RheemStudio: Cross-Platform Data Analytics Made Easy

Ji Lucas, Yasser Idris, Bertty Contreras-Rojas, Jorge-Arnulfo Quiané-Ruiz, Sanjay Chawla

Qatar Computing Research Institute, HBKU
Doha, Qatar
http://da.qcri.org/rheem/

***Abstract*—Many of today's applications need several data processing platforms for complex analytics. Thus, recent systems have taken steps towards supporting cross-platform data analytics. Yet, current cross-platform systems lack of ease-of-use, which is crucial for their adoption. This demo presents RHEEMStudio, a visual IDE on top of RHEEM. It allows users to easily specify their cross-platform data analytic tasks. In this demo, we will demonstrate five main features of RHEEMStudio: drag-and-drop, declarative, interactive, and customized specification of data analytic tasks as well as easy monitoring of tasks. With this in mind, we will consider two real use cases, one from the machine learning world and the second one based on data discovery. During all the demo, the audience will be able to take part and create their own data analytic tasks too.**

I. Need for Easy Cross-Platform Data Analytics

More and more users and organizations seek to perform data analytics over several data processing platforms (platforms, for short). For example, (i) several data warehouse applications need to move aggregated, projected, or specific data from Hadoop (or Spark) into a relational database for further analysis [1], [2], (ii) IBM reported that North York hospital needs to process 50 diverse datasets, which are on a dozen different internal platforms [3], (iii) business intelligence typically requires a single data analytic pipeline composed of different platforms [4], and (iv) latest research results have shown that using multiple platforms for ML applications improves performance significantly [5], [6]. These are few examples of applications that need data analytics over multiple platforms.

To cope with these new requirements, several systems have appeared to help users to easily combine several platforms [7]–[11]. Yet, they all need quite good expertise from users to decide which platforms to use for the query at hand. Thus, developing cross-platform applications is not an easy task as it still requires being intimate with the intricacies of the different platforms to achieve high efficiency.

The research community has recognized the need for a systematic solution that enables *efficient* cross-platform data analytics [12]–[14]. Recent systems have taken steps towards this goal [4], [15], [16]. Despite all these efforts and although *ease-of-use* is a very important feature of data analytic systems, it still remains a desired feature [17]. As a result, using any of these systems is more of an odyssey rather than a pleasant journey. Clearly, ease-of-use is crucial for the adoption of these systems. However, achieving ease-of-use in cross-platform settings is quite challenging.

We have recently introduced RHEEM [12], [18], an open source system[1] for cross-platform data analytics with the goal of addressing these two problems: (i) efficiency and (ii) easy-to-use cross-platform data analytics. It decouples applications from the platforms and allows them to efficiently run over one or multiple platforms. We have shown the benefits of running machine learning [5] and data cleaning [19] tasks over RHEEM. We have also demonstrated the efficiency and flexibility features in [18].

In this demo, we will demonstrate the ease-of-use feature of RHEEM, powered by RHEEMStudio (RHEEM's visual IDE). RHEEMStudio helps developers to build their applications on top of RHEEM in an easy and intuitive manner. It features (i) drag-and-drop plan construction, (ii) a declarative language for users to declare their data analytics tasks, which we call RHEEM*Latin* (inspired from PigLatin [20]), (iii) a dashboard for displaying RHEEM's internal details on the plan generation and allowing on-the-fly interaction with the developer, and (iv) a monitor for keeping track of the progress of data analytic tasks. To better demonstrate this ease-of-use feature, we will consider two real use cases: one from the machine learning world and the second one form the data discovery area. The audience will be able to take part of the demo by creating their own data analytic tasks too.

II. RHEEM Ecosystem

Before delving into the RHEEMStudio, let us briefly outline RHEEM [12], [18] and its ecosystem.

RHEEM is our open source cross-platform system that enables data processing over multiple data processing platforms. It acts as a unifying abstraction layer in an environment with multiple co-existing processing platforms. Data processing applications are written using RHEEM's interface and, thus, do not need to select any actual processing platform.

As depicted in Figure 1, RHEEM decouples applications from the data processing platforms. Applications do not submit tasks directly to a data processing platform. Instead, they assign their tasks to RHEEM, which then decides where to

[1] https://github.com/rheem-ecosystem/rheem

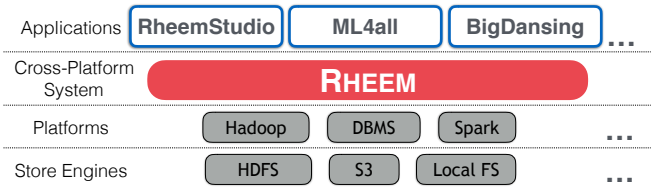

Fig. 1. RHEEM ecosystem.

execute them. Overall, the whole execution of a data analytic task is composed of four phases: (1) *task submission*, (2) *task optimization*, (3) *task execution*, and (4) *task monitoring*.

(1) *Task submission.* Applications provide a RHEEM plan, which is a directed acyclic data flow graph: the vertices are RHEEM *operators* and the edges represent data flows among the operators. RHEEM operators are platform-agnostic and define a particular kind of data transformation over their input data quanta. Specific operators, such as the Loop, further accept feedback edges, which allows to express iterative data flows. Also, the behavior of most RHEEM operators can be refined by *user-defined functions (UDFs)*, which renders RHEEM plans highly expressive.

(2) *Task optimization.* Given a RHEEM plan as input, RHEEM produces an *execution plan* by selecting one or more of the available data processing platforms. The execution plan is again a data flow graph, but with two differences from a RHEEM plan: (i) the vertices are now replaced with *execution operators*, i.e., platform-specific operators; and (ii) an execution plan may comprise additional execution operators, e.g., for data movement. For this purpose, RHEEM uses a flexible m-to-n mappings from RHEEM operators to execution operators. This means it can map subplans of RHEEM operators to subplans of RHEEM or execution operators. Those mappings eventually span the search space of possible cross-platform executions plans from which RHEEM efficiently determines the most promising alternative plan according to its cost model. During the plan selection, it particularly reasons on complex data movement paths between platforms and takes the incurring data movement costs into account.

(3) *Task execution.* Next, the system runs and orchestrates the execution of the generated execution plan on the selected processing platforms. It first divides an execution plan into *stages*. A stage is a subplan where (i) all its execution operators are from the same platform; (ii) at the end of its execution, platforms need to give back the execution control to the executor; and (iii) at the end, an output dataset is generated. The system parallelizes the execution of independent stages. It is worth noting that RHEEM has a decoupled platform-specific driver for each available processing platform. It is each of these drivers that internally take care of running the execution plan. This design choice makes our system very easy to extend to new platforms or update an existing platform driver.

(4) *Task monitoring.* During execution, the system checks the health of the execution plan. In particular, it collects light-weight execution statistics for the given plan, such as data cardinalities and operator execution times. Furthermore, it is also aware of lazy execution strategies used by the underlying platforms and attributes measured execution time correctly to operators. RHEEM uses these statistics to improve its cost model and re-optimize ongoing execution plans in case of poor cardinality estimates.

The **RHEEM ecosystem** is currently composed of three applications for data analytics:

(1) ML4ALL is a machine learning application that frees users from the burden of algorithm selection [5]. ML4ALL leverages RHEEM to automatically switch between Spark and JVM platforms for enhancing performance.

(2) BIGDANSING is a rule-based data cleaning system for big data [19]. It leverages RHEEM to scale out data detection and repair to unprecedented scales.

(3) TRUTHFINDER is a scalable truth discovery system. It allows users to discover the truth from a collection of heterogeneous sources of information that have conflicting claims on a particular fact. It uses RHEEM to seamlessly scale truth discovery algorithms.

Currently, we are building a number of other applications on top of RHEEM, such as an exploratory database and a stream data processing system. Apart from the applications, RHEEM provides a Java, Spark, and REST APIs as well as a declarative language and a visual IDE (the RHEEMStudio).

RHEEM*Studio* is a web application that is built on JavaScript using the MEAN stack[2] and on top of the RHEEM REST API. It exposes a GUI that allows users to develop data analytic tasks in both *drag-and-drop* and *declarative* fashion. For example, the left-side panel of Figure 2 shows the operators that users can drag-and-drop on the middle panel. Similarly, user can write RHEEM*Latin* queries on the top-right panel. RHEEMStudio comes with a backend that is not only responsible of connecting RHEEMStudio with the REST API, but also of automatically generating all the content to be displayed at the frontend. This means that all RHEEM operators are automatically extracted from the RHEEM code base and displayed in the frontend of RHEEMStudio. Note that the REST API is, in turn, responsible for compiling and attaching the UDFs to RHEEM plans, constructing a valid RHEEM plan, and triggering the execution. For all data operations, RHEEMStudio uses Mongodb to keep, among others, the user profiles, roles, and RHEEM plans.

III. RHEEMSTUDIO AT WORK

In this demo, we will use real-life datasets to give users the opportunity to experience the features provided by RHEEMStudio. Notice that this demonstration significantly differs from the one presented in [18], which demonstrates the flexibility and efficiency features of RHEEM. Instead, we will demonstrate RHEEMStudio by showing the audience how easily they can specify and monitor their own cross-platform data processing pipelines. We will use two different use cases:

[2]https://en.wikipedia.org/wiki/MEAN_(software_bundle)

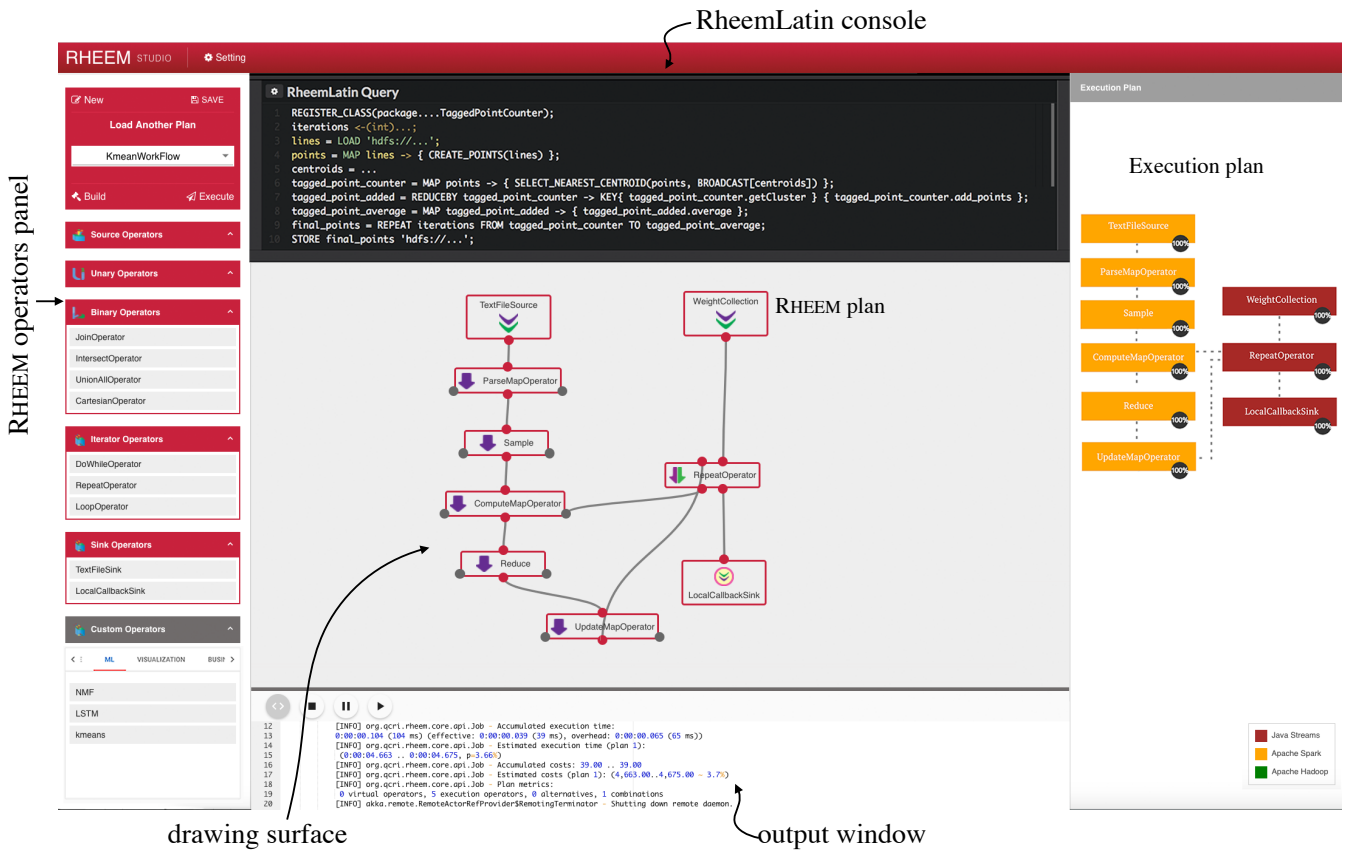

Fig. 2. K-means in the RHEEMStudio.

Machine learning (ML) has became an important piece in data analytics. However, developing scalable ML algorithms is typically a tedious task, which makes ML a luxurious asset in many organizations. Thus, achieving ease-of-use in ML is a major lacunae that needs to be urgently addressed. In this demo, we will show how users can easily develop, execute, and monitor scalable ML tasks with little effort.

Data discovery is an important piece in data lakes, where users need to find relevant data from datasets spread across several data stores. Unfortunately, current solutions are not designed for cross-platform settings and hence are difficult to employ in such settings. The research community has recently recognized that performing data discovery over multiple data stores (data lakes) is one of the important pieces of data science [21]. In this demo, we will show how users can easily perform customized data discovery over multiple data stores in a transparent manner.

In particular, having these two use cases in mind, we will showcase five main features of the RHEEMStudio:

(1) *Drag-and-drop* RHEEM *plan generation*. Our main goal is to show how easy is to specify and run tasks in RHEEMStudio. For this, we will start by showing three pre-defined plans: two popular ML tasks (k-means and stochastic gradient descent) and one data discovery task (keyword search). We will walk the audience through these three plans so that they start getting familiarized with the GUI. The middle panel of Figure 2 shows the k-means plan in RHEEMStudio. Once familiarized, we will invite the audience to specify their own data analytic tasks using the drag-and-drop interface (left panel in Figure 2).

(2) *Declarative* RHEEM *plan generation*. We will also show to the audience how the three previously presented plans can be specified declaratively. For this, we will enable the RHEEM*Latin* console embedded into the RHEEMStudio (top right-side panel) and load the respective queries. We will then ask the audience to write their analytic tasks using RHEEM*Latin*. It is worth noting that the audience will be able to see the generated RHEEM plans from their queries.

(3) RHEEM *plan customization*. Furthermore, we will walk the audience through the different ways they can easily inject their own logics, such as how to prepare data. For example, Figure 3 shows the data preparation code for the Map operator in the k-means plan of Figure 2. For each RHEEM plan, we will invite the audience to select one of the operators to revise and ask them to inject their own logics. It is worth noting that users mainly have to inject their logics in lines 14 and 15 of Figure 3. After revision, RHEEMStudio validates the new provided code. If the code is syntactically correct, the audience can submit their changes by pressing the Save button and run the plan by pressing the Execute button.

(4) *Interactive* RHEEM *plan generation*. Especially, in this

Fig. 3. Code panel for the Map operator in the k-means plan.

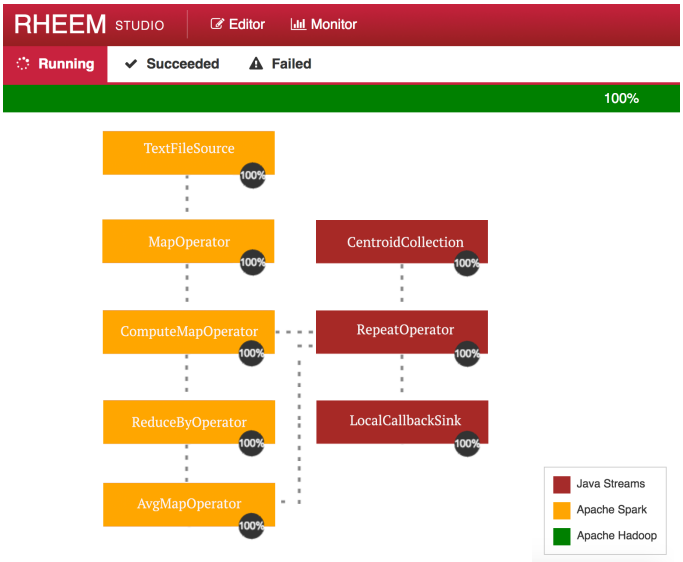

Fig. 4. Monitoring the execution of the K-means plan.

demo, we will show how users can interact with RHEEMStudio in order to enhance their data analytic tasks. For this, RHEEMStudio also displays the execution plan for any given RHEEM plan so that users can better evaluate their queries and plans (bottom right-side panel of Figure 2). By having this global view of a data analytic task, users can write better queries and RHEEM plans. As an example, we will invite the audience to provide hints to RHEEM in order to guide the execution plan generation. For example, we will show the audience how to change the data processing platforms on which a plan will be executed.

(5) *Cross-platform plan monitoring*. Last but not least, we will show how users can monitor the execution of their analytic tasks (Figure 4). RHEEMStudio not only displays the execution progress for each operator in an execution plan, but also displays the intermediate data being produced at different parts of the plan. This allows users to find possible bugs in their plans, which are usually very hard to spot.

Setup. We will run the demo on a single machine as well as on a 4-machines cluster at QCRI. We will use the latest version of RHEEMStudio and RHEEM. We will consider Java, Spark, and Postgres as underlying processing platforms.

REFERENCES

[1] D. J. DeWitt, A. Halverson, R. V. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, and J. Gramling, "Split Query Processing in Polybase," in *SIGMOD*, 2013, pp. 1255–1266.

[2] S. Shankar, A. Choi, and J.-P. Dijcks, "Integrating Hadoop Data with Oracle Parallel Processing," Oracle White Paper, http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-integrating-hadoop-data-with-or-130063.pdf, 2010.

[3] IBM, "Data-driven healthcare organizations use big data analytics for big gains," White paper, http://goo.gl/AFIHpk.

[4] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal, "Optimizing Analytic Data Flows for Multiple Execution Engines," in *SIGMOD*, 2012, pp. 829–840.

[5] Z. Kaoudi, J. Quiané-Ruiz, S. Thirumuruganathan, S. Chawla, and D. Agrawal, "A Cost-based Optimizer for Gradient Descent Optimization," in *SIGMOD*, 2017, pp. 977–992.

[6] M. Boehm, M. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. Reiss, P. Sen, A. Surve, and S. Tatikonda, "SystemML: Declarative Machine Learning on Spark," *PVLDB*, vol. 9, no. 13, pp. 1425–1436, 2016.

[7] J. Duggan *et al.*, "The BigDAWG polystore system," *ACM SIGMOD Record*, vol. 44, no. 2, pp. 11–16, 2015.

[8] "Apache Drill," https://drill.apache.org.

[9] "Apache Flume," https://flume.apache.org/index.html.

[10] "Luigi Project," https://github.com/spotify/luigi.

[11] "PrestoDB Project," https://prestodb.io.

[12] D. Agrawal, S. Chawla, A. K. Elmagarmid, Z. Kaoudi, M. Ouzzani, P. Papotti, J.-A. Quianuiz, N. Tang, and M. J. Zaki, "Road to Freedom in Big Data Analytics," in *EDBT*, 2016, pp. 479–484.

[13] M. Stonebraker, "The Case for Polystores," ACM SIGMOD Blog. [Online]. Available: \url{http://wp.sigmod.org/?p=1629}

[14] D. Tsoumakos and C. Mantas, "The Case for Multi-Engine Data Analytics," in *Euro-Par Workshops*, 2013, pp. 406–415.

[15] H. Lim, Y. Han, and S. Babu, "How to Fit when No One Size Fits," in *CIDR*, 2013.

[16] I. Gog *et al.*, "Musketeer: all for one, one for all in data processing systems," in *EuroSys*, 2015.

[17] R. Edjlali, A. M. Ronthal, R. Greenwald, M. A. Beyer, and D. Feinberg, "Magic Quadrant for Data Management Solutions for Analytics," Gartner, 2017.

[18] D. Agrawal, M. L. Ba, L. Berti-uille, S. Chawla, A. K. Elmagarmid, H. Hammady, Y. Idris, Z. Kaoudi, Z. Khayyat, S. Kruse, M. Ouzzani, P. Papotti, J.-A. Quianuiz, N. Tang, and M. J. Zaki, "Rheem: Enabling Multi-Platform Task Execution," in *SIGMOD*, 2016, pp. 2069–2072.

[19] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J. Quiané-Ruiz, N. Tang, and S. Yin, "BigDansing: A System for Big Data Cleansing," in *SIGMOD*, 2015, pp. 1215–1230.

[20] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in *SIGMOD*, 2008, pp. 1099–1110.

[21] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang, "The Data Civilizer System," in *CIDR*, 2017.