzzFlow) can be modeled in the algebra for big data processing. We also discuss some real-time queries that leverage the steering of workflows by scientists during the experiment execution.

We present an experimental evaluation using real datasets collected from DBLP and the execution of BuzzFlow with Chiron, a workflow engine that supports our algebra. The choice of different execution strategies influences the dataflow performance by up to 29.1%. We also showed the impact on performance of different ways of modeling the dataflow. The performance difference between the original dataflow and other equivalent designs generated through algebraic optimizations is up to 19.6%. These performance differences open opportunities for optimizers that, based on a cost model and provenance statistics, may choose the best design and execution strategy for a given input dataflow. On a user steering scenario, where scientists need to fine-tune the experiment to obtain more accurate results, our approach yields up to 39.1% of time saving.

By combining well-funded concepts from database technologies with new technologies for data processing, we believe the algebraic approach is a promising approach to handle big data analysis. It has the possibility to optimize the dataflow through algebraic optimizations, the choice of the most appropriate execution model for each fragment of the dataflow and real-time provenance support. The proposed algebra may be combined with current big data processing technologies, to work, for example, with Pig Latin. Other approaches such as [6–8] can also be combined to the algebraic approach to improve data partitioning and replication.

VI. REFERENCES

[1] C. Lynch, 2008, Big data: How do your data grow?, *Nature*, v. 455, n. 7209 (print Setembro.), p. 28–29.

[2] J. Dean and S. Ghemawat, 2008, MapReduce: simplified data processing on large clusters, *Communications of the ACM*, v. 51, n. 1, p. 107–113.

[3] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, 2008, Pig latin: a not-so-foreign language for data processing, In: *Proceedings of SIGMOD*, p. 1099–1110

[4] C. Olston, B. Reed, A. Silberstein, and U. Srivastava, 2008, Automatic optimization of parallel dataflow programs, In: *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, p. 267–273

[5] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, 2012, SkewTune in action: mitigating skew in MapReduce applications, *Proceedings of the VLDB Endowment*, v. 5, n. 12 (Agosto.), p. 1934–1937.

[6] J. Dittrich, J.-A. Quiané-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad, 2012, Only aggressive elephants are fast elephants, *Proceedings of the VLDB Endowment*, v. 5, n. 11, p. 1591–1602.

[7] A. Floratou, J.M. Patel, E.J. Shekita, and S. Tata, 2011, Column-oriented storage techniques for MapReduce, *Proceedings of the VLDB Endowment*, v. 4, n. 7 (Abril.), p. 419–429.

[8] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F.B. Cetin, and S. Babu, 2011, Starfish: A Self-tuning System for Big Data Analytics, In: *Proceedings of the 5th Conference on Innovative Data Systems Research*, p. 261–272

[9] M.T. Özsu and P. Valduriez, 2011, *Principles of Distributed Database Systems*. 3 ed. New York, Springer.

[10] J. Freire, D. Koop, E. Santos, and C.T. Silva, 2008, Provenance for Computational Tasks: A Survey, *Computing in Science and Engineering, v.10*, n. 3, p. 11–21.

[11] Y. Amsterdamer, S.B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen, 2011, Putting lipstick on pig: enabling database-style workflow provenance, *Proceedings of the VLDB Endowment*, v. 5, n. 4 (Dezembro.), p. 346–357.

[12] H. Park, R. Ikeda, and J. Widom, 2011, RAMP: A System for Capturing and Tracing Provenance in MapReduce Workflows, *Proceedings of the VLDB Endowment*, v. 4, n. 12

[13] D. Crawl, J. Wang, and I. Altintas, 2011, Provenance for MapReduce-based data-intensive workflows, In: *Proceedings of the 6th workshop on Workflows in support of large-scale science*, p. 21–30

[14] R. Ikeda, H. Park, and J. Widom, 2011, Provenance for Generalized Map and Reduce Workflows, In: *Proceedings of the 5th Conference on Innovative Data Systems Research*

[15] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, et al., 2007, Examining the Challenges of Scientific Workflows, *Computer*, v. 40, n. 12, p. 24–32.

[16] J. Dias, E. Ogasawara, D. Oliveira, F. Porto, A. Coutinho, and M. Mattoso, 2011, Supporting Dynamic Parameter Sweep in Adaptive and User-Steered Workflow, In: *6th Workshop on Workflows in Support of Large-Scale Science*, p. 31–36

[17] F. Hueske, M. Peters, M.J. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas, 2012, Opening the black boxes in data flow optimization, *Proceedings of the VLDB Endowment*, v. 5, n. 11, p. 1256–1267.

[18] C. Chambers, A. Raniwala, F. Perry, S. Adams, R.R. Henry, R. Bradshaw, and N. Weizenbaum, 2010, FlumeJava: easy, efficient data-parallel pipelines, In: *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*, p. 363–375

[19] E. Ogasawara, J. Dias, D. Oliveira, F. Porto, P. Valduriez, and M. Mattoso, 2011, An Algebraic Approach for Data-Centric Scientific Workflows, *Proc. of VLDB Endowment*, v. 4, n. 12, p. 1328–1339.

[20] K.A.C.S. Ocaña, D. Oliveira, J. Dias, E. Ogasawara, and M. Mattoso, 2012, Discovering Drug Targets for Neglected Diseases Using a Pharmacophylogenomic Cloud Workflow, In: *Proceedings of the IEEE 8th International Conference on e-Science*

[21] D. Oliveira, K. Ocaña, F. Baião, and M. Mattoso, 2012, A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds, *Journal of Grid Computing*, v. 10, n. 3, p. 521–552.

[22] G. Guerra, F. Rochinha, R. Elias, D. Oliveira, E. Ogasawara, J. Dias, M. Mattoso, and A.L.G.A. Coutinho, 2012, Uncertainty Quantification in Computational Predictive Models for Fluid Dynamics Using Workflow Management Engine, *International Journal for Uncertainty Quantification*, v. 2, n. 1, p. 53–71.

[23] J.M. Hellerstein and M. Stonebraker, 1993, Predicate migration: optimizing queries with expensive predicates, In: *ACM SIGMOD Record*, p. 267–276