Piglet – Interactive and Platform Transparent Analytics for RDF & Dynamic Data

Stefan Hagedorn
TU Ilmenau, Germany
stefan.hagedorn@tu-ilmenau.de

Kai-Uwe Sattler
TU Ilmenau, Germany
kus@tu-ilmenau.de

ABSTRACT

Data analytics has gained more and more focus during recent years and many data processing platforms have been developed. They all provide a powerful but often complex API that users have to learn. Furthermore, results can only be stored or printed, without any possibility for visualization. In this paper we present Piglet, a compiler for the high-level Pig Latin script language that generates code for various platforms like Spark, Flink, Storm, and PipeFabric. Piglet lets users write elegant code with extensions for SPARQL and RDF, as well as support for streaming data. An integration into the notebook-based frontend Zeppelin provides a homogeneous and interactive user interface for exploring, analyzing, and visualizing data from different sources and lets users share their scripts and results.

1. INTRODUCTION

Exploring large and often unknown data sets is one of the most challenging tasks in unlocking, preparing, and analyzing data to support decisions and derive models in various domains. Data exploration often implies the incremental analysis of data sets, starting with data cleaning and removing invalid entries and then finding a way to the information of interest. It requires easy to use interactive interfaces to quickly sketch queries and produce first results, instead of writing complex source code. Immediate visualization is another important factor in order to evaluate the next exploration steps. Finally, particularly dynamic data sets represent a big challenge for interactive exploration simply due to the amount of data and the continuous changes.

During recent years, many data processing platforms have been developed. The MapReduce paradigm and its implementation in Hadoop allowed for fault-tolerant and scalable analyses of large data sets. The Pig engine [3] provides the script language Pig Latin that allows people to avoid writing plain Java code. Pig Latin scripts are compiled into MapReduce programs, which are then executed on a Hadoop cluster.

WWW'16 Companion, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4144-8/16/04.
http://dx.doi.org/10.1145/2872518.2890530.

However, MapReduce jobs are slow also for small data sets and thus, new projects started and quickly became popular. The Apache Spark[1] platform uses an in memory data structure called Resilient Distributed Dataset (RDD) to speed-up execution. To support stream processing, Spark provides micro batches that wrap incoming values in RDDs and pass them to the Spark engine. Along with Spark, the Apache Flink[2] has become popular, too. It provides batch processing as well as a pipelined streaming engine.

Although these two platforms have gained much attention recently, in our opinion they miss some key features that would help data workers to be more efficient and productive: Spark, Flink, and other platforms provide an API for various programming languages, e.g., Java, Scala, or Python. Users that want to use these platforms, have to write their programs in any of the supported language. However, this may be a hard task for non-technically minded users with no programming background. Also, if they ever need to change the underlying platform, all programs have to be rewritten from scratch to adapt to the new API. Furthermore, these platforms only support (nested) tuple data with a fixed schema, but have no special support for Linked Data, making it hard to analyze RDF data sets and to query the vast amount of information provided in the Linked Open Data cloud. The contribution of Piglet is to bring all these fields together:

- Piglet allows to express the analysis task in the high level script language Pig Latin that was extended to support SPARQL, stream processing, as well as complex event processing (CEP) operations.
- Input scripts are compiled into programs for any of the different supported backends, making the underlying platform transparent to the user.
- The integration with the Apache Zeppelin[3] project allows to easily create Pig Latin scripts in a browser based editor, execute them and to visualize and share the results.

2. PIGLET

Piglet[4] is a code generator that creates programs for various platforms from a Pig Latin input script. In Piglet terms, a backend is a library that encapsulates a specific platform. We follow a plugin-style approach so that users can create their own backends that only have to comply to an interface through which Piglet can communicate with it and it

[1] `https://spark.apache.org`
[2] `https://flink.apache.org`
[3] `https://zeppelin.incubator.apache.org/`
[4] available at `https://github.com/ksattler/piglet`

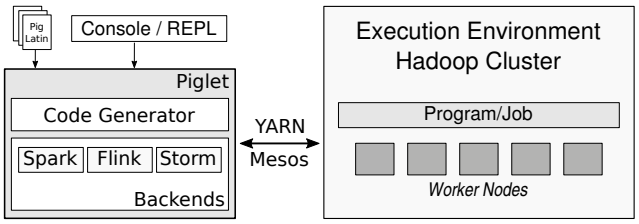

Figure 1: Overview of Piglets internal architecure.

has to be placed in a location where it can be found at runtime. Currently, we support the backends Spark, Flink, Storm[5], PipeFabric [4], as well as the original Pig engine. The overall architecture is depicted in Fig. 1. Piglet accepts script files as input, but also provides an interactive shell, similar to the Spark shell or Pig's grunt shell, were users can enter their commands. The scripts are passed to the code generator which translates them into a *DataflowPlan*, our internal intermediate representation. In an analysis and rewriting step, transformations are applied for optimization and SPARQL support (see below). Using the logical operators in the plan, the code generator creates the program for the chosen backend and packs it into an archive that is sent to the execution environment. It is the task of the backend to provide methods to submit the job archive to the platform. While we create Scala code for Spark and Flink, we produce C++ programs for Storm and PipeFabric. The code generator uses templates files provided by the backends, making the API of the target platform is transparent to Piglet. For the original Pig engine, we simply pass the Pig script as is to their compiler.

Piglet supports all operators and features as the original Pig Latin language, but also includes extensions and new language features.

SPARQL.

The Linked Open Data principle aims at providing information in a structured, machine readable format called RDF. The de-facto standard query language for RDF data is SPARQL whose main part are basic graph patterns (BGP) that are used to formulate the query. However, original Pig Latin has no support for SPARQL and can only formulate such queries using complex and cumbersome Join/Filter constructs. Piglet supports such BGP by extending Pig Latin with an additional `BGP_FILTER` operator.

```
rdf = RDFLOAD('file.nt');
filtered = BGP_FILTER rdf BY {
 ?artist <produced> ?record .
 ?artist <country> ?country .
 ?record <release> "2015" .
};
aggr = GROUP filtered BY country;
```

The above example loads a local RDF file and finds all artists that have released a record in 2015. The variable names from the `BGP_FILTER` are available in the rest of the script for further processing. However, Pig's native tuple data model does not fit the triple model of RDF well. Therefore, in [2], we analyzed rewriting rules that transform `BGP_FILTER`s into traditional Pig statements (i.e. `Join`/`Filter` combinations) and introduced a "tuplified" schema with which RDF processing can be improved significantly. To load remote RDF data, we provide an additional `SPARQLLoader` that sends a query to a SPARQL endpoint and makes the result available in the script:

```
rdf = LOAD 'http://endpoint.org:8080/sparql'
        USING SPARQLLoader(
         'CONSTRUCT ?s dbo:populationTotal ?pop
          WHERE { ?s rdf:type dbpedia:City .
                  ?s dbo:populationTotal ?pop }')
      AS (subject, predicate, object);
```

Stream processing & CEP.

Pig Latin was designed to create MapReduce programs that process batch data. Piglet, however, also supports stream processing which requires some new operators. To connect to a stream and receive and send data, Piglet provides special loader functions that connect to a TCP socket or communicate via ZeroMQ. A core operation in stream processing is the *window* function that holds a snapshot of the stream on which aggregates can be computed. To support windows, we introduced the `WINDOW` operator that can be used to define windows on a range of elements or on a period of time. The following example shows a word count example with a window of the most recent 5 seconds which is moved by 5 seconds, making it a tumbling window. The window content is grouped on the first column to count the number of words currently being in the window.

```
a = SOCKET_READ 'tcp://127.0.0.1:8889';
w = WINDOW a RANGE 5 seconds SLIDE RANGE 5 seconds;
grpd = GROUP w BY $0;
cntd = FOREACH grpd GENERATE group, COUNT(w);
```

When working on streams, a frequent task is to identify recurring patterns. This can be achieved by complex event processing (CEP) techniques, where users define sequences and combinations of events using logical operators (conjuction, disjunction, negation) and temporal operators (e.g. sequence). Piglet provides the `MATCH_EVENT` operator that allows to define the event pattern using sequences and logical operators as well as additional filters and time ranges. This example shows how to define the pattern to detect a complex event based on the Linear Road benchmark [1]:

```
a = SOCKET_READ 'tcp://127.0.0.1:8889' AS (ts:long,
      id: int, speed double, dir: int, seg: int);
b = WINODW a RANGE 60 seconds;
c = MATCH_EVENT b PATTERN SEQ (A, B) WITH (B:
     (id == A.id && dir == A.dir && seg == A.Seg));
```

This finds all sequences if events (vehicles) that are in the same road segment and drive in the same direction within a time window of 60 seconds.

UDF.

User defined functions (UDF) can be included via the known `REGISTER` statement which includes *jar* files that contain functions the script wants to use. Often, users just need single methods that perform a specific task which is not directly available in Pig Latin. Creating an extra library for just one operation includes too much overhead and thus, in Piglet users have the possibility to directly embed code inside a `<%` ... `%>` environment. The following snipped shows the definition of a Scala method that computes the Euclidean distance between a reference point and a point from the data set.

[5] `https://storm.apache.org/`

```
<%
 def euclid(x: Double, y: Double): Double = {
  val refX = 20
  val refY = 10
  return Math.sqrt( Math.pow(refX - x ,2)
           + Math.pow(refY - y,2) )
 }
%>

raw = LOAD 'file.csv' as (x: double, y: double);
dists = FOREACH raw GENERATE euclid(x,y);
```

Piglet also allows to create macros, as known from traditional Pig Latin, using the `DEFINE` keyword. These macros can be used for re-occurring Pig statements as well as to shorten the call of some UDF in a library.

Integration & API.

A famous tool for data scientists is the powerful R[6] project for statistical computing. To easily integrate R into the analytic tasks that are created with Piglet, we allow to include R code into Piglet scripts:

```
out = RSCRIPT in USING '<R code>';
```

This can be used to easily run R functions or create plots.

We also provide a convenient API to include Piglet code into Scala or Java programs. Just like many applications include SQL statements in their code to access (relational) databases, we believe that a lot of users will need to integrate the results of their Big Data analyses into their programs. Therefore, our API lets users easily formulate any Pig Latin code, execute it on a (remote) cluster, and retrieve the results back in their program:

```
val plan = piglet"a = LOAD 'file.csv'; DUMP a;"
val result = plan.execute()
```

This is achieved by a custom Scala string interpolation function *piglet* that compiles the given string into a *DataflowPlan* which can be executed (currently, in batch mode only).

3. DECLARATIVE DATA ANALYSIS WITH NOTEBOOKS

Piglet was designed to allow everyone to run data analysis tasks without any programming skills by using the high-level dataflow language Pig Latin. Users write their scripts in a text editor or enter them in our shell and the results will be saved to disk or printed on the console. However, when exploring data sets users need visualization and collect several scripts and plots in one place. From this need, the idea of interactive documents and notebooks emerged. Apart from the opportunity to combine text, executable scripts and plots in a single (web) document, the most interesting aspect of notebooks like Jupyter and Zeppelin is interactivity: previously entered commands and expressions can be modified which immediately updates former results and plots. This interaction paradigm is very well-suited for exploring and analyzing large and dynamic data sets. In Zeppelin a notebook consists of paragraphs, where each paragraph contains text or code and optionally the result and visualization. The code is passed to an interpreter, that can be selected for each paragraph individually, which communicates with the actual platform to submit the job and retrieve the results. Zeppelin already comes with a lot of interpreters for various languages: Markdown for text and wiki-like documentation, Spark, SQL, and many more.

Piglet includes a Zeppelin interpreter so that users can simply enter their Pig scripts in Zeppelin and execute them using Piglet. An instance of the Piglet interpreter generates code for one target backend. This target backend and other parameters can be configured via the properties page of Zeppelin. Hence, users have the full interactive environment of Zeppelin combined with the flexibility and power of Piglet.

Fig. 2 shows a screenshot of a Zeppelin notebook with one paragraph. The paragraph shows a Piglet script that loads some (preprocessed) DBpedia files and applies a `BGP_FILTER` to find the regions persons were born in. The result of that `BGP_FILTER` is then processed just like "normal" data and grouped by region to count the number of people for each region. As an extension to Pig Latin, Piglet includes the `DISPLAY` operator that prepares the result formatting for visualization in Zeppelin.

4. DEMO

During the demo session, the audience will be able to interact with Piglet itself and the Zeppelin frontend. Since Zeppelin is web based, the access can either be through our laptop or via (own) mobile devices. Users will be able to run our prepared analysis scripts, edit them, or create new ones as well as add new paragraphs and notebooks to get their own impressions.

Static Data.

We will prepare sample static data sets from real world sources and prepare scripts to process them. Our static data sets will include CSV files taken from GDELT[7], as well as RDF data from DBpedia. The GDELT projcet provides event data extracted from news reports of various web sites. Our scripts will demonstrate analytic workflows that make use of both, CSV and RDF data, and also join flat CSV data with RDF. For loading RDF data, we will include scripts that retrieve data live from remote SPARQL endpoints.

Users will be able to run one and the same script on different backends without any changes, inspect the produced code, and compare the results. Using the integration in Zeppelin, the results can be visualized immediately.

Dynamic Data.

To show the support for the stream processing backends, we further include scripts that work on such data streams. These scripts will connect to local as well as to remote streams and process incoming tuples. One of these streams will be live data from Twitter's public streams. Our sample scripts will compute aggregates on the incoming Tweets, e.g. the average number of tweets with a specific hashtag in the last 10 minutes. Additionally, we also include a data generator that streams the previously mentioned (local) static data sets, to be able to compare to the results of batch processing.

Same as for static data, users can run each script on all supported backends without changing any line in the script. Furthermore, we will show sample workflows that combine CSV, RDF, and streaming data in one script.

[6] `https://www.r-project.org/`

[7] `http://www.gdeltproject.org/`

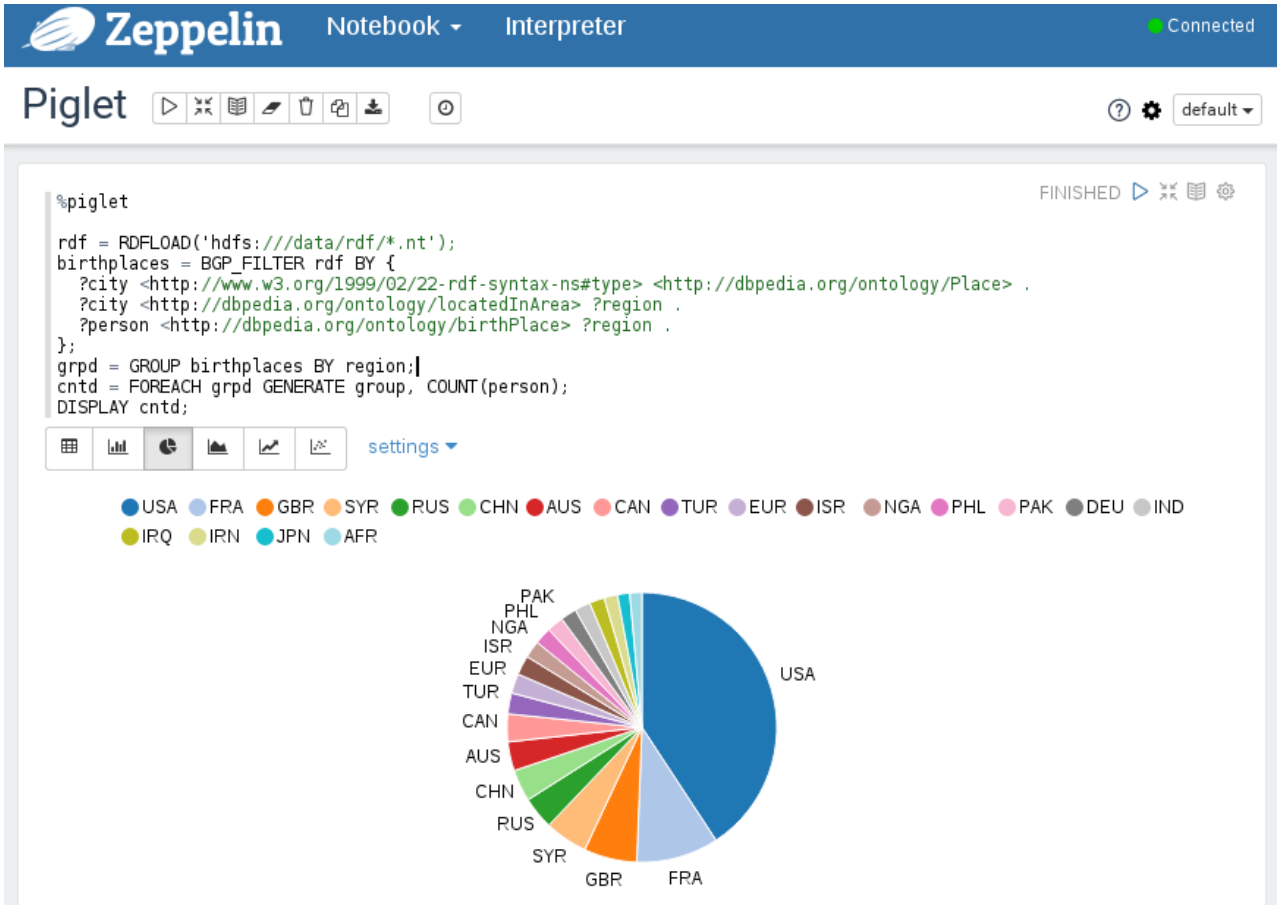

Figure 2: A screenshot of Zeppelin with a Piglet script that contains a `BGP_FILTER` mixed with traditional operators.

Audience Interaction.

Our prepared scripts can be changed by the demo audience live either through the Zeppelin interface, in the script files, or can be entered in our REPL shell.

On the command line, backends can be switched by just setting a command line argument, whereas in Zeppelin, users have to change the target backend in the interpreter settings. For convenience, we will setup an individual Piglet interpreter for each supported backend, so that users can easily switch between them by simply changing the magic word for a paragraph, e.g., `%piglet_spark`, `%piglet_flink`. As result visualization is completely performed by Zeppelin, users can select the appropriate chart type using the provided buttons.

5. SUMMARY & FUTURE WORK

Piglet is a compiler and code generator for the Pig Latin script language with the goal to unify and simplify data analytics. Piglet supports both batch and stream processing and generates code for various platforms like Spark, Flink, and Storm. Its support for BGP filters allows easy integration and querying of Linked Open Data, both from local files as well as from remote endpoints. We provide a Zeppelin integration for interactive data analytics and visualization.

Our ongoing work focuses on optimization: When scripts are executed repeatedly, expensive operations are also executed every time, although their result has not changed since the last run. We collect runtime information of the generated program, try to identify such expensive tasks and materialize their intermediate results if this promises to speed-up subsequent runs. Furthermore, in future work we are going to investigate how to support iterations in Pig scripts.

Acknowledgements.

This work was partially funded by the German Research Foundation (DFG) under grant no. SA782/22

6. REFERENCES

[1] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: a stream data management benchmark. In *PVLDB*, pages 480–491, 2004.

[2] S. Hagedorn, K. Hose, and K.-U. Sattler. SPARQling Pig - Processing Linked Data with Pig Latin. In *BTW*, March 2015.

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

[4] O. Saleh and K.-U. Sattler. Debs grand challenge: The pipeflow approach. In *DEBS*, pages 326–327, New York, NY, USA, 2015. ACM.