QryGraph: A Graphical Tool for Big Data Analytics

Sanny Schmid, Ilias Gerostathopoulos, Christian Prehofer
Fakultät für Informatik
Technische Universität München
Munich, Germany
{schmidsa, gerostat, prehofer}@in.tum.de

***Abstract*—The advent of Big Data has created a rich set of diverse languages and tools for data manipulation and analytics within the Hadoop ecosystem. Pig has a prominent role within this ecosystem as a scripting layer—a convenient way to create analytics jobs that are issued for batch processing in a Hadoop cluster. In order to leverage the benefits of graphical domain specific languages, namely intuitive visual design and inspection, we implemented a web-based graphical tool called QryGraph that complements Pig in various ways. First, it allows a user to create Pig queries in a graphical editor and check their syntax. Second, it provides an administrative interface for managing the execution and overall lifecycle of Pig queries. Finally, it will allow for debugging by running queries on test data sets and for creating user-defined query sub-graphs that can be reused across different Pig queries.**

Keywords—Big Data; tool support; Pig language

I. INTRODUCTION

Recent advances in Big Data and Internet of Things technologies have created new disruptive possibilities related to new insight that can be obtained by analyzing the large quantities of sensed data [1]–[3]. More and more enterprises are looking into ways to build value-added services on top of Big Data analytics infrastructures.

In this context, Apache Hadoop has emerged as the de facto standard in Big Data analytics. Hadoop is an open-source ecosystem comprised of a multitude of languages and tools for data manipulation and analytics, e.g. MapReduce, HDFS, Hive, Pig, HBase, Kafka, Storm, Spark, etc. It supports processing of both static data—batch mode—and streams of incoming data in a real-time fashion—stream mode. At the same time, it comprises tools to address the needs of both developers, administrators, and data scientists and analysts. The main advantage of Hadoop is that it provides a fault-tolerant infrastructure that can easily scale to several thousand nodes in a single cluster and to several petabytes of data. The data stored in a Hadoop cluster are analyzed in batch mode by developing application-specific "mappers" and "reducers", i.e. functions that work on key/value pairs: mappers operate on input data (e.g. a large file residing in the Hadoop Distributed File System—HDFS [4]); reducers combine and summarize the results of the mappers [5]. Once the application-specific mapper and the reducer is implemented in Java or another Hadoop-compliant implementation language, they are bundled together into a single analytics job (also called a *map-reduce program*) that is issued to the Hadoop cluster. This triggers the parallel execution of several mapper and reducer tasks; the end result is then written back to the HDFS.

Within the Hadoop ecosystem, the *Pig* platform and language (also called Pig Latin) provide a convenient way to create analytics jobs compared to the manual implementation of mappers and reducers in a general-purpose programming language such as Java [6], [7]. Being a domain-specific language, Pig is essentially a thin layer over Hadoop that allows for specifying succinct scripts for analytics jobs that load data, apply transformations on the data, and store the final results.

Pig has been significantly enhanced since its inception at Yahoo! Research in 2006 to include several advanced features such as error handling and type checking, together with several performance improvements [7]. We nevertheless believe that there is still room for improvement, in particular in supporting the users of Pig (typically developers) in the creation, validation and evolution of complex analytics jobs.

To this end, we implemented a tool for Big Data analytics called QryGraph. Our tool supports Pig users in creating new analytics jobs that comply with the Pig language via an intuitive graphical editor. The editor allows for both visual design and validation via visual inspection. It provides an administrative interface for managing the execution and overall lifecycle of analytics jobs, including testing and debugging features. Finally, it will provide enhanced debugging features, such as running queries on test data sets, and will allow creating user-defined query sub-graphs that can be reused across different Pig queries.

In this artifact paper, we present the main functionalities and design goals of QryGraph, together with its technical architecture and extensibility mechanisms. We detail on the ongoing implementation and articulate our long-term plans. Our initial experience with using the tool indicates that it could become a vehicle for enhancing the understanding, creation, and evolution of Big Data analytics.

The rest of the text is structured as follows. Section II presents the running example and its solution in Pig. Section III presents our tool and discusses the main features and usage scenarios. Section IV reflects on our experience so far with QryGraph, and its current limitations. Finally, Section V compares our approach in supporting Big Data analytics to the state of the art and practice, and Section VI concludes and presents our future work plan.

978-1-5090-1897-0/16/$31.00 ©2016 IEEE

car_id	speed	longitude	latitude	timestamp
1	20	48.137	11.577	55
2	7	48.141	11.532	66
3	0	48.187	11.521	77

PL_id	available	longitude	latitude	timestamp
1	1	48.198	11.578	31
2	1	48.126	11.512	44
3	0	48.132	11.501	55

Fig. 1. Exemplary data sets in the running example.

II. Running Example and Background

A. Running Example

In our running example, the administration of the city of Munich aims to implement a new pricing system for parking lots (PLs). The price of each PL should be calculated based on the number of cars driving in its vicinity. The prices should be updated in a periodic fashion to ensure a fair pricing allocation.

To be able to implement the above mechanism, the city has access to data collected from user cars and PLs belonging to city-run parking stations. For each car, its position and speed are periodically monitored; for each PL its position and availability. Data is stored locally for the course of a full day and submitted for analysis as a full day batch. For illustration, the data sets could look like the ones depicted in Fig. 1.

To get the necessary information of all driving cars near an available PL, the cars data set is filtered to only consider cars with a speed higher than, e.g., 5 km per hour (which indicates that they are not parked). This data set is combined with the PL data sets in order to find cars driving near a PL. This creates a list of PLs and number of cars driving near them. This information is then used to determine the price of a PL.

COMMAND	DESCRIPTION	EXAMPLE
LOAD	Used to load data from the HDFS	A = LOAD 'sample.csv' USING PigStorage(',') AS (name:chararray, age:int, gpa:float);
FOREACH	Run a command for each data row	B = FOREACH A GENERATE name;
GROUP	Groups data into tuples	C = GROUP B BY name;
JOIN	Performs a join on two data sets	N = JOIN A BY name, K BY name;
DISTINCT	Removes duplicate tuples in a relation.	C = DISTINCT B;
CROSS	Creates the cross product of two data sets	F = CROSS C, E;
FILTER	Filters a data set based on an expression	H = FILTER G BY <Boolean expr>

Fig. 2. Popular Pig commands.

B. Background: Pig

Pig Latin [6] is a procedural query language for very large data sets. It offers an alternative to the well-known SQL standard for querying HDFS and is designed to work with the Hadoop infrastructure. Instead of one single relational query, a Pig query consists of a directed acyclic graph of nodes that can be seen as an execution plan. Within this graph, each node describes one step that is needed to execute the query—from loading the data, to the final output. Pig optimizes a query on the fly before sending it to the Hadoop cluster [7] and reaches a performance that is comparable with native Hadoop implementations. It is widely used as an alternative query language and is also included into popular Hadoop distributions like Hortonworks [8].

The Pig language consists of a broad set of commands. The result of each command is assigned to a variable that can be used by other commands in the query. A number of popular Pig commands are depicted in Fig. 2. An important difference to

```
DEFINE Distance
       datafu.pig.geo.HaversineDistInMiles();

-- Create a list of PLs and their position
A = LOAD 'slots.csv' USING PigStorage(',') as
       (PL_ID:int, AVAILABLE:int,
       LONGITUDE:double, LATITUDE:double,
       TIMESTAMP:long);
B = FOREACH A
       GENERATE PL_ID, LONGITUDE, LATITUDE;
C = DISTINCT B;

-- Create a list of all cars that were driving
D = LOAD 'cars.csv' USING PigStorage(',') as
       (CAR_ID:int, SPEED:int, LONGITUDE:double,
       LATITUDE:double, TIMESTAMP:long);
E = FILTER D BY speed > 5.0;

-- Join the data by GPS distance
F = CROSS C, E;
G = FOREACH F GENERATE *, Distance(C::LATITUDE,
       C::LONGITUDE, E::LATITUDE, E::LONGITUDE)
       as DISTANCE;
H = FILTER G BY DISTANCE < 5.0;

-- Count the amount of cars for each PL
I = GROUP H BY PL_ID;
J = FOREACH I {distCars = DISTINCT H.CAR_ID;
       GENERATE $0, COUNT(distCars);};
```

Fig. 3. Possible Pig query for the running example.

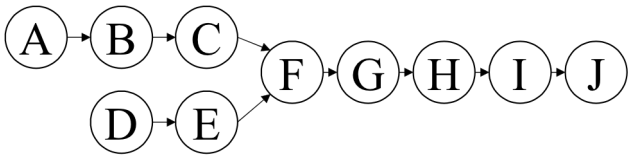

Fig. 4. Abstract graph of the Pig query of the running example.

SQL is that Pig allows special data formats like TUPLE, BAG or MAP. A GROUP command, e.g., puts all elements into a BAG that is associated with the group key. These additional formats can also be used as nested data structures, e.g. a bag within a bag. Additional to the native Pig commands, there is the possibility to use *user defined functions* (UDFs) that implement custom behavior that might require a complex computation. These can be written in Java or Python and included into the Pig query as seen in Fig. 3 (line 1).

For illustration, the running example could be modeled by the Pig query depicted in Fig. 3—the corresponding abstract query graph is depicted in Fig. 4.

III. Tool Description

QryGraph is a tool to simplify the creation, maintenance, evolution, and management of Big Data analytics jobs. An analytics job in QryGraph corresponds to a Pig language query. A Pig query is specified via the use of a graphical editor (the heart of QryGraph), which represents a query by its abstract graph.

The user is able to specify Pig queries via modeling the data flow between nodes corresponding to Pig language commands (see Fig. 2). Each command corresponds to a node in the graph; nodes' input and outputs are connected to create the query graph. The user receives immediate feedback once a type error is introduced in the design process (e.g. when trying to connect a node's output of type A to another node's input of type B). Once a correct query graph is created, the tool automatically compiles it down to valid Pig code and presents it to the user for validation. The generated query can then be issued to a Hadoop cluster.

Apart from the graphical editor, QryGraph provides also an interface for keeping track of all created queries, for issuing in an on-demand or periodic schedule basis, and for notifying the user for the termination of issued queries and presenting the query results.

In the following, we detail on the main design goals of QryGraph, its main usage scenarios, as well as its technical architecture and implementation.

A. Main Design Goals

1) Easy to use

The tool should reduce the cognitive barrier in understanding and creating complex Pig queries. For this, it needs an intuitive and easy to use user interface that helps the novice designer in creating a query (e.g., by offering the possible fields to filter by in a FILTER node). At the same time, it should offer full flexibility to advanced Pig users, who might need to visually inspect or even edit the generated Pig code.

The tool should also provide an ergonomic interface for managing created queries and configuring their scheduling policies.

2) Quick feedback

The tool should offer quick, preferably immediate, feedback when designing a query, so that the designer can resolve type errors early on.

The tool should also provide a "test run" of a query, i.e., execution of the query on a small set of fabricated data instead of production data. This would offer the possibility of semantic checks and validation, answering the question "Does the query actually perform what it is supposed to?"

3) Reuse support

The tool should offer the user the possibility to extract common patterns used across several queries and reuse them in new queries as "composite nodes". One such example could be a composite node that joins GPS position data from two inputs based on a user defined distance function.

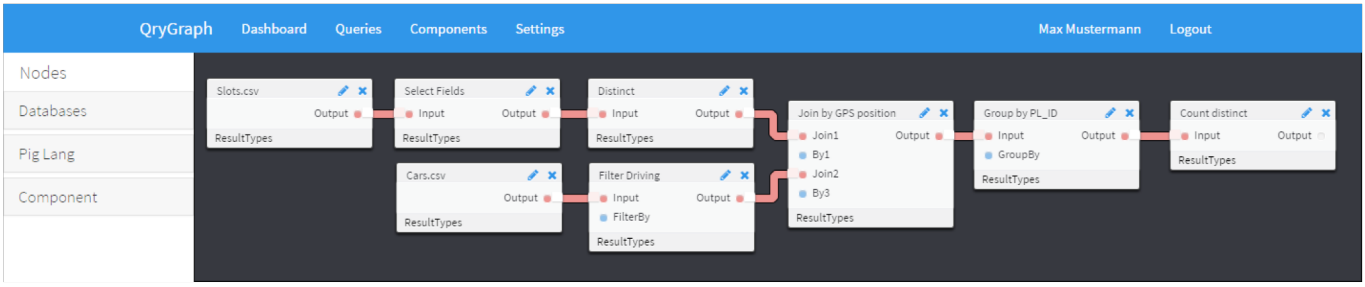

Fig. 5. QryGraph graphical editor.

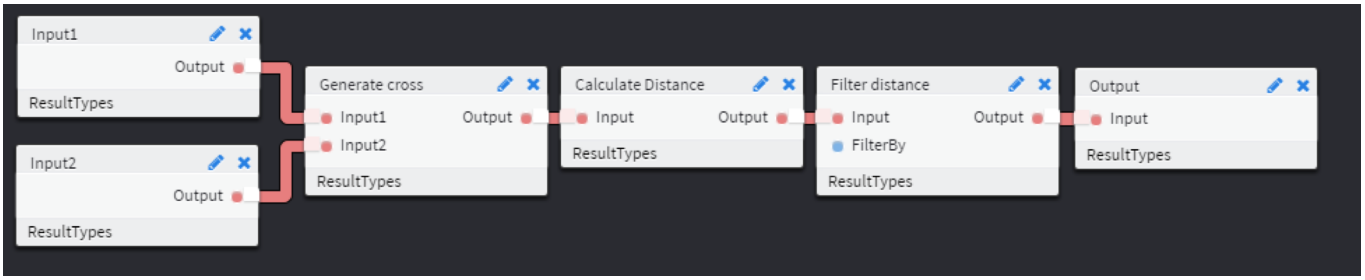

Fig. 6. Subgraph of "Join by GPS position" node.

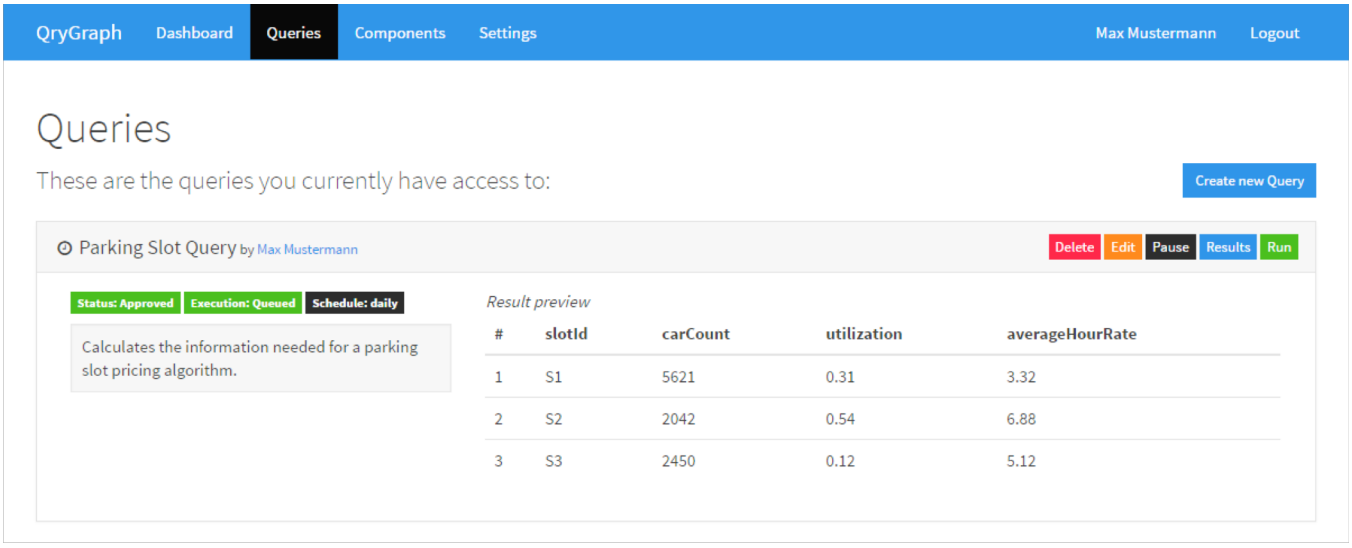

Fig. 7. Query management interface.

4) Easy to setup

The tool should be setup with minimum effort from the user. This will allow prospective users to experiment with the tool and consider contributing by extending its functionalities.

B. Usage

QryGraph offers two main functionalities to its users: query design in a graphical editor and query administration and lifecycle management.

1) Query design

When a user wants to create a new query or edit an existing one, he/she uses the graphical editor. The editor features a command menu on the left side and a big pane to create the query graph on the right side (Fig. 5). Here the user can:

- select the data sources (e.g. files in CSV format) of the query and add them to the graph;
- add native Pig functions (e.g., FILTER, GROUP, JOIN) as nodes to the graph;
- edit the configuration and parameters of the nodes;
- plug nodes together via connecting input ports to output ports;
- automatically get instant feedback on possible type errors after every change;
- inspect the Pig code that is generated on the fly from an error-free graph.

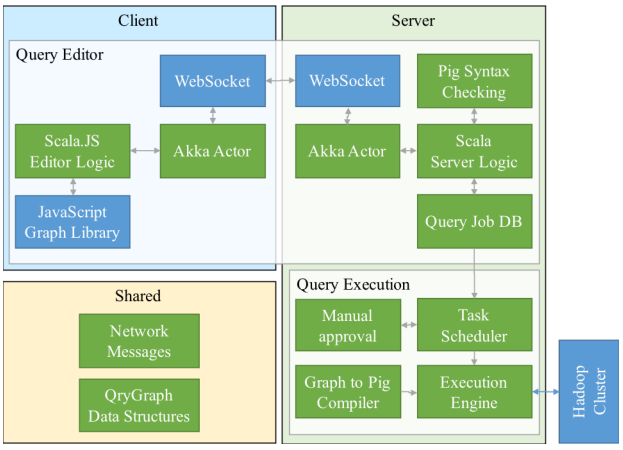

Fig. 8. QryGraph architecture – custom code (green) / libraries (blue).

2) Query administration and management

When a user needs to obtain an overview of the created queries, manage their triggering policies, and view their sample results he/she uses the administrative interface of QryGraph (Fig. 7). Here the tool offers the user to:

- review execution statistics on a dashboard;
- list all queries the user has created;
- pause and suspend query execution;
- change the execution schedule of a query;
- check the approval status of a query (each query has to be approved at the server-side in order to run on production data—see Section III.C);
- view the results of a query.

C. Technical Architecture and Implementation

QryGraph has been implemented as an open-source web application. Its latest version is accessible at https://github.com/Starofall/QryGraph.

The tool follows a client-server architecture (Fig. 8) and is mainly written in Scala. The client component is built with Scala.js[1], a library that compiles Scala into JavaScript. The

[1] https://www.scala-js.org

server is built with the Play Framework[2]; the dynamic server-client communication is done using the actor-oriented Akka Library[3] and Akka.js[4] with underlying web sockets. This connection allows to update the client on compilation results without using any long polling technique. Using the abstraction layer provided by Akka actors, the client and server communicate via typed messages sent serialized through the socket. The message classes and data structures are shared by the server and client Scala code. This simplifies development and debugging since a node instance has the same behavior on both the client and the server and no manual data parsing on the server or the client is required.

In summary, the internal workings of the graphical editor are the following: Once a user edits an element of a graph, the graph object is serialized and sent over the network to the server. Then the server performs syntax checking. If the graph is syntactically correct, Pig code is generated out of it and sent back to the user for visual inspection. Once the user wants to issue the query to the Hadoop cluster, the execution request is send to the server where it is added to the queue of queries that need manual approval by the cluster administrator (an optional step in the process). Then, the query is issued to the Hadoop cluster. Once the results are computed, they are sent back to the user via the same actor-based communication.

IV. Discussion and Work in Progress

QryGraph is a project under active development. In the initial stages documented in this paper, we were mainly focusing on implementing the programming abstractions and overall infrastructure that would allow to speed up feature development. For instance, we now have a very robust actor-based client-server communication that allows for seamless development.

In the following, we reflect on the main architectural decisions and on the features under implementation.

A. Web-Based

The QryGraph client has been implemented as a web-based application. One of the primary drivers for doing so was the ease of use, as users are typically familiar with browser environments. At the same time, this allows the developers to use many off-the-shelf libraries that are available for JavaScript and CSS (e.g. Twitter Bootstrap).

The web-based environment is also making the tool setup easy: it is only required to deploy a JVM-based application on a server. Then any user with a modern browser is able to start using the tool and create queries.

B. Seamless Client-Server Development

In order to leverage on the domain-specific modeling capabilities and the type system of Scala, we opted for using Scala.js to generate most of the JavaScript code, in particular the parts related to the representation of queries on the client side. Using a well-known Scala library for actors, Akka, and its Scala.js counterpart, Akka.js, allows developers to work with the same actor-based abstractions on both client and server side. The communication is conveniently abstracted into typed messages sent between Scala actors over a web sockets connection.

C. Enhanced Testing and Debugging – In Progress

At its current state, the graphical editor gives instant feedback to the user, as changes are being made on the graph, regarding syntax errors. This is performed at the server side by type checking (e.g. checking that connected inputs and outputs are of the same type, GROUP operators operate on attributes given as inputs, etc.). This offers a mechanism to spot errors early on and refactor the query.

An additional mechanism (under development) is to allow sample runs of an error-free query for debugging reasons. Running a query on the production data can be a lengthy process taking minutes or hours to complete. However, for debugging purposes, a much quicker response has to be provided to the user. To this end, a test data set has to be provided against which test runs can be issued. Providing such a test data set is far from easy and typically has to be tailored to the particular user program at hand [6]. This data set has to be considerably smaller than the production data set, yet still realistic; it can be used for two types of tests:

- *Fit-for-purpose validation*, i.e. checking whether the query has the intended effect;
- *Performance* testing: When performing multiple queries to the same test data set or test data sets of approx. the same size, the execution times of past runs can be used as an indicator of a low-performing query. Such analysis is based on the fact that the relative difference in the execution times of queries is of importance, and not the actual values.

D. Component System – In Progress

One of the advantages of the graphical editor is that it makes the data-flow architecture of a query explicit, which improves program comprehension and maintainability. To enable reuse of query fragments (subgraphs) that are common across different queries, we are working on providing a mechanism that allows the creation of components with well-defined inputs and outputs out of query subgraphs. These components would then be used as regular nodes in the graphical editor, similar to black-box component composition.

An example of such a component could be a subgraph that joins GPS position data from two inputs based on a user defined distance function. The creation of these component will be done via the main graphical editor, as illustrated in Fig. 6.

V. Related Work

Due to the recent advancements in Big Data infrastructures and tools and in the Hadoop ecosystem in particular, many different tools and products have been proposed. We focus here on tools that offer graphical interfaces for viewing or specifying Big Data analytics jobs, thus are directly comparable to QryGraph.

The open-source project PigPen [6], [9] is an extension to the Eclipse platform that allows the user to specify Pig queries in a

[2] https://www.playframework.com
[3] http://akka.io
[4] https://github.com/unicredit/akka.js

textual editor and then inspect the corresponding query graph, which is updated on the fly. However, editing the graph is not supported. PigPen also offers error checking, and running a query in a "sandbox" data set for debugging. We intend to reuse their approach in the creation of our test data set infrastructure.

A mature open-source solution, also based in Eclipse, is Talend Open Studio for Big Data [10]. It allows the specification of Big Data analytics jobs in a graphical interface. Compared with this tool, QryGraph is more lightweight and requires less upfront effort for setting up and getting started with the tool.

When looking at closed-source solutions, Tableau Desktop [11] is a tool that offers an easy-to-use user interface for specifying and running Big Data analytics. Instead of supporting the generation of a specific query, this tool focusses on data visualization. Pentaho [12] is offering several closed source products that also help the user define a data flow using a graphical interface. They support multiple database configurations.

VI. Conclusions

In this artifact paper, we presented our ongoing implementation and long-term plan for a new tool for simplifying the creation and management of Big Data analytics jobs. QryGraph focuses on the popular scripting language Pig and offers visual representation and editing of Pig queries. It allows for rapid prototyping by on-the-fly compilation and syntax checking, and promotes reuse by allowing for specifying user-defined query subgraphs.

One of the features that we would like to include in QryGraph in the future is the possibility for importing existing Pig queries into the graphical editor. This will allow for creating a library of example queries for educational purposes and of subqueries that can be reused.

Finally, we would like to support cooperative query editing. The existing implementation based on web sockets already provides the base upon which several users can work locally and communicate with the server with independent query updates.

Acknowledgment

This work is part of the TUM Living Lab Connected Mobility project and has been funded by the Bayerisches Staatsministerium für Wirtschaft und Medien, Energie und Technologie.

References

[1] J. Needham, *Disruptive Possibilities: How Big Data Changes Everything*. O'Reilly Media, 2013.

[2] E. Dumbill, *Planning for Big Data*. O'Reilly Media, 2012.

[3] S. Srinivasa and V. Bhatnagar, Eds., *Big Data Analytics*, vol. 7678. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Washington, DC, USA, 2010, pp. 1–10.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM - 50th Anniv. Issue*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[6] 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*, New York, NY, USA, 2008, pp. 1099–1110.

[7] 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.

[8] "Hortonworks," 01-May-2016. [Online]. Available: http://hortonworks.com/.

[9] "PigPen Wiki," 01-May-2016. [Online]. Available: https://wiki.apache.org/pig/PigPen.

[10] "Talend Open Studio for Big Data," 01-May-2016. [Online]. Available: https://de.talend.com/download/talend-open-studio.

[11] "Tableau Desktop," 01-May-2016. [Online]. Available: http://www.tableau.com/de-de/products/desktop.

[12] "Pentaho," 01-May-2016. [Online]. Available: http://www.pentaho.com/product/data-integration.