Project SaYa : Tamper Proof Temporal Provenance Storage Platform

Student Name: Sahar Siddiqui
Roll Number: 2014091

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Computer Science & Engineering
on April 18, 2017

BTP Track: Research Track

BTP Advisor
Dr. Sameep Mehta
Dr. Ponnurangam Kumaraguru

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

I hereby declare that the work presented in the report entitled **Tamper Proof Temporal Provenance Storage Platform** submitted by me for the partial fulfillment of the requirements for the degree of *Bachelor of Technology* in *Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr. Ponnurangam Kumaraguru and Dr. Sameep Mehta**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

..... **Place & Date:** New Delhi, April 18, 2017
Sahar Siddiqui

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

..... **Place & Date:** New Delhi, April 18, 2017
Dr. Ponnurangam Kumaraguru

Abstract

In the era of big data where every individual is a target of intensive data collection, there is a need to create technological tools that empower individuals to track what happens to their data. Provenance has been studied extensively in both database and workflow management systems, so far with little focus on text-retrieval based workflows with user defined operators. Such kind of workflow provenance aims to capture a complete description of evaluation (or enactment) of a workflow, and this is crucial to this problem of personal data use. As an initial step to solving this problem, the work presented in this report aims at developing our own tamper proof temporal provenance storage platform and query based model that can track, store and analyze data transformations.

Keywords: Privacy, Information Retrieval, Text Retrieval, Provenance, Data Lineage, Query Model

Acknowledgments

I would sincerely like to thank my advisors Dr. Ponnurangam Kumaraguru and Dr. Sameep Mehta (collaborator from IBM Research) for their constant guidance and enduring support. They redirected me to the right and helpful resources and provided me with valuable suggestions at every stage of the project. I would also like to thank Yashovardhan Sharma (B.Tech 2013121) who worked on the same as an Independent Project and helped me with constant technical guidance.

Work Distribution

In the first semester, my work was mostly on understanding the the need for a better solution to the problem of personal data use and the privacy issues related to it. We argued that building a tool that would enable users to browse how a company or government agency uses their personal data is a key requirement for a further increase in data transparency. After identifying the key research challenges and requirements for building such a data provenance tool, we discussed our initial approach that aims at enabling a provenance supplemented data workflow which can respond to temporal queries using Apache NiFi. This portion is described in the second chapter.

My second semester started with the discussion about how NiFi would be difficult to integrate with the existing workflows and the focus of the project shifted to building an in-house text-retrieval based lineage storage and query system which would track data transformations in the workflow on the go and store it in an efficient graph based query-able form. I further discuss the state-of-the-art approaches for storing different types of provenance data and how graph based approach seemed to be the most suitable in our case. In the end, we were able to build a system to store and query provenance data using the well known graph database, Neo4j. This portion is described in the third chapter.

Contents

Chapter 1

Introduction

1.1 Motivation

In the era of Big Data, every person on the Internet is a target of the intensive data collection process by parties ranging from marketing companies to government agencies. Anecdotal evidence suggests that opting out of the data collection process is almost impossible nowadays [1]. As a result, measures have been taken to increase data transparency. Many companies now publish their privacy policies stating when, where and how they may use or collect a user's data. However, there are shortcomings in these privacy policies as well : being merely English documents, they seem to be too confusing for the users and too vague for the experts.

When it comes to personal data use, we need better choices than just opting out. Perhaps, a better solution, would be to create a tool where the user can track what happens with their data. Individuals should have access to a Personal Data Use Workbench, where they can browse how a company or government agency is using their data. A similar problem has been addressed in the scientific workflow systems where scientists wanted to capture the provenance or lineage of a data product which contains information about the process and data used to derive the data product [2]. Thus, a similar technique could be applied to this problem of personal data use to track the provenance of data.

1.2 Research Challenges

Building such a tool isnt too simple. As described in [2] following are research challenges in this task :

- Answering a detailed provenance query would require access to the entire dataset which could lead to privacy issue for the other user's information
- Dealing with complex operators which are used in personal data processing while tracing provenance

- Deciding querying language, a structure and fast processing algorithms to easily deal with complex queries posed by users which may not occur in a typical scientific workflow.

My main focus in this project has been into dealing with the second and third research challenge. The operators used in personal data processing are complex and heterogeneous. Tracing provenance (data lineage) in frameworks like Map Reduce [4] and Pig [5] already exist but they are very specific to operators that the particular framework can handle. Tracing provenance, on the other hand, in a pipeline with arbitrary operators is difficult. When operators are user-defined functions, the most realistic approach is to provide a provenance API and ask a human developer to specify how the operator maps input data to output data. This approach was pioneered in the SubZero system [6] but needs to be extended beyond the setting of scientific data processing.

1.3 Proposed Solution

In sum, the broader goal of the project is to make a tool which enables users to track the usage of their personal data over the internet. Further sections of this report explain the initial approach to building a platform which enables users to make temporal queries on data provenance in a workflow using the technology of Apache NiFi followed by the second approach of building an in-house text-retrieval based lineage storage and query system which would track data transformations in the workflow on the go and store it in an efficient graph based query-able form.

Chapter 2

Apache Nifi based approach

2.1 Work Done

We began our work by choosing Apache NiFi as a platform for composing and executing a data workflow and recording the provenance data associated with it. We then went onto building our own platform which could store the provenance data extracted from NiFi and respond to temporal queries related to the data workflow.

2.1.1 Understanding NiFi

During the early stage of the project, we spent most of the time in understanding the working of Apache NiFi as the technology was completely new to us. NiFi is an open source system designed to automate the real time flow of data between systems. It is a powerful system which can be used to build data workflows to process and distribute data. A data workflow in NiFi has two major components :

- FlowFile : Each piece of "User Data" (i.e., data that the user brings into NiFi for processing and distribution) is referred to as a FlowFile. A FlowFile is made up of two parts: Attributes and Content. The Content is the User Data itself. Attributes are key-value pairs that are associated with the User Data.
- Processor : The Processor is the NiFi component that is responsible for creating, sending, receiving, transforming, routing, splitting, merging, and processing FlowFiles. It is the most important building block available to NiFi users to build their dataflows. NiFi has a total of 154 in-built processors which can be directly used to compose a workflow. We can also code and create custom workflows and add them to our Nifi environment

Another key feature of Nifi, which makes it crucial for our project is its concept of Data Provenance, also known as Data Lineage. While Lineage here refers to recreating the path of how the data was derived, Data Provenance also comprises of the state of that data at each of the stages

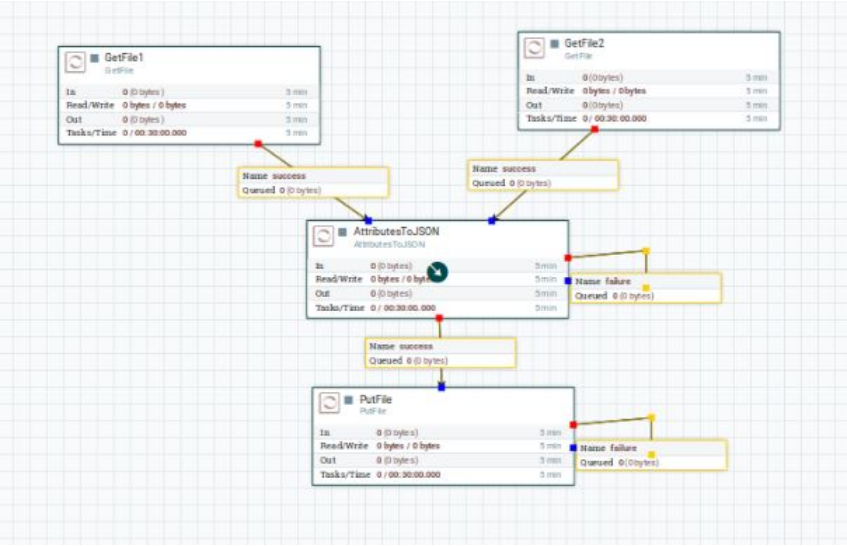

Figure 2.1: A typical workflow in NiFi

of this path. The information gained from this feature can thus be used to respond to temporal queries on the data flow, making it apt for our project. To support this feature, NiFi maintains three repositories :

- Flowfile_repository : Contains the attributes of flowfiles currently in use and a pointer to its content. Essentially, it keeps a log of changes made in the flowfiles as they flow through the system.
- Content_repository : Contains the actual content of the flowfiles. Essentially, the content of a flowfile is a superset that encompasses all the sets of input and output for each of the processors in the workflow.
- Provenance_repository : It keeps a record of every event that occurs for a flowfile (a new provenance event gets created when a flowfile is created, forked, cloned, modified etc.). This provenance event encaptures the state of the flowfile at that instant, comprising of its attributes, a pointer to its content and its relationship with other provenance events.

To support high throughput, NiFi allows writing to multiple disk partitions simultaneously while updating these repositories. Hence, during an eventflow, multiple journal files get created in these repositories and multiple threads write to these journal files. After a considerable period of time (30 seconds, by default), all the existing journal files get merged into a single Provenance Event Log file in a zipped folder written in hex format. This file could be converted to text to extract the provenance details for the eventflow till that instant.

2.1.2 Experimenting with NiFi

Our next step was to determine the kind of provenance data that can be obtained using NiFi. We did this by testing different types workflows on NiFi and identify the kind of provenance data associated with each workflow. Given below are some of the questions that we had to answer and a description of the workflows we created to test the same :

- *Can we track which processes acted on the data?*
  For this, we created a database of 10 students containing their Name, Roll Number and Marks as columns. Using the ExecuteSQL Processor [1] we filtered a list of students having marks greater than 70. Since the format in which the result of ExecuteSQL is saved is Avro, we used another processor ConvertAvroToJson[2] to convert the result into a JSON object (basically, key-value pairs). Finally, we used the PutFile[3] processor to save this resultant JSON file. Figure 2 shows the workflow we created, for the given process. We observed that given a dataset and a set of processes, we can track which subset of processes act on that subset of data.

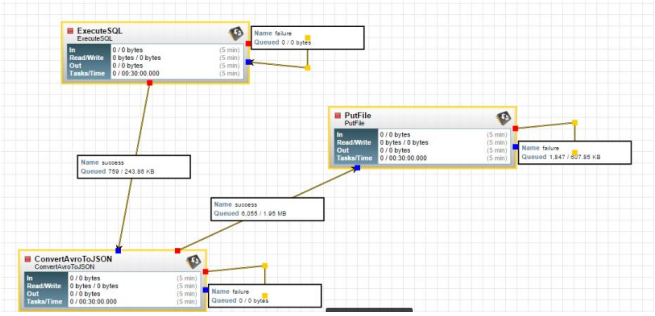

Figure 2.2: NiFi Workflow for MySQL database containing "Student" data

- *Once we filter the data and perform some data transformation on it, can we track which subset of data has been processed and can recreate the pipeline?*
  For this, we created another dataflow by using the same database with the same table Students. PutSQL[4] processor was used to make the first query - **create table tempsample (Select * from students where Marks¿70);** - More elaborately, we filtered some data out of the original table and saved it into a temporary table tempsample so that we can further execute queries on this data. The ExecuteSQL processor was used to calculate the Average of marks present in the tempsample table (filtered data). Result obtained in the AVRO format was converted to JSON format and saved to disk. Figure 3 shows the

[1]ExecuteSQL is an in-built processor in NiFi which takes in as input, a flowfile with a SQL select query written in it and executes the command if NiFi is successfully connected to the MySQL database

[2]ConvertAvroToJson is an in-built processor in NiFi which simply converts the content of the incoming file from Avro format to JSON format

[3]PutFile is an in-built processor which takes in a flowfile as an input and write its content to a file in the destination folder

[4]PutSQL is an in-built processor in NiFi which takes an SQL update query as an attribute and executes it if NiFi is successfully connected to the MySQL database

workflow we created to acheive this. We observed that some amount of pipeline can be recreated if we store data in temporary files and then combine it programmatically later on when the processes end. However, it can not be created simultaneously along with the NiFi processors. Also, the data, if needed to be tracked by NiFi, needs to go through NiFi in the form of Flowfiles. It will ultimately be stored locally on the machine, however it will have to be accessible by NiFi to able to flow through it.

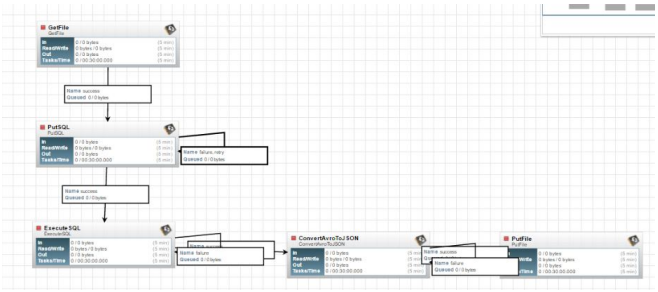

Figure 2.3: NiFi Workflow that further processes "Student" data

- *In case of a complex dataflow where data from multiple sources goes into a common processor, can we create a separate lineage for each of the datasets?*
  For this, we created a dataflow where the data from 2 sources (2 different files present on local disk) and went into a processor (AttributesToJSON). The result of this went to PutFile processor which basically writes the result to files on disk. Figure 1 shows a screenshot of the workflow we created. We observed that even though data from two different sources was going to a common processor, internally NiFi treated their FlowFiles separately, i.e. it created separate Flowfiles for the two data sources and every processor acted on them individually, thus forming a separate lineage for both the flowfiles.

2.1.3 Building our own Platform

After exploring the provenance feature of NiFi and confirming that we can extract the provenance data from NiFi's central storage, we moved to the next phase of our project : building our own temporal provenance storage platform. We extracted the provenance information regarding various processors from Provenance Event log files (.prov files) and stored it in a database on a server (localhost, for now). Next, we created a platform (currently, command-line based) where users can make temporal queries on data provenance in a workflow.

2.2 Pipeline

2.2.1 Composing and Executing Data workflows

We used NiFis web interface to compose and execute the data workflow. For our sample workflow, we used GetFile, AttributesToJSON and the PutFile processors. Figure 1 shows the sample

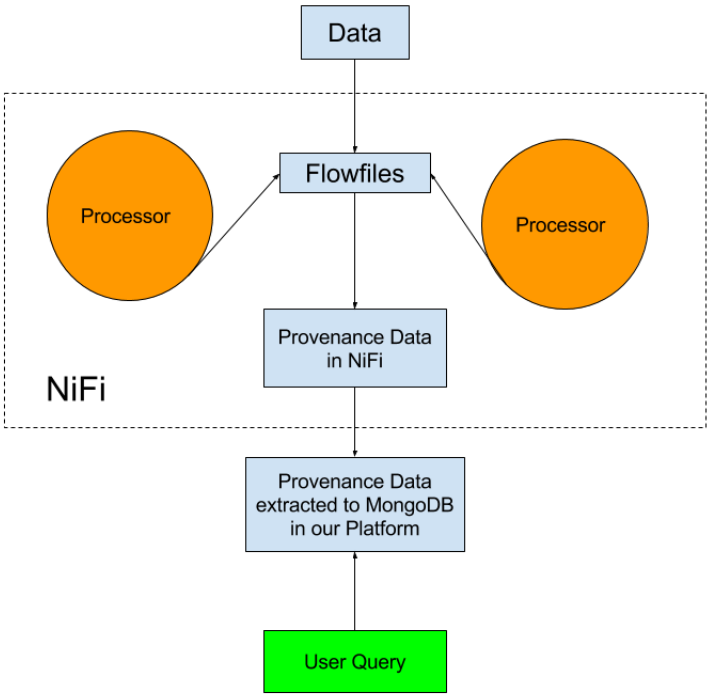

Figure 2.4: Architecture Diagram

workflow. Once the data workflow gets executed, the data gets processed and provenance information associated with the workflow gets generated and stored in the Provenance Event Log file situated in the provenance repository.

2.2.2 Extracting data from Nifi

Extracting data from the provenance event log file was a rather challenging task as NiFi itself encodes the file in unique way. We created a python script which ran in the background and waited for a Provenance Event Log file (.prov file) to be created in the Provenance repository. The file is created in a zipped folder written in hex format (.prov.gz format). Once the file gets created, the script takes the .prov.gz file, unzips it and converts it into plain text. It then traverses through the file, removing the unnecessary characters and extracts the useful information related to every event in the workflow.

2.2.3 Storing Provenance Data

After extracting the data from the Provenance File, we store it on a NoSQL (MongoDB) database to make it query-able. MongoDB stores data in the form of JSON documents and makes querying efficient. Each entry/document in the database corresponds to an event that took place in the workflow and comprises of the meta-data (content and attributes) of processor and

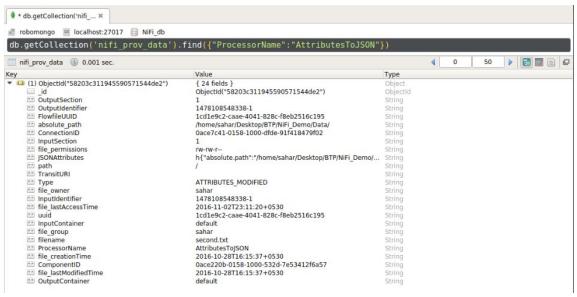

Figure 2.5: Data stored in our MongoDB Database

flowfile involved in that event. There are a total of 24 fields in every record. Figure 5 shows a screenshot of the data stored in the MongoDB database.

2.2.4 Query Model

Another script was written to create a command-line based querying system. Currently, we focused majorly on two types of temporal queries :

- Given a time instance t, we should be able to see the state of the eventflow, i.e the state of the files and what processes took place at that time etc.
- Given time instances t_1 and t_2, what is the difference of state of the eventflow between time instances t_1 and t_2.

Currently, the platform has a CLI which first asks the user for the type of temporal query to be made and then takes as input the dates as the time instances. The result returned is a list of key-value pairs for every event that took place at that time instance or in the specified interval of time. There are a total of 24 fields that are returned for each processor, major ones of them being - ProcessorName, filename, file_creationTime, file_lastAccessedTime, file_lastModifiedTime, FlowfileUUID etc. Figure 6 shows an instance of our command-line based platform.

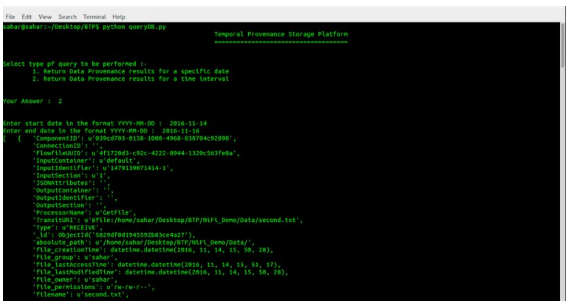

Figure 2.6: The Command-line based platform

Chapter 3

Graph based lineage storage approach

Although NiFi serves as a great platform for recording provenance in a workflow, it cannot be integrated with any existing workflow very easily as all the processes need to be present (and hence, coded) in the NiFi system and all the inflow and outflow of data need to be through NiFi.

Working towards the primary aim of this project, in this approach we shifted towards building our own text-retrieval based lineage storage and query system. Replicating a somewhat OSM behaviour, the system consisted of a dataset with text-retrieval based analysis conducted on it and the results were stored efficiently as a set of timed key-value pairs. The primary aim of this in-built system was to track the data transformations, store the provenance data in an efficient query-able form and retrieve information from this stored temporal data to generate a pipeline for explaining the result of a query.

3.1 Running Example

As a simplified example that serves primarily to illustrate our definitions and techniques, consider the workflow: Given a set of 50,000 movie reviews extracted from IMDb with multiple reviews per movie, we analyse each movie review by its sentiment and give it a rating ranging from 1 to 5. We then calculate the aggregate of all the ratings for each movie title to give a final rating to the movie.

An example of how provenance might be useful in this workflow, suppose we are surprised to see that Twilight has got a very high rating of 4 and is falling under the category of Good Movies (ones with rating above 3). Tracing provenance back one level we see that there are over 30 good ratings which contributed to the final rating. Further tracing provenance all the way back to the original movie reviews, we can view and read the movie reviews which actually contributed

to the high rating. As an extension, if a username is attached to the review, we can view all the other reviews posted by the user and infer that teenage girls in particular might have been flooding IMDb with good reviews.

3.2 Methodology

The main technical challenge is how to represent workflow so that these lineage queries can be answered efficiently. In this section we describe, at an abstract level, how provenance according to our definitions can be captured and stored during workflow execution.

3.2.1 Provenance Capture

We develop a graphical model for representing provenance in a workflow and call it a Provenance Graph. A provenance graph G = (V,E) is an undirected acyclic graph. Each node in the graph is uniquely identifiable by an ID and represents a data value (input, intermediate and output). In our running example, these data values correspond to movie_title, movie_review, review_rating and final_rating. Edges represent the transformations which link the input data values to the output data values. Since there can be multiple incoming and outgoing undirected edges associated with a node, each of these edges are labelled with a property or a transformation which links the adjacent nodes.

3.2.2 Provenance Tracing

Now, suppose we have our running example workflow, and we would like to find the provenance of an output element in terms of the initial inputs. We implemented a fairly straightforward recursive backtracking algorithm which traverses through the provenance graph to extract the data lineage. Given an output element ID, our system traverses through each of its incoming edges and reaches the corresponding. The incoming edges can be identified with the labels. In our running example, we trace back the graph by traversing through all the neighbors of the output element (final_rating) which would lead us to the individual review_rating and then for each individual review_rating we can follow the edges labeled as "Sentiment_Transformation" to find the exact movie reviews which contributed into forming the final_rating. Our tracing scheme, however, has not yet been made as efficient as possible.

3.2.3 Persistent Storage

While defining the provenance model is one task, provenance storage is another major challenge. Provenance differs from other forms of metadata because it is based on relationships among objects. As defined in the above sections, it includes ancestry relationships which are in the form of undirected graphs. Previous researches on provenance systems have used relational, XML, or RDF storage, and the corresponding languages for querying [7]. However, it has

been seen that none are well suited for querying provenance data [8]. The data and query models for provenance should directly and naturally address this graph-centric nature of provenance.

For these reasons, we make use of Neo4j for persisting the provenance graph and the data values. Neo4j is a well known graph database, which is easy to use with its python framework called Bulbflow. The movie reviews and movie titles which are the initial inputs to the workflow are stored as separate files on the system.

The implementation details for the work in Neo4j are yet to be added to this report and will be added as soon as possible.

Bibliography

[1] My experiment opting out of big data made me look like a criminal. Time Magazine, May 1st, 2014, http://time.com/83200/privacy-internet-big-data-opt-out/.

[2] Lucja Kot, Tracking Personal Data Use: Provenance And Trust, http://cidrdb.org/cidr2015/Papers/4_Abstract_5KL.pdf.

[3] S. B. Davidson and J. Freire. Provenance and scientific workflows: Challenges and opportunities. In SIGMOD, 2008.

[4] R. Ikeda, H. Park, and J. Widom. Provenance for generalized map and reduce workflows. In CIDR, 2011.

[5] 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., 5(4):346357, Dec. 2011.

[6] E. Wu, S. Madden, and M. Stonebraker. SubZero: A fine-grained lineage system for scientific databases. In ICDE 2013.

[7] L. Moreau et al. The First Provenance Challenge. Concurrency and Computation: Practice and Experience. Published online. DOI 10.1002/cpe.1233, April 2008.

[8] David A. Holland, Uri Braun, Diana Maclean,Kiran-Kumar Muniswamy-Reddy, Margo I. Seltzer. Choosing a Data Model and Query Language for Provenance. http://www.eecs.harvard.edu/ kiran/pubs/ipaw08.pdf