PROB: A tool for Tracking Provenance and Reproducibility of Big Data Experiments.

Vlad Korolev
University of Maryland, Baltimore County
Baltimore, MD, USA
vkorol1@umbc.edu

Anupam Joshi
University of Maryland, Baltimore County
Baltimore, MD, USA
joshi@umbc.edu

Abstract

Reproducibility of computations and data provenance are very important goals to achieve in order to improve the quality of one's research. Unfortunately, despite some efforts made in the past, it is still very hard to reproduce computational experiments with high degree of certainty. The Big Data phenomenon in recent years makes this goal even harder to achieve. In this work, we propose a tool that aids researchers to improve reproducibility of their experiments through automated keeping of provenance records.

1. Introduction

Reproducibility and provenance tracking is an ongoing concern in the scientific community. Historically computational sciences were particularly bad about disclosing the actual data and code used to obtain published results. To quote David Donoho "Computing results are now being presented in a very loose, "breezy" way – in journal articles, in conferences, and in books. All too often one simply takes computations at face value. [1]."

With the recent increase of collaboration between computational and biological scientific communities, it is especially important to demonstrate provenance of the source and allow other researchers to reproduce experiments described in the publications. In fact according to [2] a major cause of low rate of collaboration between different research groups is attributed to difficulties in reproduction of experiments. Which consequently leads to the poor quality of research. Conversely it has been argued that allowing for easier reproducibility of the experiments leads to the higher quality of research [3].

To address this problem we must strive for higher degree of reproducibility. Which is impossible without thorough tracking of the provenance of original data and all manipulations done to it.

In this work, we propose a tool that aids tracking of the provenance and the ability to achieve reproducibility of experiments that involve big data workflows. The development of this tool is still ongoing, therefore, the code is not available to the public yet.

2. Background

Most of the scientific data analysis workflows have very similar structure. Especially big-data workflows that are based on Map-Reduce model[4]. A typical six step big-data scientific pipeline is shown on Figure 1. At step 1, the data is collected by observing some phenomenon or performing a study of some population of the subjects. For big-data workflows, this is usually done by a third party. In the case of Gene Wide Association Studies (GWAS) in United States, the National Center for Biotechnology Information (NCBI) [5] would be conducting the work.

At step 2, the data is transferred to the researcher who considers it as raw data. For many domains especially the ones related to medicine or biology, where the privacy of subjects is of very high importance, researchers must follow specific protocols before the data is released. Often most of the controlled distribution datasets arrive encrypted and must be decrypted beforehand. Very often the agencies encrypt the dataset using the tools not available for the operating systems that will be used for the actual analysis of the data. This step is usually performed on a different machine.

For big-data workflows, after the data is decrypted and converted it must be loaded onto the distributed computation cluster that will be used for actual analysis. Such transfer happens at step 3.

At step 4, the actual analysis takes place. During this step the researcher uses some software that reads the input data and outputs some results. Besides the input data, the researcher can supply different parameters to the software which will tune the course of the analysis. The result of this step is a derived data that was obtained as a result of the performed analysis. This step is usually performed repeatedly, while the researcher fine-tunes the parameters given to the analytic software. The software itself could be changed either by the researcher or a third party vendor. Also, this step could consist of multiple sub-steps where the output of one sub-step is supplied as an input or a parameter to the subsequent sub-step.

After the researcher have obtained the desired result in step 4, step 5 is performed. In this step, the data is extracted out of a distributed cluster and then visualized and described to produce a publication.

At the final step, the derived data and/or publication is placed into publicly accessible storage where it could serve as raw data for a different research.

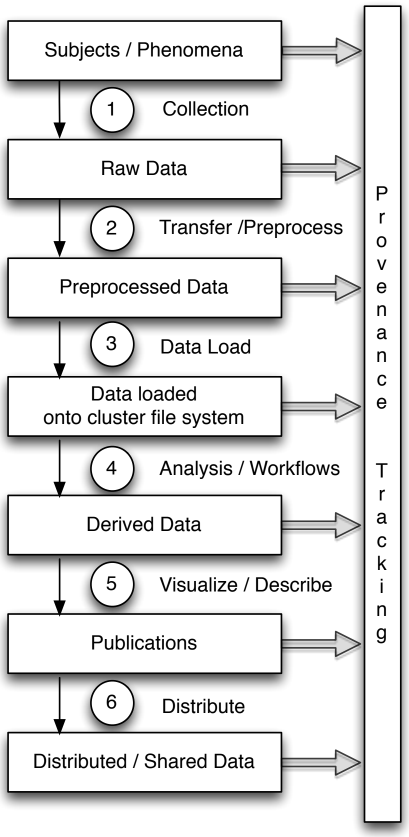

Figure 1. Typical scientific big data workflow.

3. Prior work

Scientific workflow systems are widely used to represent and execute computation experiments. Systems such as Kangaroo[6], VisTrails[7], NiPype[8], Hamake[9] and Taverna[10] has been around for several years and were used in a variety of research projects. These systems allow the researcher to setup and execute analytic workflows. All of these tools represent a workflow as a directed acyclic graph (DAG) where the nodes are either data files, parameters or preprocessing and analysis routines. The edges connect the inputs of these routines to the datasets, parameters or the outputs of another step. Usually these tools are more domain oriented than traditional programming languages, thus making a job of the researcher much easier.

Software developers have been using version control systems (VCS) for considerable time [11]. Among the modern VCS tools there is a growing trend towards distributed version control systems DVCS which do not rely on a central server. The emergence of cloud based web services such as GitHub and BitBucket has led to the explosive popularity of DVCS tools.

Expressing and recording provenance information has been a known problem. There were several attempts to address it such as [12]. Recently W3C has introduced a new standard PROV[13] for tracking provenance of the artifact. A notable tool Git2Prov[14] was developed to convert metadata stored in GIT version control system into the PROV format.

Unfortunately, we believe that none of the tools mentioned above provide functionality that is comprehensive enough to track provenance and provide reproducibility of big-data map-reduce workflows. Such workflows have their own challenges. They must run on Hadoop type distributed clusters. Whereas most of the scientific workflow tools mentioned above, with the exception of Kangaroo and Hamake, are designed to be used on a single workstation. Also most of the tools above do not provide the means to store and reconstruct inputs and exact workflow steps. The tools that do either rely on VCS systems to store the artifacts or in case of VisTrails they assume that entire experiment including all code and data files could be packaged and given to a third party. Such approach would not work with strictly controlled datasets that have strict distribution policies. Even for publicly available datasets such approach is not feasible. For example, GoogleBooks dataset is 2.2TB. Including a copy of such large datasets in every workflow archive will make regular exchange of the workflows infeasible. Storing such large datasets directly in the VCS also does not work. In addition to poor performance and inefficiency of these tools with very large files, most of these tools limit the maximum object size well below the file size found such datasets. Moreover, a typical map-reduce file systems such as HDFS, do not implement full POSIX call interface, so it is not possible to use standard VCS with such file-system.

Recently a new tool called Git-Annex[15] was developed to address some problems of storing large files in git repositories and allow for controlled transfers of restricted datasets. This tool uses a storage format similar to git, but it keeps the large object store separately from the regular git-object store. The storage format of git-annex has been modified to allow handling of very large files on the order of multiple terabytes. However the git-annex tools still maintains full metadata related to the large files in the main git repository so git still can record all manipulations done to these large files as well as maintain the integrity of the repository through the use of checksums. This is achieved by git-annex checking-in the links to the hashes of the large files into the git repository. Such scheme also allows research groups at different organizations to share the code used for processing the datasets without sharing the datasets themselves, and at the same time both parties can obtain the same dataset from the proper authority and independently inject it into their repository. All of this could be achieved through standard git commands without tedious and error prone verifying of long checksums.

4. The PROB Tool

4.1. Design and Components

As a solution to the problems of tracking provenance and ensuring reproducibility of big data workflows used in our research, we have started the development of PROB, a system for ensuring Provenance and Reproducibility Of Big Data workflows.

The system is based on Git, Git-Annex[15], Git2Prov[14]. The Git is used as a main version control system and a repository of the provenance data. The Git-Annex is an extension of the Git that allows for tracking of version information about large files without checking them into the repository. The reasons to use Git-Annex for keeping track of the datasets are two-fold. First, it solves the problem of inefficient handling of large files by git. Second, it allows to share our workflows with other groups without violating the terms of dataset usage. The Git-Annex stores only the hash values of datasets in the repository and keeps the data files themselves in a separate location. Since many dataset related to genetic information of human subjects are strictly controlled by NCBI and institutional IRBs, posting only the hashes of the files allows us to ensure that all the parties have exactly the same datasets, even though we never exchanged the controlled information. Rather such information was independently obtained by each research group from the source authority through the appropriate protocols.

The PROB defines it's own ontology to represent the provenance information. In addition to its own ontology, the PROB tool uses several existing ontologies discussed earlier. To represent basic provenance assertions such as authorship and artifacts, we use PROV W3C standard. The Git2Prov ontology is used to represent information contained in Git version control system. The NiPype ontology is used to express facts about data analysis workflows and the execution environments.

At present the PROB tool is geared towards workflows written in PIG language which are intended to run Hadoop Map-Reduce clusters. The tool could be used to support the pipelines such as the one depicted on figure 1. While the steps 2 through 4 of the pipeline are being executed the tool makes the record of all produced artifacts as well as exact operations that were performed on the data objects. Later the same researcher, or a different person could query the provenance store to determine the exact steps that were performed to create any artifact of the analysis also the reproducer will be able to obtain the exact input files that were used as an input to these steps. By expanding the search to the bounds of the initial and final steps the verifier could reconstruct the entire analysis pipeline.

The design goal of the PROB tool is to support both scripted and interactive workflow. The scripted workflows are the workflows saved into the script file prior to the analysis. On the other hand the interactive workflows are not stored in any files, but rather dynamically constructed by the researcher by typing commands into the PIG's interactive shell called Grunt. It is actually very important to keep track of dynamically constructed workflows since such workflows facilitate creative exploration of the data, and at the same time unlike the workflows executed from the script files there is never a record of these workflows besides rudimentary command history kept by the PIG's Grunt shell.

In addition to the third-party tools mentioned at the beginning of this section, the PROB tool contains the following components – (a) a set of python scripts, (b) instrumented PIG interpreter, (c) HDFS backend for git-annex, (d) ontology defined to enrich provenance data beyond what is provided by PROV and Git2Prov

The scripts are used to record the provenance of the code and data as well as facilitate the transfer of large data objects between different hosts involved in the analysis. The instrumented PIG interpreter is used to track the execution of interactive workflows and record which inputs and parameters were used at each stage of the workflow. The HDFS backend for git-annex creates an interface between git repository and the map-reduce file system. The extra ontology types were developed to fix problems in Git2Prov ontology, to allow execution on the Hadoop clusters and achieve greater degree of reproducibility. The extra ontology types will be explained in the examples of the next section.

4.2. The tool operation

Below we will show the example of operation of a PROB tool on a big data workflow. For the sake of simplicity let's consider this analysis. Say we wish to find a frequency of each word in a text file. For this we will use the PIG script shown below

```
A = load '/data/input.txt';
B = foreach A generate flatten(TOKENIZE($0)) as word;
C = group B by word;
D = foreach C generate COUNT(B), group;
Dump D ;
```

We assume the input file was just decrypted and still located on the Windows machine. During our analysis we want to transfer the input file to the Hadoop cluster and use the pig script above to perform the analysis.

So here are the steps that will be run to achieve this goal and the artifacts that will be produced at each step. The steps below describe the actions are performed by the PROB at each step of the pipeline as well as show the provenance data artifacts generated by the steps.

For the sake of brevity we will not include the namespace declarations in the RDF documents. Besides the standard namespaces of `xsd` and `rfds` , we will use the following namespaces `prov` for PROV ontology defined by the W3C standard, `git` for Git2Prov ontology, `n` for NiPype ontology and finally `prob` for the PROB tool defined ontology.

4.3. Initial state

At the beginning of operations we assume that researchers have setup the initial git repositories on all the hosts involved in the processing. This includes linking all the repositories together. Installing git-annex, creating all necessary meta-data for git-annex to run, establishing remote relationships between git-annex repositories, and of seting-up the git-annex HDFS plugin on the machine that is used to launch hadoop jobs.

We also assume that the processing script show above has been checked in into the git repository and the large input file has been enrolled into git-annex. After all the above operations are done, both git and git-annex repositories have been synchronized.

4.4. Recording the state of git repository

This is the first operation of the pipeline. In fact it has to be performed before and after of execution of each step in the pipeline on every host involved in processing. The operation is idempotent, meaning that it could be performed any number of times with exactly the same result provided there were no changes to the repository between executions.

This operation involves synchronizing all repositories on all hosts, and making a web-service call to Git2Prov service. The Git2Prov service downloads all meta-data stored in the git repository and returns expresses it as a set of PROV and Git2Prov assertions.

An example output is shown below. Note that for the sake of brevity the output has been significantly abridged to show only the triples relevant to discussion of the subsequent steps. From this example you can infer that there is a file named **word-count.pig** committed by the user Vlad Korolev. There is a commit with the id a2e1, and there is a particular version of the **word-count.pig** script that was created during above mentioned commit.

```
git:commit-a2e1  a  prov:Activity ;
  prov:endedAtTime "2013-05-21T09:08:21.000Z"^^xsd:dateTime ;
  prov:qualifiedAssociation [ a prov:Association ;
     prov:agent git:user-VladKorolev ;
     prov:hadRole "author,␣committer"@en ] ;
  prov:used     git:file-word-count-pig_commit-a2e1 ;
  prov:wasAssociatedWith git:user-VladKorolev ;
  prov:wasInformedBy git:commit-a2e1 ;
  rdfs:label    "update"@en .

git:file-word-count-pig prov:Entity ;
   rdfs:label  "word-count-pig"@en .

git:file-word-count-pig_commit-a2e1 a prov:Entity ;
   prov:qualifiedAttribution    [    a prov:Attribution ;
          prov:agent git:user-VladKorolev ;
          a "authorship"@en ] .
```

Unfortunately, the information generated by git2prov is not enough for our purposes. Therefore the PROB tool performs the following operation right after gets results from execution of git2prov. The operation is: go to the git repository and obtain a tree id linked to the commit id. The result of this operation is shown below.

```
prob:git_tree_23442   a   prob:git_tree ;
       prov:wasGeneratedBy     git:commit_a2e1 ;
       prov:wasGeneratedBy     git:commit_3455 .
```

It shows that the a tree identified by id 23442 could be materialized by checking either commit a2e1 or commit 3455. This step is necessary because the PROB tool uses tree id rather than commit id to restore contents of a working directory. The advantage of using tree id over commit id is that commit id contain extra metadata such as author name, date of commit, position of commit within the hierarchy. Thus, if two independent parties performed a series of operations that resulted in exactly the same dataset, the commit ids recorded by this step would be different even though the contents are exactly the same. However, the tree id are always the same for the directories with the same content. Thus making the tree more suitable for provenance tracking.

4.5. Loading the data

The next step in the pipeline will be loading the dataset onto Map-Reduce cluster. This step involves loading using a sequence of *git-annex sync* commands to make sure the data is propagate to the machine on HDFS cluster. On that particular machine the tool executes a sequence of *git-annex get* command for each dangling link in the working directory. Next the git-annex HDFS plug-in is used to push the working directory into the HDFS data-store. The small files that are commit directly to git are loaded into HDFS as is. The large files tracked by git annex are loaded into special folder. The git working directory is swept for the git-annex links to record the mapping of file names to hashes in the annex datastore. The mapping is recorded into the file and pushed on hadoop cluster.

4.6. Running the analysis

Once data has been loaded to the HDFS, the actual analysis of the data can take place. To perform the analysis the user runs instrumented version of the Pig interpreter giving the script as an argument. The interpreter reads the script and executes workflows described in it on the map-reduce cluster. For each such execution the instrumentation code outputs the analysis provenance record similar to the one shown below.

```
prob:analysis_11299    a  prob:analysis ;
        prob:  module   module_2302 ;
        n:duration  20s ;
        n:wd   /home/user1 ;
        n:ret_code    15 ;
        n:platform     MacOS ;
        n:platofrmVersion 10.5.0.2 ;
        prob:treeId   prob:git_23442 ;
        prob:isClean  y ;
        prob:input    prob:input_3579 ;
        prob:output  prob:output_4593    ;
        prob:hdfs_map  file-hdfs-entity-map-22333 .
```

This analysis record contains information about which module was executed, in what environment, with what inputs, and whether the working tree was clean. Also it records the input taken by the workflow and the output produced as a result.

Also another important modification of the PIG interpreter is the level of indirection for the input files. Although the script used in this analysis mentions the file named *Data.txt* , no such file actually exist on the HDFS system. Rather HDFS system contains a file named *SHA-133aaa* which contains the data loaded in the previous steps. The instrumented PIG interpreter uses the mapping file to resolve this indirection.

The PROB tool has a concept of a module. A module is something that could be executed which could be a script contained in a file, or a sequence of commands typed into the interactive shell. For each execution of the analysis the PROB tool identifies the executed module and outputs provenance record like this.

```
1 prob:module_23111  a prob:module ;
2         prov:Entity  file-word-count-pig_commit-a2e1;
3         prob:Dependency  prob:module_92234 .
```

Also, module could depend on modules in the file system. Line 2 of the example above, refers to the PIG script. Line 3 refers to the dependency which is in this case a PIG interpreted itself. The information about dependencies is stored in similar module records.

Besides the module information the instrumented interpreter creates a provenance record regarding the input and output. An example of such record is shown below. Note that the PROB concept of the input refers to the actual input parameters given to the script not the contents of the files.

The actual content read by the analysis workflow during particular run could be identified by combining the elements of the input record with the tree id element of the analysis execution record. Such combination will uniquely identify the actual input data used by the analysis.

```
 1 prob:input_9322  a   prob:input ;
 2      prob:parameter_1   "dataFile.tsv" ;
 3      prob:parameter_2   "10" .
 4
 5 prob:output_11234  a prob:output ;
 6      prob:consoleOutput_23222 .
 7
 8 prob:consoleOutput_3322 a prob:console ;
 9      prob:run_222_line_1    "dog␣34334343" ;
10      prob:run_222_line_2    "cat␣123344" .
```

The output record is shown on the lines 8-10. In this particular case for the sake of illustration we used the console output. In practice use of console outputs should be discouraged and all the intermediate results show be saved to the files. The console output record consist of a collection of output lines produced by the script.

After the desired output is obtained and recorded the goal of analysis is achieved and all the provenance records are stored. At any future time the original researcher or any third party could recreate this analysis. Since all the artifacts used in the analysis are identified by strong checksums one could determine with high degree of certainty that the analysis performed later on is equivalent to the analysis described in the records.

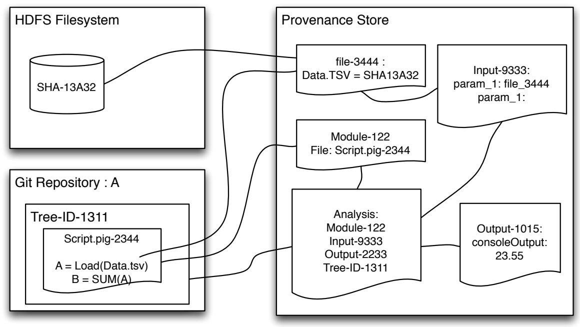

Figure 2. Typical scientific big data workflow.

The figure 2 shows the relationship between all objects described above after the analysis took place. The environment consists of a git repository A, that contains a git tree identified by id 1311. This tree contains a file identified by name `Script.pig` and hash 2344. This file was executed as part of the analysis for which the analysis record was created. The analysis record refers to the module identified by id 122, the module in turn points back to the PIG script file. The script mentions the file with id 3444 used as an input. This file id resolves to an object stored in git-annex with the hash value of SHA-13a32. This object is also stored in the HDFS under the same name. When the script is executed the custom indirection layer of PIG interpreter resolves the file name `data.tsv` to the actual file stored HDFS and performs the analysis on that file. The output is printed to the console, which in turn captured by the interpreter and placed into provenance record Output-1015. The output record is in turn linked back to the analysis record. This collection of records contains enough information for someone to reproduce exactly the same analysis at any point later in time.

4.7. Conclusion and future work

In this work we have shown a design of a PROB tool and corresponding ontology. We have shown how the provenance information of every computational step in the big data data workflow could be captured and stored. We have shown that provided all the provenance information is kept, the analysis could be recreated later on. The tool described in this work also allows different teams to collaborate together by sharing the workflows without sharing the actual dataset. This is a major concern for bio-informatics work that involves human subjects. In the near future we will continue development of this tool with the intent of sharing it among the fellow researchers. Although this particular implementation of the PROB tool is geared towards Map-Reduce workflows written in PIG, the general approach and developed ontology could be re-used by other scientific workflow tools.

The tool will be solidified through additional experiments. Then we will derive a final ontology that uses work by the groups mentioned above and is rich enough to express provenance for high degree of reproducibility.

Although we have shown how high degree of reproducibility could be achieved using the method described above, some challenges still do exist. One particular challenge is reproducing results of stochastic algorithms such as CART or K-Means. Another challenge is a case of two different analysis arriving at exactly the same result even though the input and algorithms were different, which common case for processing of satellite imagery.

References

[1] D. L. Donoho, “An invitation to reproducible computational research,” *Biostatistics*, vol. 11, no. 3, pp. 385–388, 2010.

[2] J. P. Ioannidis, “Why most published research findings are false,” *PLoS medicine*, vol. 2, no. 8, p. e124, 2005.

[3] R. Moonesinghe, M. J. Khoury, and A. C. J. Janssens, “Most published research findings are falseÑbut a little replication goes a long way,” *PLoS medicine*, vol. 4, no. 2, p. e28, 2007.

[4] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] National center for biotechnology information. [Online]. Available: http://www.ncbi.nlm.nih.gov/

[6] K. Zhang, K. Chen, and W. Xue, “Kangaroo: Reliable execution of scientific applications with dag programming model,” in *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, 2011, pp. 327–334.

[7] C. Scheidegger, D. Koop, E. Santos, H. Vo, S. Callahan, J. Freire, and C. Silva, “Tackling the provenance challenge one layer at a time,” *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 473–483, 2008.

[8] K. Gorgolewski, C. D. Burns, C. Madison, D. Clark, Y. O. Halchenko, M. L. Waskom, and S. S. Ghosh, “Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python,” *Frontiers in neuroinformatics*, vol. 5, 2011.

[9] V. Zaliva and V. Orlov. (2012) HAMAKE: A Dataflow Approach to Data Processing in Hadoop. https://code.google.com/p/hamake/. Accessed 04-January-2014.

[10] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat *et al.*, “Taverna: a tool for the composition and enactment of bioinformatics workflows,” *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[11] M. J. Rochkind, “The source code control system,” *Software Engineering, IEEE Transactions on*, no. 4, pp. 364–370, 1975.

[12] P. Buneman, A. Chapman, and J. Cheney, “Provenance management in curated databases,” in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 539–550.

[13] L. Moreau and P. Missier, “Prov-dm: The prov data model,” 2013.

[14] T. De Nies, S. Magliacane, R. Verborgh, S. Coppens, P. Groth, E. Mannens, and R. Van de Walle, “Git2prov: Exposing version control system content as w3c prov,” in *Posters & Demonstrations Track within the 12th International Semantic Web Conference (ISWC-2013)*. CEUR-WS, 2013, pp. 125–128.

[15] J. Hess. git-annex. http://git-annex.branchable.com/. [accessed 04-January-2014].