Rheem: Enabling Multi-Platform Task Execution

Divy Agrawal[2*] Lamine Ba[1] Laure Berti-Equille[1] Sanjay Chawla[1] Ahmed Elmagarmid[1]
Hossam Hammady[1] Yasser Idris[1] Zoi Kaoudi[1] Zuhair Khayyat[4*] Sebastian Kruse[3*]
Mourad Ouzzani[1] Paolo Papotti[5*] Jorge-Arnulfo Quiané-Ruiz[1] Nan Tang[1] Mohammed J. Zaki[6*]

[1]Qatar Computing Research Institute, HBKU [2]University of California, Santa Barbara [3]Hasso Platner Institute
[4]King Abdullah University of Science and Technology [5]Arizona State University [6]Rensselaer Polytechnic Institute

ABSTRACT

Many emerging applications, from domains such as healthcare and oil & gas, require several data processing systems for complex analytics. This demo paper showcases RHEEM, a framework that provides multi-platform task execution for such applications. It features a three-layer data processing abstraction and a new query optimization approach for multi-platform settings. We will demonstrate the strengths of RHEEM by using real-world scenarios from three different applications, namely, machine learning, data cleaning, and data fusion.

1. NEED FOR FREEDOM

Following the philosophy "one size does not fit all", we have embarked on an endless race of developing data processing platforms for supporting different tasks, e.g., DBMSs and MapReduce-like systems. While these systems allow us to achieve high performance and scalability, users still face two major problems.

Platform Independence. Users are faced with a large number of choices on where to process their data. Each choice comes with possibly orders of magnitude differences in terms of performance. Moreover, whenever a new platform that achieves better performance than the existing ones becomes available, users are enticed to move to the new platform, e.g., Spark taking over Hadoop. Typically, such a move does not come without pain. Therefore, there is a clear need for a system that frees us from the burden and cost of re-implementing and migrating applications from one platform to another.

Multi-Platform Task Execution. Several complex data analytic pipelines are emerging in many different domains. These complex pipelines require combining multiple processing platforms to perform each task of the process and then integrating the results. Performing this combination and integration requires users to be intimate with the intricacies of the different processing platforms and to deal with their interoperability. There is a clear need for a system that eases the integration among different processing platforms by automatically dividing a task into subtasks and determining the underlying platform for each subtask.

Motivating Example. Let us illustrate these two problems by using an oil & gas use case [7]. A single oil company can produce more than 1.5TB of diverse data per day [3], which may be structured or unstructured. During the exploration phase, data has to be acquired, cleaned, integrated, and analyzed in order to predict if a reservoir would be profitable. Thousands of downhole sensors in exploratory wells produce real-time seismic data for monitoring resources and environmental conditions. Users integrate these data with the physical properties of the rocks to visualize volume and surface renderings. From these visualizations, geologists and geophysicists formulate hypotheses and verify them with machine learning methods. Thus, an application supporting such a complex analytic pipeline has to access several sources for historical data (relational, but also text and semi-structured), remove the noise from the streaming data coming from the sensors, possibly integrate a subset of these data sources, and run both traditional (such as SQL) and statistical analytics (such as machine learning algorithms) over different processing platforms.

Rheem. We recently presented a vision of RHEEM[1], our solution to tackle these two problems and thus provide data processing freedom [2]. We propose a three-level data processing abstraction that allows a large variety of applications to achieve processing *platform independence* as well as *multi-platform task execution*. The latter makes our proposal distinct from [5], which does not include an optimizer for automatic multi-platform task execution. Furthermore, in contrast to [6], RHEEM also considers the cost of data movement across underlying platforms. In this demonstration, we will showcase the benefits of RHEEM in terms of flexibility and efficiency. To prove its applicability, we will present three applications, namely, machine learning, data cleaning, and truth discovery.

2. RHEEM OVERVIEW

The current prototype of RHEEM provides two main concepts of the vision presented in [2]: the *data processing abstraction* and the *multi-platform task optimizer*. In this section, we start by describing the general architecture of RHEEM. We then briefly discuss its two main features.

*Work done while at QCRI.

[1]http://da.qcri.org/rheem/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA

© 2016 ACM. ISBN 978-1-4503-3531-7/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2882903.2899414

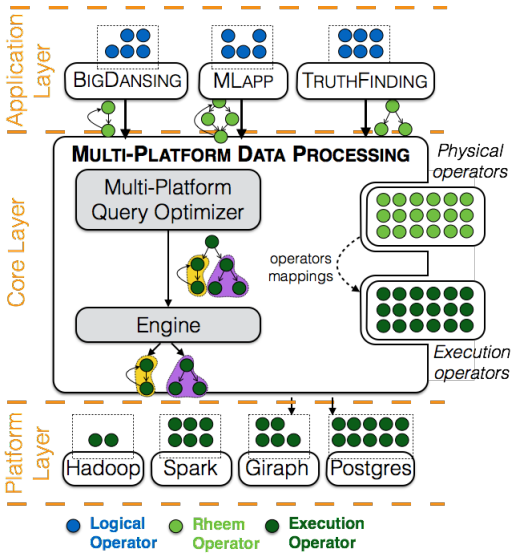

Figure 1: Rheem architecture.

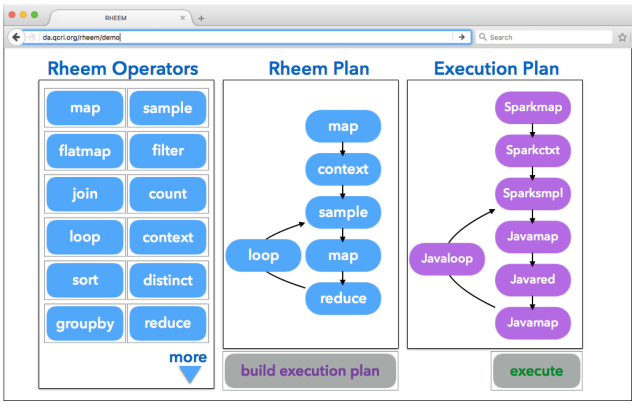

Figure 2: Rheem development interface.

2.1 Overview

Overall, our system provides a three-layer data processing abstraction that sits between user applications and data processing platforms such as Hadoop and Spark (see Figure 1): (i) an application layer that models all application-specific logic; (ii) a core layer that provides the intermediate representation between applications and processing platforms; and (iii) a platform layer that embraces the underlying processing platforms. The communication among these three layers is enabled by operators defined as user defined functions (UDFs). RHEEM provides a set of operators at each layer, namely, logical operators, Rheem operators, and execution operators. Using user-provided implementations of the logical operators specific to the application, the application layer produces a set of possible optimized Rheem plans. An application passes these Rheem plans to the core layer together with the cost functions to help the optimizer in choosing the best plan. These cost functions are obtained by applications and passed to the core layer as UDFs. At the core layer, RHEEM performs several multi-platform optimizations and outputs an execution plan. Then, the underlying processing platforms might further optimize a subplan for better performance. Notice that, in contrast to a DBMS, RHEEM decouples the core (physical) level from the execution one. This separation allows applications to express a Rheem plan in terms of algorithmic needs, without being tied to any platform.

These three layers allow RHEEM to provide applications with platform independence, which is exploited by the RHEEM optimizer to perform multi-platform task execution. Such execution is crucial to achieve the best performance at all times. In the following, we discuss in more details the data processing abstraction as well as the RHEEM optimizer.

2.2 Data Processing Abstraction

We now discuss how developers can define operators at the three levels of the RHEEM data processing abstraction.

Application Layer. A logical operator is a UDF that acts as an application-specific unit of data processing. Basically, it is a template where users provide the logic of their tasks. Such an abstraction enables both ease-of-use, by hiding underlying implementation details from the users (e.g., task parallelization), and high performance, by allowing several optimizations, e.g., seamless distributed execution. A logical operator works on data quanta; these are the smallest units of data from an input datasets. For example, a data quantum might represent a tuple in the input dataset or a row in a matrix. This fine-grained data model allows us to apply operators in a highly parallel fashion and thus achieve better performance.

To provide more insights about this layer, let us illustrate its use via a machine learning (ML) example. Consider a developer who wants to offer end users logical operators to implement various ML algorithms. He can define three basic operators: (i) Initialize, to initialize algorithm-specific parameters, e.g., initializing cluster centroids, (ii) Process, for the computations required by the ML algorithm, e.g., finding the nearest centroid of a point, and (iii) Loop, for specifying the stopping condition. Users can implement algorithms, e.g., SVM, K-means, and linear/logistic regression, using these operators.

Core Layer. This layer, which is at the heart of RHEEM, exposes a pool of Rheem operators. Each represents an algorithmic decision for executing an analytic task. A Rheem operator is a platform-independent implementation of a logical operator whereby a developer can deploy a new application on top of RHEEM. For example, in the above ML example, the application optimizer maps Initialize to a Map Rheem operator and Process to a GroupBy Rheem operator. The system also allows developers to define new operators as needed. Once an application has produced a Rheem plan, the system translates this plan into an execution plan by optimizing it according to the underlying processing platforms. Therefore, in contrast to DBMSs, RHEEM produces execution plans that can run on multiple platforms.

Platform Layer. An execution operator (in an execution plan) defines how a task is executed on the underlying processing platform. In other words, an execution operator is the platform-dependent implementation of a Rheem operator. For instance, consider again the above ML example, the MapPartitions and ReduceByKey execution operators for Spark are one way to perform Initialize and Process.

Defining mappings between Rheem and execution operators is the developers' responsibility whenever a new execution platform is plugged in. RHEEM relies on an internal mapping structure that models the correspondences be-

tween operators together with context information such as cost functions. The context is needed for the effective and efficient execution of each operator.

2.3 Multi-Platform Optimization

The optimizer is responsible for translating a given abstract RheemPlan (or a set of alternatives) into the most efficient ExecutionPlan. The resulting ExecutionPlan consists of a set of platform-specific subplans. Notice that this optimization problem is quite different from traditional database query optimization and thus poses several challenges. First, RHEEM itself is highly extensible and hence neither the RheemPlan operators nor the platforms are fixed. Second, the optimizer should be extensible on how to translate RheemPlans to platform-specific plans. Finally, RHEEM's data processing abstraction is based on UDFs, so operators appear to the optimizer as black-boxes; making cost and cardinality estimations harder.

RHEEM tackles these challenges as follows. First, it applies an extensible set of graph transformations to the RheemPlan to find alternative ExecutionPlans. Then, it compares those alternatives by using the ExecutionOperator's cost functions. These can either be given or learned, and are parameterized *w.r.t.* the underlying hardware (e.g., number of computing nodes for distributed operators). Because RHEEM data processing abstraction is based on UDFs, which are black-boxes for the optimizer, domain-specific optimizations have to be done in collaboration with applications. RHEEM lets applications expose semantic properties about their functions as in [10], and furthermore provides optimization hints (e.g., numbers of iterations), constraints (e.g., physical collocation of operators), and alternative plans. The optimizer uses those artifacts where available in a best-effort approach.

When the optimizer has decided upon an ExecutionPlan, the Engine executes that plan by (i) scheduling the different subplans, (ii) orchestrating the data flow across platforms, and (iii) collecting statistics of the execution to further improve its optimizations.

3. DEMONSTRATION

Our main goal in this demo is to show the benefits of RHEEM with respect to three main aspects: (i) diversity of applications, (ii) platform independence, and (iii) multi-platform execution. The audience will be able to interact with RHEEM directly through its GUI, as well as with applications, namely machine learning, data cleaning, and truth discovery, built on top of the system. The GUI enables users to drag-and-drop Rheem operators to create a physical plan and see how the RHEEM optimizer transforms it into an execution plan (see Figure 2).

3.1 Machine Learning

Many current applications, such as our oil & gas example, require highly efficient machine learning algorithms (MLAs) to perform scalable statistical analytics. However, ensuring high efficiency for MLAs is challenging because of (i) the amount of data involved and (ii) the number of times a MLA has to process the data. Simply distributing MLAs is not obvious due to their iterative nature. Hence, current distributed solutions suffer from performance bottlenecks. In addition, users have to deal with many physical details for implementing a MLA. Achieving high performance and ease-of-use are major lacunae that need to be urgently addressed. Finally, distributing an MLA is not always the best choice to proceed, especially if the data is small or the MLA is sequential. In this demo, we will show how RHEEM tackles the above problems through platform independence and multi-platform execution.

Scalable Clustering. We will consider the real case from a large airline based in the middle east to carry out large scale clustering for personalizing promotion offers to customers. For this, we will use the K-means algorithm and demonstrate the power of RHEEM to distribute the execution of a classical clustering algorithm depending on the dataset size. The audience will also be able to choose among a selection of input datasets are taken from UCI, a publicly available ML respository, and visually see the Rheem plans produced by the application as well as the execution plans produced. The goal of this use case is to show how platform independence not only frees the users from platform-specific implementation details but can also lead to huge performance benefits.

Multi-Platform Gradient Descent. We consider three variations of gradient descent (GD), namely batch GD (BGD), stochastic GD and mini-batch GD (MGD). The audience will be able to choose among these three methods to perform classification on a set of input datasets from UCI as well as tune some parameters such as the batch size. Depending on the chosen method and the size of the batch, the users will be able to see how a specific plan can run partly in Spark and partly in a JVM platform. The overall goal of this use case is to demonstrate how multi-platform execution achieves better performance.

Datasets. We will use a dataset from a large airline based in the middle east. The data is spread across intra-organizational boundaries making it impossible to create a unified dataset to apply a standard clustering algorithm. We will show how RHEEM operators can be used to decouple the algorithm design from the underlying processing platform which in turn can be decoupled from the precise data storage layout. The second set of datasets we will use is taken from UCI, a publicly available ML repository.

3.2 Data Cleaning

Data cleaning, which is detecting and repairing data errors, is critical in data management and data analytics. This is because high-quality business decisions must be made based on high-quality data. In response to the need of supporting heterogeneous and ad-hoc quality rules for various applications, we have built a commodity data cleaning system NADEEF [4]. We have further extended NADEEF using RHEEM [8]. In this demo, users will have the opportunity to experience how RHEEM can boost the performance of NADEEF through platform independence.

Rule Specification in Logical Level. Figure 3 displays the NADEEF GUI for specifying data quality rules. The users can either (a) load rules using rule classes e.g., CFDs, MDs or DCs; or (b) implement a customized rule by writing functions based on programming interface in a few lines of code.

Scalable NADEEF. We will first show how logical NADEEF operators can be mapped to RHEEM operators. Through these different operators, RHEEM frees NADEEF from platform-specific implementation details. That is, RHEEM will decide, based on the logical NADEEF operators, the best platform to execute this task for best performance. Moreover, to show extensibility, we extended the

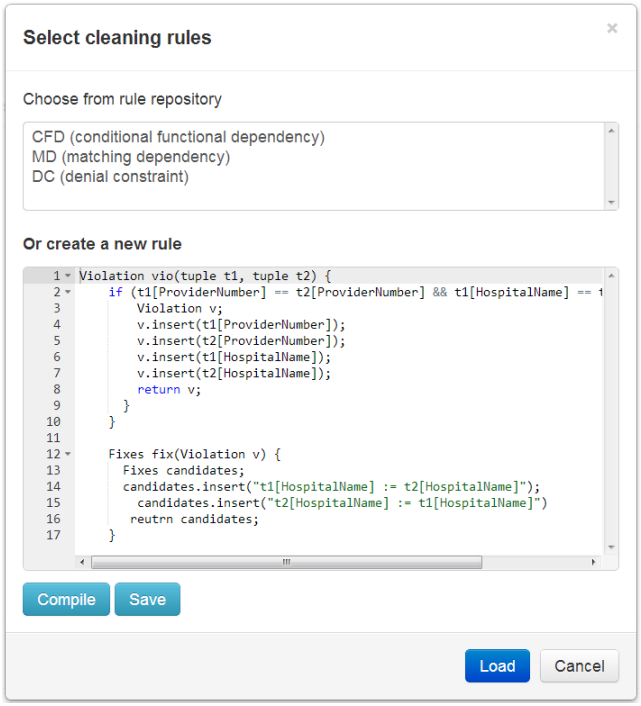

Figure 3: Rule specification in NADEEF.

set of physical RHEEM operators with a new join operator (called IEJoin [9]) to boost performance. This new operator provides a fast algorithm for joins containing only inequality conditions.

Datasets. We use three real-world large datasets. The first one is from an intelligence company that monitors over 700K Web sources. Errors are captured by temporal FD rules [1]. The second is a traffic data of Doha. Duplicates and erroneous readings are reported due to anomalies both in the detectors and in the Bluetooth devices. The third dataset comes from sensors reading from wells in an oil field. Erroneous readings need to be detected from different sensors in the Well during its normal operation, shutting, and opening . They are detected using different statistics-based data cleaning rules. We will demonstrate the significant performance gain (in orders of magnitude), by leveraging the easy mapping from NADEEF logical operators to RHEEM operators, and the platform independence of RHEEM to find the fastest execution plan.

3.3 Truth Discovery

Many real-world applications, e.g., our oil & gas use case, might face the problem of discovering the truth when merging conflicting information from a collection of heterogeneous sources. Such settings can use truth discovery algorithms that aim at resolving conflicts from different data sources in an automatic manner. Most of current truth discovery algorithms conduct an iterative process, which converges when a certain accuracy threshold or a given user-specified number of iterations is reached. As a result, some operations, such as the pairwise value similarity, highly impact the performance of truth discovery algorithms, limiting their scalability [11]. In fact, most truth discovery techniques are platform-dependent implementations with single-node execution. In this demo, we will use a typical data fusion scenario to demonstrate how RHEEM enables truth discovery algorithms to scale and be platform independent, with no additional effort.

Scalable Truth Discovery. We will show how a truth discovery application can be built on top of RHEEM, thanks to the flexibility of its operators. In particular, we will show how such an application can leverage the platform independence of RHEEM to overcome the common single-node execution of such algorithms. We will show how RHEEM leverages Spark by seamlessly distributing operations like value similarity.

Datasets. We showcase our scalable RHEEM truth discovery application using a large-size real-world biographical dataset. We show the improvement obtained from distributing RHEEM operations on SPARK against single-node execution on Java platform. Our dataset, whose entries (claims and sources) are extracted from several Wikipedia articles, contains over 10 millions values from one million sources.

4. REFERENCES

[1] Z. Abedjan, C. G. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker. Temporal rules discovery for web data cleaning. *PVLDB*, 9(4):336–347, 2015.

[2] D. Agarwal, S. Crawla, A. Elmagarmind, Z. Kaoudi, M. Ouzzani, P. Papati, J.-A. Quiané-Ruiz, N. Tang, and M. J. Zaki. Road to Freedom in Big Data Analytics. In *EDBT*, 2016.

[3] A. Baaziz and L. Quoniam. How to use big data technologies to optimize operations in upstream petroleum industry. In *21st World Petroleum Congress*, 2014.

[4] M. Dallachiesa et al. NADEEF: A Commodity Data Cleaning System. In *SIGMOD*, 2013.

[5] A. Elmore et al. A Demonstration of the BigDAWG Polystore System. In *VLDB 2015 (demo)*, 2015.

[6] I. Gog, M. Schwarzkopf, N. Crooks, M. P. Grosvenor, A. Clement, and S. Hand. Musketeer: All for One, One for All in Data Processing Systems. In *EuroSys*, 2015.

[7] A. Hems, A. Soofi, and E. Perez. How innovative oil and gas companies are using big data to outmaneuver the competition. Microsoft White Paper, http://goo.gl/2Bn0xq, 2014.

[8] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzanni, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. BigDansing: A System for Big Data Cleansing. In *SIGMOD*, 2015.

[9] Z. Khayyat, W. Lucia, M. Singh, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and P. Kalnis. Lightning Fast and Space Efficient Inequality Joins. *PVLDB*, 8(13), 2015.

[10] A. Rheinländer, A. Heise, F. Hueske, U. Leser, and F. Naumann. SOFA: An extensible logical optimizer for UDF-heavy data flows. *Inf. Syst.*, 52:96–125, 2015.

[11] D. A. Waguih and L. Berti-Equille. Truth Discovery Algorithms: An Experimental Evaluation. *CoRR*, 1409.6428, 2014.