Scalable Provenance Storage and Querying Using Pig Latin for Big Data Workflows

Fahima Bhuyan, Dr. Shiyong Lu, Dong Ruan
Department of Computer Science
Wayne State University
Detroit, MI
fahima.amin@wayne.edu, shiyong@wayne.edu, du7600@wayne.edu

Dr. Jia Zhang
Electrical and Computer Engineering
Carnegie Mellon University Silicon Valley
Mountain View, CA
jia.zhang@sv.cmu.edu

***Abstract*—Provenance refers to the information about the derivation history of a data product. It is important for evaluating the quality and trustworthiness of a data product and ensuring the reproducibility of scientific discoveries. Much research has been done on storing and querying scientific workflow provenance - provenance that is produced in the execution of data-centric scientific workflows. To address the challenges of big data in increasing volume, velocity and variety, a new generation of scientific workflows, called big data workflows are under active research. As both data and workflows increase in their scale, the scale of provenance naturally increases, calling for a new scalable storage and querying infrastructure. This paper leverages Pig Latin, a high-level platform for creating programs that run on Apache Hadoop, and OPQL, a graph-level provenance query language, to build a scalable provenance storage and querying system for big data workflows. Our main contributions are: i) we propose algorithms to translate OPQL constructs to equivalent Pig Latin programs; ii) we extend OPQL, to support the W3C PROV-DM standard provenance model; iii) we develop and evaluate our system on provenance datasets from the UTPB benchmark; and (iv) we create some visual OPQL constructs in the DATAVIEW big data workflow system to facilitate the easy creation of complex OPQL queries in a visual workflow style. Our preliminary experimental study shows the feasibility of our framework for big-data-scale provenance storage and querying.**

***Keywords*-Provenance; Pig; Big data; Hadoop Distributed File System; Query language;**

I. Introduction

Provenance refers to the information about the derivation history of a data product [1], [2]. It is important for evaluating the quality and trustworthiness of a data product and ensuring the reproducibility of scientific discoveries [3], [4]. Much research has been done on storing and querying scientific workflow provenance - provenance that is produced in the execution of data-centric scientific workflows [2], [5]. Since scientific workflow provenance are essentially directed acyclic graphs, a leading trend of querying models is the graph-based querying models, represented by two provenance graph query languages: OPQL [6] and QLP [5]. While QLP provides query constructs for querying both structure and lineage information in provenance graphs, OPQL, in addition, supports the Open Provenance Model [7], a community-driven data model, which captures main aspects of the workflow provenance and does not enforce a particular physical representation of the provenance data.

Recently, big data workflows have emerged as the next generation of data-centric workflow technologies to address the five "V" challenges of big data: volume, variety, velocity, veracity, and value [8]. While its precedent, scientific workflows, focus on dataflow and automation management [9], big data workflows focus on large-scale data processing and analytics with a "scale-out" architecture and a "moving-computation-to-data" processing paradigm [10]. As both data and workflow increases in their scale, the scale of provenance naturally increases, calling for a new scalable storage and querying infrastructure.

To this end, we propose to leverage Pig Latin [11], a high-level platform for creating programs that run on Apache Hadoop, and OPQL [6], the most popular graph-level provenance query language, to build a scalable provenance storage and querying system. Pig is a platform for analyzing large data sets on top of Hadoop with a rich, multi-valued, and nested data model. Pig's language, Pig Latin, is a comprehensive imperative query language that let users express data transformation such as filtering datasets, merging them and applying functions to groups of records or records. It is a simple data flow language, yet a fast iterative language with an efficient MapReduce compilation engine. Pig Latin gives a higher level of abstraction from the MapReduce procedural model by providing join, filter like relational style operators, which are not feasible in MapReduce out of the box. OPQL is a graph-level provenance query language that include six types of graph patterns. It is based on a rigid provenance graph algebra and clear syntax and semantics. As OPQL queries are not tightly coupled to the underlying provenance storage strategies, an OPQL user does not need to be aware of the underlying schema design. Moreover, OPQL is technology-independent, and therefore can be integrated with any big data workflow system. To our best knowledge, this is the first effort to ensure both scalability (leveraging a scalable platform) and usability (leveraging a graph-level provenance query language) of provenance querying in the area of big data workflows.

Our main contributions are four-fold: i) we propose algorithms to translate OPQL constructs to equivalent Pig Latin programs; ii) we extend OPQL, a graph-level provenance query language, to support the W3C PROV-DM standard provenance model, the current de facto standard provenance model, which extended OPM with additional features to capture provenance of provenance and accommodate the Web; iii) we develop and evaluate our system on provenance datasets from the UTPB benchmark; and (iv) we have created some visual OPQL constructs in the DATAVIEW big data workflow system to facilitate the easy creation of complex OPQL queries in a visual workflow style. Our preliminary experimental study shows the feasibility of our framework for big-data-scale provenance storage and querying.

The rest of the paper is organized as follows: In Section I, we introduce the preliminary framework of provenance models, OPQL query language, big data provenance, and the Apache Pig/Pig Latin data model. In section II, we propose a new framework called $OPQL^{Pig}$ to translate OPQL constructs to equivalent Pig Latin programs. Section III proposes our extension of OPQL to support the PROV-DM provenance model. The algorithm for translating OPQL to Pig Latin is presented in Section IV. Finally, in Section V, we discuss about data preparation and experimental section with some visual OPQL constructs in the DATAVIEW big data workflow system to facilitate the easy creation of complex OPQL queries in a visual workflow style.

II. Motivation

A. Provenance Models

1) OPM-Model: The Open Provenance model (OPM) is the first model of provenance that supports digital representation of provenance. The core set of rules in OPM identifies the valid inferences by directed acyclic graph to express such dependencies. OPM graph has three types of nodes, (i.e, Artifact, Process and Agent) and five types of relations between nodes (i.e. Used, WasGeneratedBy, WasControlledBy, WasTriggeredBy, WasDerivedFrom) for representing causal dependencies. In recent years, another provenance data model was introduced in 2013, named PROV-DM. Fig. 1 represents both provenance models in graphical format.

2) PROV-DM Model: The PROV set of specification is designed to promote PROV provenance model for more generic and domain-agnostic, while remaining easily extensible and exploited for modeling specific domains. This provenance model offers interoperability across diverse provenance management systems and accommodates the provenance of data generated from a diverse data sources. PROV-DM is the core of PROV set of specification which is essentially a relational model that captures intrinsic elements of provenance and tailored to accommodate the requirement of specific application domains [12]. There are other expressions of PROV family like PROV-O for OWL ontology,

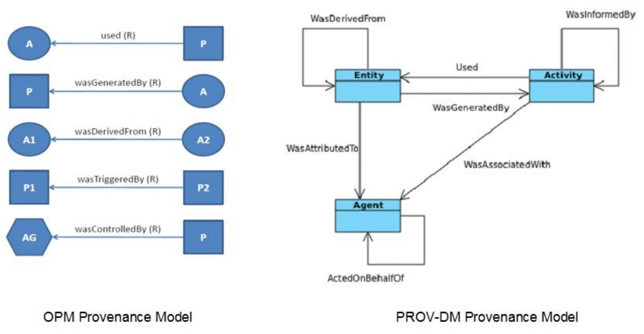


Figure 1: Provenance Models

PROV-N for human readable relational syntax and PROV-XML for XML encoding. Additionally, for accessing provenance document PROV-AQ and strong notion of valid provenance defined by a system of constraint PROVCONSTR are also provided. PROV provenance model has flexibility when it deals with attributes. Most of the provenance statement has annotated with optional attributes. This provenance model has a mechanism for asserting provenance of provenance specified as 'bundles'. To avoid to make the model unnecessary complex, PROV-DM does not model uncertainty. Even though this new model also has three types of nodes, but their names are different. "Artifact" become "Entity", "Process" become "Activity" and "Agent" remains same. Also in PROV-DM, two additional relations are introduced, "ActedOnBehalfOf()" and "WasAttributedTo()" and two relations are renamed "WasTriggeredBy()" as "WasInformedBy()" and "WasControlledBy()" as "WasAssociatedWith()".

B. Provenance Query Language: OPQL

OPQL aims to query the provenance data most of the existing query languages such as SQL, SPARQL, XQuery etc uses the provenance storages like RDB, RDF, XML etc. Therefore to prevail the underlying schemas and structures of provenance storage and also the semantics of models, the VIEW workflow system has proposed OPQL query language to work on OPM-level provenance [13], [6], [14], [15]. In the direction of provenance lineage queries, very few work has been done. OPQL is one of those where queries for tracking ancestor nodes without writing recursive statement or needed any recursive functionality for writing recursive queries. OPQL queries are formulated based on OPM graph. An OPQL query is either a basic query or composition of multiple queries via set operators like UNION, INTERSECT or MINUS. In OPQL, the basic queries are defined in 4 different ways:

- Single node construct A, P, AG
- Single-step-edge-forward constructs USD, WGB, WCB, WDF, and WTB

- Single-step-edge-backward constructs $USD\wedge$, $WGB\wedge$, $WCB\wedge$, $WDF\wedge$, and $WTB\wedge$
- Multi-step-edge constructs $USD*$, $WGB*$, $WDF*$, and $WTB*$

C. Apache Pig

Apache Pig is a fairly competitive framework for processing and continuous optimization, enhanced with new features and maintained by Yahoo! Researchers [16]. The advantage of using Pig is that it is independent of Hadoop framework changes and can be benefited by all the optimization techniques offered by Pig developer community. Moreover, as Pig is backward compatible, further developments or optimizations will not effect any single line of code. During runtime, the resulting Pig Latin script automatically maps into a sequence of MapReduce iterations. Hence, complicated deployment, configuration or installation do not require. We have used few Pig Latin operator in our translation. We explained each instructions of pig performed in our case in later section. The more detail description will be in Pig Latin Manual in [17].

III. $OPQL^{Pig}$ Querying Framework

$OPQL^{Pig}$ is an integrated system of Storage and Query engines and its underlying MapReduce and Apache Hadoop framework. Though there are several concurrent projects like DryadLINQ, Hive, Jaql, Sawzall, Scope etc., blending Pig with its underlying Hadoop execution engine shows an impressive benefits of scalability and fault tolerance. The system overview of $OPQL^{Pig}$ is to take datasets from OPQL client, processing the data in the storage engine, translate OPQL query into Pig Latin program, parse and compile into Pig and one or more MapReduce Jobs and execute those jobs in Hadoop cluster. We will discuss each of the parts of the system in this section.

A. Storage Engine

1) Data Modeling and Representation: The provenance can be captured in any form of modeling and representation like PROV-DM, PROV-O and RDF/XML, RDF/Turtle or RDF/NTriple. For our $OPQL^{Pig}$ system we have used UTPB benchmark for capturing provenance data from templates presented there.

2) Data Loading: After capturing provenance data, we have data loading phase for creating the dataset in our feasible format and store that in Hadoop Distributed File system for querying. For storing the dataset the feasible format we have used is: source_node, source_node_type, destination_node, destination_node_type, relation_construct.

B. Query Engine

Query engine has three major functional units with three unique responsibilities: Query Translation, Apache Pig and Query execution.

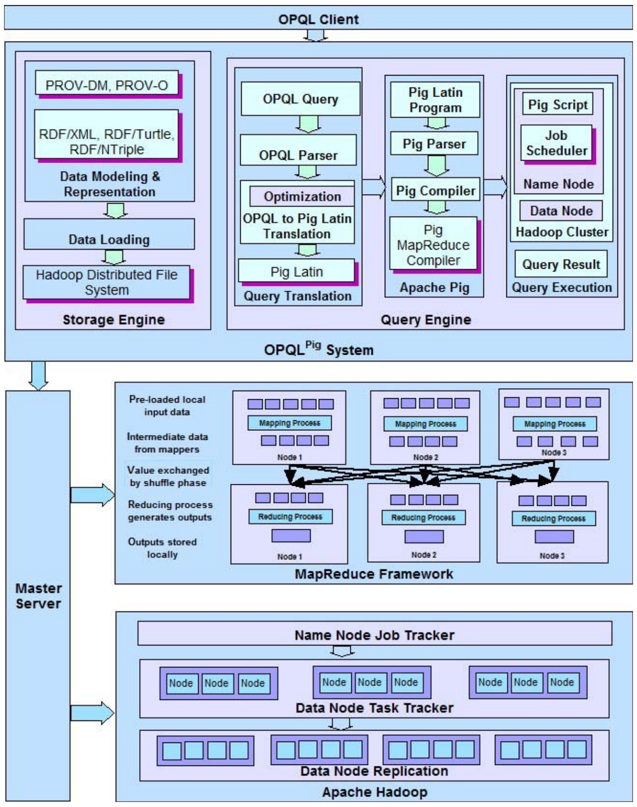


Figure 2: $OPQL^{Pig}$ Architecture

1) Query Translation: First all OPQL queries are passed through OPQL parser to verify their syntactic correctness. Each of the construct of OPQL is transformed into its corresponding Pig program through a dedicated shell script for each relation in PROV-DM model. The description of each translation from PROV-DM relations to its correspondence pig latin program will be explained in the next section.

2) Apache Pig: After translating OPQL query into pig latin program which consists of a sequence of instructions, each instruction performs a single data transformation. During the phase of Pig parser schema inference, type checking and all referenced variables are defined. The output of the parser is arranged as a Directed Acyclic Graph which is a logical plan of one-to-one correspondence between pig latin statements and logical operators [18].

3) Query Execution: The MapReduce jobs in DAG is topologically sorted and submitted to the Job Scheduler's Name node. Finally jobs are executed in that order inside Hadoop cluster's data node. $OPQL^{Pig}$ Querying Framework is represented in Fig. 2.

IV. Extending OPQL to support PROV-DM

A. Provenance Constructs

We have modified the constructs based on the PROV-DM standard for all Single-node constructs, Single-step construct (edge-forward and edge-backward) and Multi-step-edge construct.

1) Single-node Construct: We have formulated our single-node construct in the following formulation:

$$E(X_e) = \{e_n \mid e_n \in X_e\}$$
$$A(X_a) = \{a_n \mid a_n \in X_a\}$$
$$AG(X_{ag}) = \{ag_n \mid ag_n \in X_{ag}\}$$

2) Single-step Construct: In a Single-step construct, there are two kinds of constructs; Single-node constructs (i.e., Entity, Activity and Agent) and Single-step construct. For Single-step-edge-forward and Single-step-edgebackward, we formulate all the relations in both directions.

$$USD(X_a) = \{e_n \mid a_n \in X_a \ and \ (a_n, e_n) \in Edge_u\}$$
$$WGB(X_e) = \{a_n \mid e_n \in X_e \ and \ (e_n, a_n) \in Edge_g\}$$
$$WCB(X_a) = \{ag_n \mid a_n \in X_a \ and \ (a_n, ag_n) \in Edge_c\}$$
$$WDF(X_e) = \{e_{n2} \mid e_{n_1} \in X_e \ and \ (e_{n_1}, e_{n_2}) \in Edge_d\}$$
$$WTB(X_a) = \{a_{n2} \mid a_{n_1} \in X_a \ and \ (a_{n_1}, a_{n_2}) \in Edge_t\}$$
$$USD^\wedge(X_e) = \{a_n \mid e_n \in X_e \ and \ (a_n, e_n) \in Edge_u\}$$
$$WGB^\wedge(X_a) = \{e_n \mid a_n \in X_a \ and \ (e_n, a_n) \in Edge_g\}$$
$$WCB^\wedge(X_{ag}) = \{a_n \mid ag_n \in X_{ag} \ and \ (a_n, ag_n) \in Edge_c\}$$
$$WDF^\wedge(X_e) = \{e_{n_1} \mid e_{n_2} \in X_e \ and \ (e_{n_1}, e_{n_2}) \in Edge_d\}$$
$$WTB^\wedge(X_a) = \{a_{n_1} \mid a_{n_2} \in X_a \ and \ (a_{n_1}, a_{n_2}) \in Edge_t\}$$

3) Multi-step Construct: Multi-step Constructs are a little bit more complicated, which uses both direction of single-step constructs in a repetitive way.

$$WDF^*(X_e) = \{e_n \mid \bigcup_{e_n \in WDF(X_e)} WDF^*(e_n) \ \cup \ WDF(X_e)\}$$
$$WTB^*(X_p) = \{a_n \mid \bigcup_{a_n \in WTB(X_a)} WTB^*(a_n) \ \cup \ WTB(X_a)\}$$
$$WGB^*(X_a) = \{a_n \mid \bigcup_{a_n \in WGB(X_e)} WTB^*(a_n) \ \cup \ WGB(X_e)\}$$
$$USD^*(X_p) = \{e_n \mid \bigcup_{e_n \in USD(X_a)} WDF^*(e_n) \ \cup \ USD(X_a)\}$$

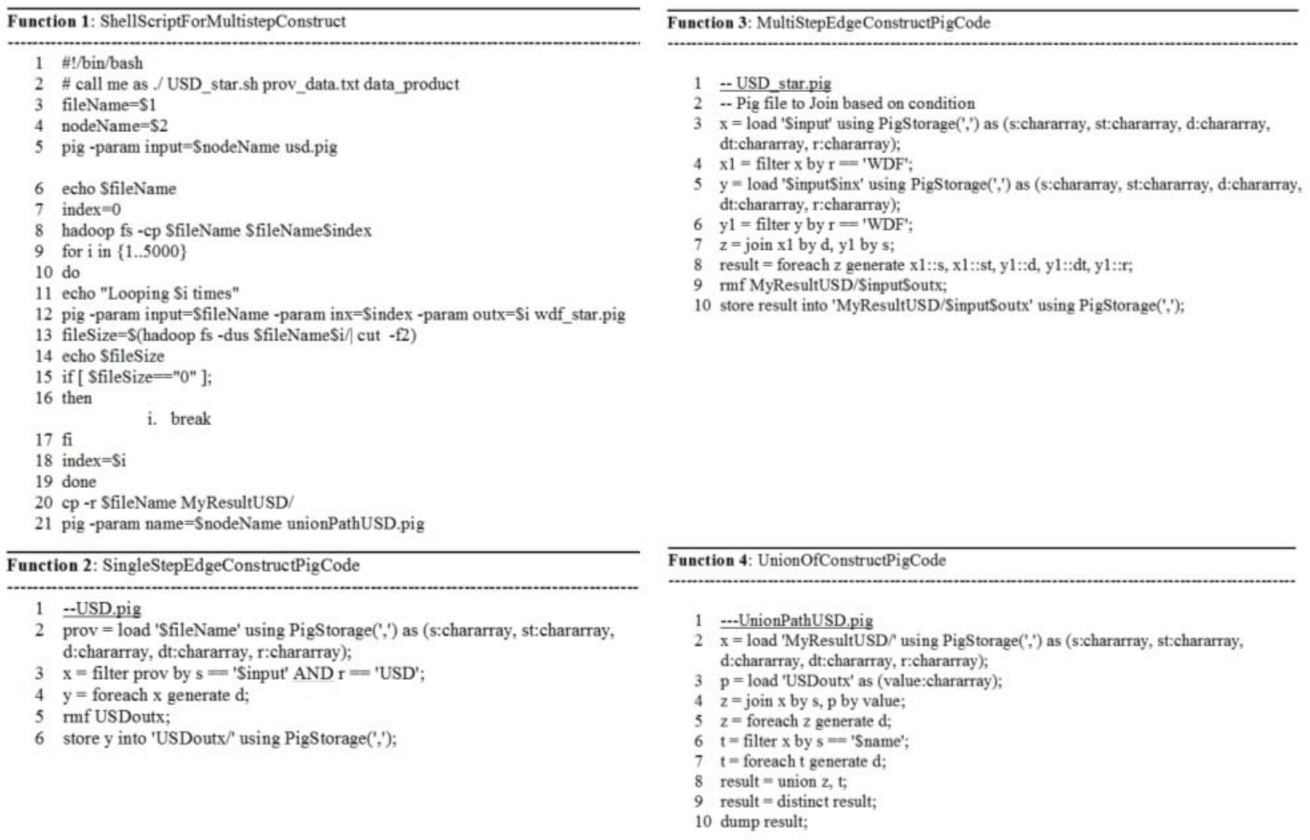
V. Translating $OPQL$ to Pig Latin

In this section we have explained the translation of $OPQL$ to $OPQL^{Pig}$. We have translated each of the construct to Pig latin script. The implementation of each of the constructs is handled through separate shell scripts, because Pig Latin does not support loop and if condition. As each of the construct is designed to handle inference queries, it needs to traverse the whole provenance graph to provide query answer. To handle such a scenario, shell script is used to run multi-step-edge-constructs by provisioning join operations in one pig file, going through a loop. In addition, the loop break condition is handles by shell script. The join operation will generate a set of records which provide transitive relation and the loop condition will break when there will be no new transitive relations. The loop break condition measures the file size, if there is no new relations, which means file size zero, then loop break will happen. The last step of each construct is to run another pig file which will retrieve all the destination nodes along the path computed by join operations. The pseudo code of translating from $OPQL$ to $OPQL^{Pig}$ for single-step and multi-step constructs is summarized in Fig 3.

In Fig 3 we have given the algorithm just for one example construct USD. Function 1 delineates the entire process. Function 1 essentially acts as the driver program that iteratively invokes the constituent functions until the completion condition is met. It can be noted that function 1 can be written as a driver program in Map-Reduce framework as well and is preferred, we present it as a shell script to intuitively represent the flow. We assume that the input file has the following format: source_label, source_type, destination_label, destination_type, relation_type. We present the algorithm in terms of USD construct, however, the approach is a generic one and can be applied to any other construct type. In line 5, we invoke usd.pig, which essentially selects rows that have the relation USD and have the source that we are interested in making query against. This part of the algorithm essentially persists destinations that are reachable via one hop from the source of interest. Subsequently in line 8, we prepare for the iterative self-join phase by cloning the input file. Line number 9 shows the termination condition for the iteration. Line 12, consists of invoking USD_star.pig with variable inputs. In each phase of the iteration, we join the original input with the recently computed output, as shown by the parameters of the pig file. The output of one phase of USD_star.pig is used as one of the inputs of the next phase of USD_star.pig. Finally in UnionPathUSD.pig, we combine outputs from usd.pig and USD_star.pig and project distinct destinations that are reachable and persist them on HDFS for subsequent usage in the workflow.

VI. Experiments

A collection of experiments were conducted on a machine with Intel core $i7-3612QM$ CPU $@2.10GHz$ x 8 processor and 7.7 GB memory running on Ubuntu 12.10 (quantal) 64 bit. The experiments were designed on Apache Pig with version $0.8.1-cdh3u6$ and the Hadoop framework with version $hadoop0.20.2-cdh3u6$. Even though there are 27 different provenance templates representing provenance capture from three different workflows, we have chosen one particular presentation capturing data from one specific workflow. We have captured provenance using UTPB template. UTPB was selected as the benchmark template because it automatically generates datasets with varying sizes.



Function 1: ShellScriptForMultistepConstruct

```
#!/bin/bash
# call me as ./ USD_star.sh prov_data.txt data_product
fileName=$1
nodeName=$2
pig -param input=$nodeName usd.pig

echo $fileName
index=0
hadoop fs -cp $fileName $fileName$index
for i in {1..5000}
do
echo "Looping $i times"
pig -param input=$fileName -param inx=$index -param outx=$i wdf_star.pig
fileSize=$(hadoop fs -dus $fileName$i/| cut  -f2)
echo $fileSize
if [ $fileSize=="0" ];
then
              i.  break
fi
index=$i
done
cp -r $fileName MyResultUSD/
pig -param name=$nodeName unionPathUSD.pig
```

Function 2: SingleStepEdgeConstructPigCode

```
--USD.pig
prov = load '$fileName' using PigStorage(',') as (s:chararray, st:chararray,
d:chararray, dt:chararray, r:chararray);
x = filter prov by s == '$input' AND r == 'USD';
y = foreach x generate d;
rmf USDoutx;
store y into 'USDoutx/' using PigStorage(',');
```

Function 3: MultiStepEdgeConstructPigCode

```
-- USD_star.pig
-- Pig file to Join based on condition
x = load '$input' using PigStorage(',') as (s:chararray, st:chararray, d:chararray,
dt:chararray, r:chararray);
x1 = filter x by r == 'WDF';
y = load '$input$inx' using PigStorage(',') as (s:chararray, st:chararray, d:chararray,
dt:chararray, r:chararray);
y1 = filter y by r == 'WDF';
z = join x1 by d, y1 by s;
result = foreach z generate x1::s, x1::st, y1::d, y1::dt, y1::r;
rmf MyResultUSD/$input$outx;
store result into 'MyResultUSD/$input$outx' using PigStorage(',');
```

Function 4: UnionOfConstructPigCode

```
---UnionPathUSD.pig
x = load 'MyResultUSD/' using PigStorage(',') as (s:chararray, st:chararray,
d:chararray, dt:chararray, r:chararray);
p = load 'USDoutx' as (value:chararray);
z = join x by s, p by value;
z = foreach z generate d;
t = filter x by s == '$name';
t = foreach t generate d;
result = union z, t;
result = distinct result;
dump result;
```

Figure 3: Algorithms for multi-step edge

A. Data Preparation with benchmark

The original manually created template of PROV-DM contain around 66 triples. It always makes at least 3 copies of the original template to reach around 200 triples, so it is easy to get 200, 400 .... $1,000$, $10,000$ etc triples depending on the needs. However, it is only the template generator. The data generator takes the "labels" statements and can change the values to generate different kinds of data, either fixed or randomized. The template generator adds a line to connect templates, $dataset\_n$ was derived from $results\_(n-1)$, but this union is very particular to the original template.

There are three main components based on data preparation with the benchmark:

- Original template: This is a provenance graph for one database experiment. It was created manually [19].
- Template generator: It takes the original template and creates a bigger template automatically. The bigger template is the result of cloning the original template multiple times and connecting clones together into a single graph. Template generator allows one to choose any times one would clone her original template and how one would connect the clones, i.e.,sequentially or grid-shape. The file output.prov is generated by cloning the original template three times. Connections between clones are:
  utpb:$dataset_1$ prov:wasDerivedFrom utpb:$result_0$. and
  utpb:$dataset_2$ prov:wasDerivedFrom utpb:$result_1$.
- Data generator: It takes the original or generated template and generates as many disconnected clones or template instances of the template as specified. This time, each clone represents a provenance graph for one workflow execution. Data generator ensures that there are no ID conflicts between template instances.

During our experimental study, we focused on the performance and functionality of $OPQL^{Pig}$. The experimental data we have gathered were based on the size of big data that $OPQL^{Pig}$ can query and provide the query results. After data preparation, we started with 10 million triples and eventually going to 15, 20, 30, 40 and 50 million triples for generating big data, therefore testing the feasibility of processing queries by $OPQL^{Pig}$. Fig. 4 shows the correlation between the size of triples correspond to big data size.

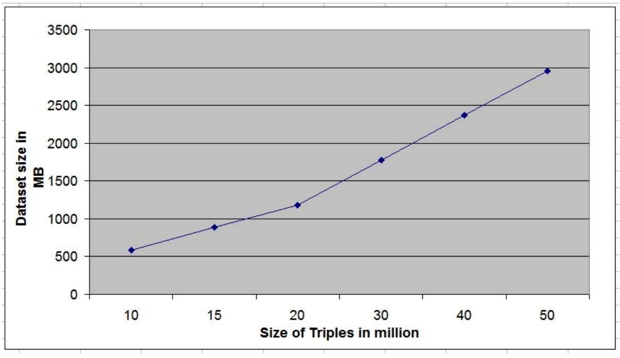


Figure 4: Dataset size vs triple number: Showing Big Data Scenario

B. Building Construct in DATAVIEW

We used DATAVIEW, a computational thinking online service for experimenting our queries. For each provenance construct in PROV-DM model, we created one primitive, also known as built-in workflow in DATAVIEW for querying big data. Fig. 5 shows DATAVIEW system with an executable workflow.

Fig. 5 shows one sample built-in construct USD in DATAVIEW. A 2 input data product is shown. One is our big data file and other one is text data file which gives the query input. The result will be saved in an output file. For giving user more flexibility the current DATAVIEW has integrated Dropbox feature, so that users can provide any big data file and a simple query in the text file and drag-drop the construct and finally can get the query result in output which also can be accessible in a Dropbox folder. In this way, end users do not have to deal with the underlying complexity and can easily obtain query results.

VII. Related Work

Provenance problems become prohibitive and hard to solve when applied to big data repositories. There are many alleys of research challenges and open issues in big data provenance research. Several relevant and advanced concepts and challenges in big data provenance research arise, like accessing big data, analyzing big data, scalability issues, information sharing, query optimization issues, data modeling support for provenance, flexible provenance query tools, etc [20].

Reduce and Map provenance(RAMP) [21], [22] propose a wrapper-based method as an extension of Hadoop by deploying on top of Hadoop yet resulting transparent to it. Another extension of Hadoop for implementing provenance detection in MapReduce jobs is called Hadoop-Prov [20], while the system minimizes the overheads introduced by computing provenance by providing flexible tools for querying, in big data provenance graph. There is a hybrid big data provenance system named Pig Lipstick [23], which combines the management of fine-grained dependencies, with the management of workflow-grained dependencies [20], [23]. Fine-grained dependencies are typical of database-oriented provenance systems and workflow-grained dependencies are typical of workflow-oriented provenance systems. Another type of big data provenance supports the functionalities of layer-based architecture by focusing on provenance collection, querying and visualization of provenance in the context of specialized scientific applications [24].

In cloud environment, CloudProv is a proposed framework for integrating, modeling and monitoring data provenance. It continuously acquires and monitors all the collected provenance information for real-time applications. Gravic et al. [25] propose a big data provenance framework based on fine-grained provenance through several operator instrumentation of a query. Managing fine-grained provenance in Data Stream Management Systems (DSMS) remains a hard problem due to the need of supporting flexible analysis tools over the so-computed provenance, such as revision processing or query debugging. Oruta is developed as a privacy-preserving public auditing mechanism in untrusted Cloud environments [26], aiming to support data sharing. Most of the recent works deal with query processing and Pig, focusing on SPARQL to Pig Latin translation [16], [27].

Out of all scientific workflow systems, few of them have adopted PROV extension. Taverna is one of them [15]. DataONE scientific workflow and provenance working group specified D-PROV provenance model [12]. Only few PROV applications that use and extend PROV. UrbanMatch [28] extends the PROV model by using Human computation ontology; and CollabMap [29] records provenance information that logs citizens actions [12]. Kepler records provenance data in a relational database during execution time, whereas VisTrails generates and stores XML provenance data. Taverna implements the Janus Provenance model and generates the graph during workflow execution and stores in a relational database(mySQL), where provenance graph can be queried and exported in OPM [30].

VIII. Conclusions and Future Work

In this paper, we reported the $OPQL^{Pig}$ query language, extending OPQL for large datasets by implementing the translation operation of OPQL to Pig Latin data flow language with elongated features to make the query language scalable, robust, reliable and parallel for working on top of Hadoop Distributed File System. This query language relishes the MapReduce model without reinventing common functionality such as join, filter and so on. Based on the graph pattern, provenance graph algebra and syntax-semantics of single-step-edge-forward and Multi-step-edge constructs, the $OPQL^{Pig}$ translation covers every scenario.

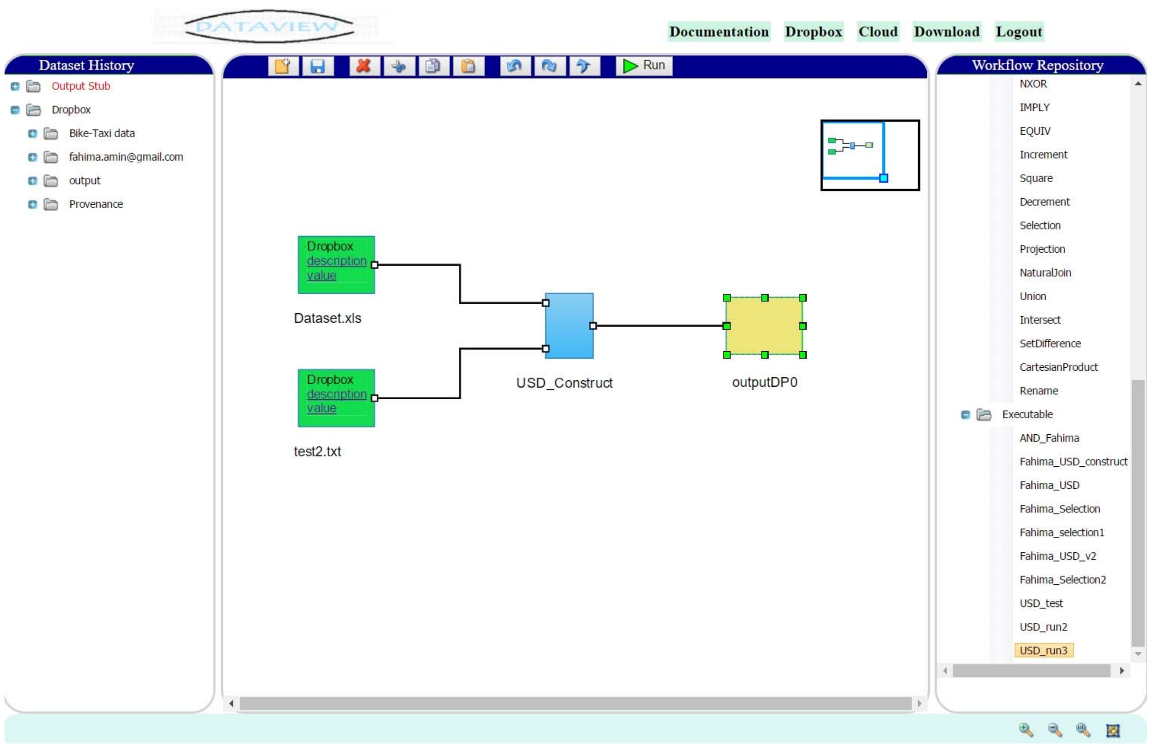


Figure 5: DATAVIEW with executable workflow

In the future, we would like to explore our research in four major directions. First, we plan to extend our query language, so that it can read input from and write output to sources other than HDFS, for example from NoSQL Databases like HBase or Cassandra. Second, we plan to focus on optimization issues in $OPQL^{Pig}$ query language and conduct experimental study based on current system and NoSQL database techniques. Third, we will handle join operation based on Pigs three special join algorithms like replicated, skewed, merge and analyze which works the best in querying provenance data. Finally, we will make $OPQL^{Pig}$ applicable for other queries such as sub-graph isomorphism, pattern matching, and shortest path.

Acknowledgment

This work is supported by National Science Foundation, under grant NSF ACI-1443069. In addition, this material is based upon work supported in part by the National Science Foundation under Grant No. 0910812. I would like to thank Mahdi Ebrahimi and Aravind Mohan for their contribution in DATAVIEW and Dr. Artem Chebotko and his team for helping to use UTPB in PROV-DM format.

References

[1] S. B. Davidson and J. Freire, "Provenance and scientific workflows: challenges and opportunities," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 1345–1350.

[2] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi, "Storing and querying scientific workflow provenance metadata using an rdbms," in *e-Science and Grid Computing, IEEE International Conference on*. IEEE, 2007, pp. 611–618.

[3] O. Hartig and J. Zhao, "Using web data provenance for quality assessment," in *Proceedings of the First International Conference on Semantic Web in Provenance Management-Volume 526*. CEUR-WS. org, 2009, pp. 29–34.

[4] F. S. Chirigati, D. Shasha, and J. Freire, "Reprozip: Using provenance to support computational reproducibility." in *TaPP*, 2013.

[5] M. K. Anand, S. Bowers, and B. Ludäscher, "Techniques for efficiently querying scientific workflow provenance graphs." in *EDBT*, vol. 10, 2010, pp. 287–298.

[6] C. Lim, S. Lu, A. Chebotko, F. Fotouhi, and A. Kashlev, "OPQL: Querying scientific workflow provenance at the graph level," *Data & Knowledge Engineering*, vol. 88, pp. 37–59, 2013.

[7] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers *et al.*, "The open provenance model core specification (v1. 1)," *Future generation computer systems*, vol. 27, no. 6, pp. 743–756, 2011.

[8] A. Kashlev and S. Lu, "A system architecture for running big data workflows in the cloud," in *Services Computing (SCC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 51–58.

[9] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, and J. Hua, "A reference architecture for scientific workflow management systems and the view soa solution," *IEEE Transactions on Services Computing*, vol. 2, no. 1, pp. 79–92, 2009.

[10] M. Ebrahimi, A. Mohan, A. Kashlev, and S. Lu, "Bdap: a big data placement strategy for cloud-based scientific workflows," in *Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on*. IEEE, 2015, pp. 105–114.

[11] 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*, ser. SIGMOD '08, pp. 1099–1110.

[12] P. Missier, K. Belhajjame, and J. Cheney, "The W3C PROV family of specifications for modelling provenance metadata," in *Joint 2013 EDBT/ICDT Conferences, Genoa, Italy, March 18-22, 2013*, pp. 773–776.

[13] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi, "Storing, reasoning, and querying opm-compliant scientific workflow provenance using relational databases," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 781 – 789, 2011.

[14] C. Lim, S. Lu, A. Chebotko, F. Fotouhi, and A. Kashlev, "OPQL: A first opm-level query language for scientific workflow provenance," *2013 IEEE International Conference on Services Computing*, vol. 0, pp. 136–143, 2013.

[15] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a high-level dataflow system on top of map-reduce: The pig experience," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1414–1425, Aug. 2009.

[16] A. Schätzle, M. Przyjaciel-Zablocki, T. Hornung, and G. Lausen, "PigSPARQL: A SPARQL Query Processing Baseline for Big Data," in *International Semantic Web Conference (Posters and Demos)*, 2013, pp. 241–244.

[17] I. Ltkebohle, "Apache. Pig Latin Reference Manual 1 and 2," http://pig-apache.org/docs/, 2010.

[18] A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a highlevel dataflow system on top of mapreduce: The pig experience," *PVLDB*, vol. 2, no. 2, pp. 1414–1425, 2009.

[19] A. Chebotko, E. D. Hoyos, C. Gomez, A. Kashlev, X. Lian, and C. Reilly, "Utpb: A benchmark for scientific workflow provenance storage and querying systems," in *2012 IEEE Eighth World Congress on Services*, 2012, pp. 17–24.

[20] A. Cuzzocrea, "Big data provenance: State-of-the-art analysis and emerging research challenges," in *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT Workshops 2016, Bordeaux, France, March 15, 2016.*, 2016.

[21] R. Ikeda, H. Park, and J. Widom, "Provenance for generalized map and reduce workflows," in *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12*, 2011, pp. 273–283.

[22] H. Park, R. Ikeda, and J. Widom, "RAMP: A system for capturing and tracing provenance in mapreduce workflows," *PVLDB*, vol. 4, no. 12, pp. 1351–1354, 2011.

[23] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen, "Putting lipstick on pig: Enabling database-style workflow provenance," *Proc. VLDB Endow.*, vol. 5, no. 4, pp. 346–357, Dec. 2011.

[24] R. Agrawal, A. Imran, C. Seay, and J. J. Walker, "A layer based architecture for provenance in big data," pp. 1–7, 2014.

[25] B. Glavic, K. S. Esmaili, P. M. Fischer, and N. Tatbul, "Efficient stream provenance via operator instrumentation," *ACM Trans. Internet Techn.*, vol. 14, no. 1, pp. 7:1–7:26, 2014.

[26] B. Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditingfor shared data in the cloud," *IEEE Trans. Cloud Computing*, vol. 2, no. 1, pp. 43–56, 2014.

[27] A. Schätzle, M. Przyjaciel-Zablocki, and G. Lausen, "Pigsparql: Mapping sparql to pig latin," in *Proceedings of the International Workshop on Semantic Web Information Management*, ser. SWIM '11, pp. 4:1–4:8.

[28] I. Celino, S. Contessa, M. Corubolo, D. Dell'Aglio, E. D. Valle, S. Fumeo, and T. Krüger, "Linking smart cities datasets with human computation - the case of urbanmatch," in *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part II*, 2012, pp. 34–49.

[29] M. Ebden, T. D. Huynh, L. Moreau, S. D. Ramchurn, and S. J. Roberts, "Network analysis on provenance graphs from a crowdsourcing application," in *Provenance and Annotation of Data and Processes - 4th International Provenance and Annotation Workshop, IPAW 2012, Santa Barbara, CA, USA, June 19-21*, pp. 168–182.

[30] B. Amann, C. Constantin, C. Caron, and P. Giroux, "Weblab PROV: computing fine-grained provenance links for XML artifacts," in *Joint 2013 EDBT/ICDT Conferences, Genoa, Italy, March 22, Workshop Proceedings*, pp. 298–306.