Cross-Platform Data Processing: Use Cases and Challenges

Zoi Kaoudi and Jorge-Arnulfo Quiané-Ruiz
Qatar Computing Research Institute, HBKU
Doha, Qatar
$\{zkaoudi, jquianeruiz\}$@$hbku.edu.qa$

***Abstract*—There is a zoo of data processing platforms which help users and organizations to extract value out of their data. Although each of these platforms excels in specific aspects, users typically end up running their data analytics on suboptimal platforms. This is not only because choosing the right platform among the myriad of big data platforms is a daunting task, but also due to the fact that today's data analytics are moving beyond the limits of a single platform. Thus, there is an urgent need for cross-platform data processing, i.e., using more than one data processing platform to perform a data analytics task. Despite the need, achieving this is still a dreadful process where developers have to get intimate with many systems and write ad hoc scripts for integrating them. This tutorial is motivated by this need. We will discuss the importance of supporting cross-platform data processing in a systematic way as well as the current efforts to achieve that. In particular, we will introduce a classification of the different cases where an application needs or benefits from cross-platform data processing and the challenges of each case. Along with this classification, we will also present the efforts known up to date to support cross-platform data processing. We will conclude this tutorial with a discussion of several important open problems.**

I. The Case For System-Polygamy

Since already several years ago, we have been witnessing the emergence of applications that produce diverse and large amounts of data. As a result, we have embarked on an endless race to develop data processing platforms (*platforms* for short), such as Spark and Giraph, that enable users and organizations to extract value out of this diverse and big data asset [1]. Just under the umbrella of NoSQL, there are reportedly over 200 different platforms[1]. Although each of these platforms excels in different aspects in the design space, users typically end up running their data analytics on suboptimal platforms. This is because choosing the right platform among the myriad of big data platforms is simply a daunting task. Furthermore, data analytics are moving beyond the limits of a single platform, which makes the task of choosing the right platform (or platforms) much more difficult.

We are indeed observing an emergence of more and more applications requiring several platforms to perform data analytics. For example, (i) IBM reported that North York hospital needs to process 50 diverse datasets, which are on a dozen different internal platforms [2], (ii) oil & gas companies need to process large amounts of data they produce everyday [3], e.g., a single oil company can produce more than 1.5TB of diverse (structured and unstructured) data per day [4], (iii) business intelligence typically require a single analytics pipeline composed of different processing platforms [5], (iv) several data warehouse applications require aggregated, projected, or selected data to be moved from a big data platform, such as Hadoop or Spark, into a relational database for further analysis [6], [7], and (v) airlines need to analyze large datasets, which are produced by different departments and reside on multiple data sources, in order to produce global reports for decision makers [8]. These are just few examples of emerging applications that use or can exploit the use of a diversity of platforms for effectiveness or efficiency.

[1]http://db-engines.com

Moving Beyond One Single Platform. We call *cross-platform data processing* the fact of requiring more than one platform for performing a data analytics task. The task can be from a very simple task, such as k-means clustering, to a very complex data analytical pipeline, e.g., one that includes data cleaning, preparation, feature extraction and model training. Unfortunately, achieving cross-platform data processing is quite challenging, because applications are typically tied to one single platform. The common practice to support cross-platform data processing is to develop several specialized analytic applications on top of different platforms and write ad-hoc programs (or scripts) to glue them all together. This is not only a tedious and costly task, but it also requires being intimate with the intricacies of the different platforms to achieve high efficiency and scalability. Furthermore, users need to manually combine results to draw a conclusion. The research community has recently recognized the need for a systematic solution that enables cross-platform data processing [9], [10], [11], [12], [13]. The holy grail is to allow users to write platform-agnostic queries while an intermediate system decides on which platforms to execute each query with the goal of minimizing its cost (e.g., runtime or monetary cost). However, despite many current efforts [5], [14], [15], [16], [12], [9], the road to make automatic cross-platform data processing a reality is still way long!

Tutorial's Goal & Outline. In this tutorial, we have three objectives: introduce the different use cases of cross-platform data processing, present the challenges and current efforts to support it, and discuss a research agenda for achieving automatic and efficient cross-platform data processing. In detail, we will start by introducing a classification for the

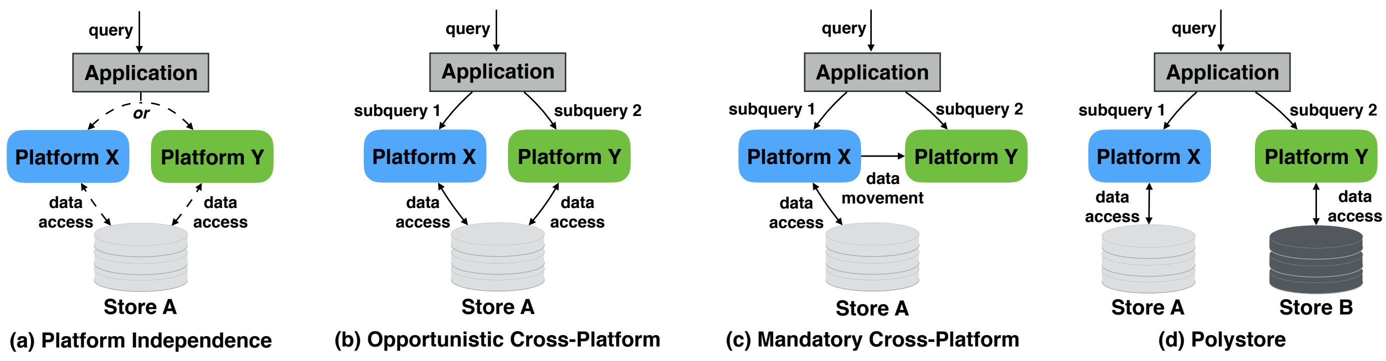

Fig. 1. Cross-platform use cases.

different cases where an application needs support for cross-platform data processing. We will show that, surprisingly, most of us have required more than once support for cross-platform data processing even without noticing it. We will present the different existing systems that support cross-platform data processing and classify them using the proposed classification. Especially, we will discuss the approaches and algorithms of each of these works and list their benefits and limitations. We will then highlight the crucial research areas that have not yet been exploited to achieve efficient cross-platform data processing. In the remainder of this document, we present the different parts in which this tutorial will be structured.

Targeted Audience. The intended audience consists of researchers and developers who are keen to know how cross-platform data processing is essential for today's applications, but are also seeking ways to speed-up their applications in an easy and systematic manner.

II. Cross-Platform Use Cases

Overall, we identify four different cases where an application needs support for cross-platform data processing:

- *Platform-independence*: applications might require to switch platforms according to the input datasets and/or queries in order to achieve higher performance than when using one single platform (Figure 1(a)). Hive and Mahout are just two clear examples of systems suffering from the lack of platform independence. The community had to implement two other new systems (SparkSQL and MLlib) on top of Spark in order to have the counterparts of these two Hadoop-based systems.
- *Opportunistic cross-platform*: applications might benefit from using multiple platforms throughout a single query (Figure 1(b)). An example that clearly shows the benefits of exploiting multiple platforms to perform one single query is ML4all [17]: It significantly decreases runtimes of machine learning algorithms by simultaneously using both a Java standalone program and Spark.
- *Mandatory cross-platform*: applications might also require to use multiple platforms because the platform where the data resides, e.g., PostgreSQL, cannot perform an incoming query, e.g., a machine learning task (Figure 1(c)). A data warehouse is a typical example of this case [7], where data must be imported from external systems, such as Hadoop, in order to run complex analytics.
- *Polystore*: applications might require to use multiple platforms because the input data is stored on multiple data stores and hence a query must be divided accordingly (Figure 1(d)). The Data Civilizer [18] system is a clear example of a system requiring polystore capabilities: It is a data discovery and integration system that operates over hundreds or thousands of different data sources.

We will start this tutorial by introducing these four different cross-platform use cases and discuss the importance of each one with real applications. In particular, we will contrast cross-platform data processing with parallel, distributed, federated, and mediator database systems, such as Garlic[19], [20], TSIMMIS [21] and Interbase [22], in order to highlight its uniqueness. In the remainder of this tutorial, we will delve into the details of each of these cross-platform use cases and discuss the different efforts to support each of these.

III. Platform Independence

Applications are usually tied to one specific platform. This can be a roadblock for organizations and individuals for three main reasons. First, as new and more efficient platforms become available, developers need to re-implement existing applications on top of faster platforms. For example, Spark SQL [23] and MLlib [24] are the Spark counterparts of Hive [25] and Mahout [26] which were implemented on top of Hadoop. Of course, migrating an application from one platform to another is a time-consuming and costly task and is not always a viable choice. Second, for different inputs of a specific query (or for different incoming queries), a different platform may be the most efficient one. Thus, one cannot determine the best platform statically. For instance, running a specific query on a big data platform for very large datasets is often more beneficial than running it on a single-node platform, such as a DBMS. In contrast, for smaller datasets, running the same query on a single-node platform might be much faster due to little overhead costs. In fact, several applications in companies and organizations run on two different processing platforms, such as the machine learning system of IBM [27]. Figure 1(a) illustrates this case: an incoming query can run on two different platforms, depending on its own and the input dataset characteristics, for efficiency or effectiveness reasons.

In this part of the tutorial, we will delve into the details of platform independence by giving concrete examples from real applications. In addition, we will demonstrate with experimental results how platform independence can speed-up execution. We will then present systems that already support platform independence both domain-specific such as [27], [17], and general-purpose [15], [28], [9].

IV. Opportunistic Cross-Platform

There are cases where applications can be executed on one single platform, but using multiple platforms would benefit in performance (or monetary cost). For instance, users can run a gradient descent algorithm, such as SGD, on top of Spark relatively fast. Still mixing it with a standalone Java program significantly increases performance. This approach is the execution counter-part of *polyglot persistence* [29], where different types of databases are combined to leverage their individual strengths. However, supporting opportunistic cross-platform data processing is not only time consuming but also developers must know (i) a priori all the situations where it is beneficial to use multiple platforms and (ii) how exactly to use them (i.e., which part of the query to run on a specific platform). Obviously, in most cases, spotting these opportunities is very hard, if not impossible. Even worse, similarly to the platform independence case, one usually cannot spot such opportunities statically. Figure 1(b) depicts the opportunistic cross-platform case: one single incoming query is divided into two subqueries with the only goal of decreasing its total execution time (or monetary cost).

In this part of the tutorial, we will discuss the challenges of the opportunistic cross-platform data processing and some solutions that have been proposed for dealing with them. For instance, Rheem [15] provides an optimizer to automatically discover the platforms that each subquery has to be run on for obtaining maximum performance. Similar systems that go towards this direction include Ires [9] and Musketeer [16]. We will delve into the details of such systems and emphasize their differences and limitations.

V. Mandatory Cross-Platform

In other cases, applications need to go beyond the functionalities offered by the platform on which the data is stored. This is because there is no platform that can fit all the data analytics spectrum (following the *one-size-does-not-all* dictum). Imagine for example that a dataset is stored on a relational database and a user needs to perform a clustering query. Doing so inside the relational database might simply be disastrous in terms of performance. Thus, the user needs to move the data out of the relational database. For example, she might move the data to HDFS in order to use Spark [30], which is known to be efficient for iterative queries. A similar situation occurs in complex data analytics applications with disparate subqueries. As an example, an application might extract a graph from a text corpus to perform subsequent graph analytics on. This might require using both a text and a graph analytics system. The required integration of platforms is tedious, repetitive, and particularly error-prone. Currently, developers of such applications write ad-hoc programs or scripts to move the data out of the database and integrate different platforms. Figure 1(c) illustrates this case: one single incoming query must be divided into two subqueries as the platform, where its input data is, cannot perform it entirely.

In this third part of the tutorial, we will investigate efforts that have been made on this direction such as MapReduce-based integration systems [31], [6]. In particular, we will discuss the importance of achieving mandatory cross-platform data processing in a systematic manner.

VI. Polystore

In many current applications, datasets are produced by different sources and on different formats (*data lakes*). Datasets in data lakes reside natively on their format and hence on different storage platforms, such as DBMSs, document stores, key-value stores, and pure file systems. Oil & gas [3], [4], health care [2], airline [8] industries, and business intelligence [5] are just few examples of such scenarios. In these cases, users often not only perform tedious, time-intensive, and costly data migration, but also integration tasks for analyzing the data. Therefore, supporting the execution of queries over multiple data stores systematically is very important. Figure 1(d) shows this polystore case: one single incoming query is divided into two subqueries because its data is stored on two different storage engines.

In this part of the tutorial, we will elaborate on use cases that require or already use such cross-platform data processing. In addition, we will discuss the features and limitations of polystore systems such as [32], [33]. For instance, BigDawg [32] provides a declarative language that enables users to specify where to run their subqueries via its Scope and Cast commands. Although this facilitates the task of integrating several platforms, users still must determine the platforms to run each part of their queries.

VII. Research Agenda

Although supporting cross-platform data processing is crucial in today's applications, the road to establish it in an automatic and efficient way contains many open research problems. In the last part of the tutorial, we will highlight the research areas that have not been exploited yet and draw a research agenda with the open problems. We list only few of them in the following:

(i) *Automatic integration with new platforms*. A cross-platform system must keep up with new platforms that appear or existing ones that get updated. Achieving this with very little effort or even with no effort at all is a very challenging task.

(ii) *Cardinality and cost estimation*. Setting the cardinalities and cost of operators is a hard problem even in more conventional environments, such as DBMSs [34]. An interesting direction is to investigate machine learning techniques (e.g., regression or even deep learning) for estimating the cardinality and costs of operators.

(iii) *Data movement*. Optimizing data movement is a very crucial aspect in a cross-platform setting where data has to be moved constantly from one platform to another. Although there has already been some initial works on this, e.g., [35], there is still room for improvement. For example, finding the right format for intermediate data so that data movement is speeded up is an open problem.

(iv) *Fault tolerance*. While a cross-platform system can rely on the underlying platforms to recover from failures when executing queries on them, it is also crucial to be able to recover from failures across platforms, e.g., when data is being moved from one platform to another.

Presenters

Zoi Kaoudi is a Scientist in the Qatar Computing Research Institute (QCRI), HBKU. She has previously worked in IMIS-ATHENA RC as a research associate and INRIA as a postdoctoral researcher. She received her PhD from the National and Kapodistrian University of Athens in 2011. She has previously presented tutorials at ICDE 2013 and SIGMOD 2014. Her research interests include machine learning systems, big data management, and distributed RDF query processing and reasoning. Personal webpage: http://da.qcri.org/zkaoudi

Jorge Arnulfo Quiane-Ruiz is a Senior Scientist at the Qatar Computing Research Institute (QCRI), HBKU. He has previously worked in Saarland University and INRIA. He received his PhD from University of Nantes in 2008. He has previously presented tutorials at VLDB 2012 and received an Excellent Presentation Award at VLDB 2014. His research mainly focuses on efficient and scalable big data management. Personal webpage: http://da.qcri.org/jquiane

Acknowledgments. We have studied the material on which the tutorial is based while collaborating with many colleagues. We would like to thank the entire RHEEM team[2] for their feedback and support. Especially, we would like to thank Sebastian Kruse and Sanjay Chawla for many insightful discussions.

References

[1] M. Stonebraker and U. Çetintemel, ""One Size Fits All": An Idea Whose Time Has Come and Gone (Abstract)," in *ICDE*, 2005.

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

[3] 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.

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

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

[6] 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.

[7] 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.

[8] "Fortune magazine," http://fortune.com/2014/06/19/big-data-airline-industry/.

[9] K. Doka, N. Papailiou, V. Giannakouris, D. Tsoumakos, and N. Koziris, "Mix 'n' match multi-engine analytics," in *IEEE BigData*, 2016, pp. 194–203.

[10] D. Agrawal *et al.*, "Road to Freedom in Big Data Analytics," in *EDBT*, 2016, pp. 479–484.

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

[12] M. Abadi *et al.*, "TensorFlow: A System for Large-Scale Machine Learning," in *OSDI*, 2016, pp. 265–283.

[13] S. Palkar, J. J. Thomas, A. Shanbhag, M. Schwarzkopt, S. P. Amarasinghe, and M. Zaharia, "A common runtime for high performance data analysis," in *CIDR*, 2017.

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

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

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

[17] Z. Kaoudi, J.-A. Quiane-Ruiz, S. Thurumuruganathan, S. Chawla, and D. Agrawal, "A Cost-based Optimizer for Gradient Descent Optimization," in *SIGMOD*, 2017.

[18] 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.

[19] M. J. Carey *et al.*, "Towards Heterogeneous Multimedia Information Systems: The Garlic Approach," in *RIDE-DOM*, 1995, pp. 124–131.

[20] M. T. Roth and P. M. Schwarz, "Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources," in *VLDB*, 1997, pp. 266–275.

[21] S. S. Chawathe *et al.*, "The TSIMMIS Project: Integration of Heterogeneous Information Sources," in *IPSJ*, 1994, pp. 7–18.

[22] O. A. Bukhres *et al.*, "InterBase: An Execution Environment for Heterogeneous Software Systems," *IEEE Computer*, vol. 26, no. 8, pp. 57–69, 1993.

[23] "Spark SQL programming guide," http://spark.apache.org/docs/latest/sql-programming-guide.html.

[24] "Spark MLlib: http://spark.apache.org/mllib."

[25] "Apache Hive: A data warehouse software for distributed storage," http://hive.apache.org.

[26] "Apache Mahout. http://mahout.apache.org."

[27] 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.

[28] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum, "FlumeJava: easy, efficient data-parallel pipelines," in *ACM Sigplan Notices*, vol. 45, no. 6, 2010, pp. 363–375.

[29] P. J. Sadalage and M. Fowler, *NoSQL distilled: A brief guide to the emerging world of polyglot persistence*. Addison-Wesley Professional, 2012.

[30] "Apache Spark: Lightning-Fast Cluster Computing," http://spark.incubator.apache.org/.

[31] J. LeFevre, J. Sankaranarayanan, H. Hacigümüs, J. Tatemura, N. Polyzotis, and M. J. Carey, "MISO: souping up big data query processing with a multistore system," in *SIGMOD*, 2014, pp. 1591–1602.

[32] A. J. Elmore *et al.*, "A Demonstration of the BigDAWG Polystore System," *PVLDB*, vol. 8, no. 12, pp. 1908–1911, 2015.

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

[34] V. Leis *et al.*, "How good are query optimizers, really?" *Proc. VLDB Endow.*, vol. 9, no. 3, pp. 204–215, 2015.

[35] B. Haynes, A. Cheung, and M. Balazinska, "PipeGen: Data Pipe Generator for Hybrid Analytics," in *SoCC*, 2016, pp. 470–483.

[2] http://da.qcri.org/rheem/about.html