Next-generation ETL Framework to address the challenges posed by Big Data

Syed Muhammad Fawad Ali
Poznan University of Technology
Poznan Poland
trivago N.V.
Leipzig Germany
fawadali.ali@gmail.com

ABSTRACT

The specific features of Big Data i.e., variety, volume, and velocity call for special measures to create ETL data pipelines and data warehouses. A rapidly growing need for analyzing Big Data calls for novel architectures for warehousing the data, such as *data lakes* or *polystores*. In both of the architectures, ETL processes serve similar purposes as in traditional data warehouse architectures. Except the fact that the data to process has multitude of formats and the relationships between data are often very complex. Furthermore, most of the times data transformations are required on-the-fly that have to be executed and completed in near real-time. For these reasons designing and optimizing ETL workflows for Big Data is much more difficult than for traditional data. In this paper, we focus on the ETL aspect of Big Data and propose an extendable ETL workflow that addresses the aforementioned challenges posed by Big Data.

1 INTRODUCTION

Since early 2000s, the volume of produced, collected, and stockpiled digital data has been continuously growing exponentially. It is expected that by 2020 there will be more than 16 Zettabytes (16 Trillion GB) of useful data.This Big Data provides great opportunities and harnessing that leads to great benefits in science and business. Thus, having the right technological basis to exploit the potential of Big Data, it is essential for most organizations to gain competitive advantage or even survive in today's world.

Big data itself requires a great scientific contribution to deal with it. That is, most of such data is not easily accessible or can be processed by existing technologies and this imposes to academics a lot of challenges to be solved and questions to be answered. For example, traditional ETL or data integration tools work well with clean and consistent data and are not capable to efficiently deal with the variety of Big Data. Although there are already numerous Big Data, IoT, and analytics solutions that enable people to obtain valuable insights from vast amount of data, such solutions are still in their early stages of development[8]. Therefore, there is a need for developing new and advanced methods and technologies to extract, transform, load, analyze, and visualize such data in order to obtain valuable insights from it.

This paper focuses on the ETL aspect of Big Data. Traditional ETL frameworks, methods, and built-in operators provided by the existing ETL tools are now obsolete in case of Big Data due to its volume, variety, and velocity. The existing ETL tools and frameworks were designed for creating a traditional Data Warehouse (DW), which efficiently supports light-weight computations on smaller data sets. However, Big Data demands new and advanced computations, as an example from the data cleansing side, the messy and noisy nature of Big Data requires new types of cleansing operators, such as outlier detection or de-duplication that specifically fit the ever changing characteristics of the data. The same applies to the data analytics side where we find a zoo of algorithms such as classification, regression, clustering, collaborative filtering, and many more.

We carried out an extensive study on the current practices, short-comings, limitations, and open issues of existing ETL methodologies and tools [1]. According to this study, the still open issues on ETL development become much more difficult to solve in the field of Big Data. Therefore, in this paper, we propose a next-generation extendable ETL framework in order to address the challenges caused by Big Data. The proposed framework is based on the outcome of our aforementioned study.

In Section 2 we present our motivation for the new ETL framework. We then introduce and explain the proposed extendable ETL framework in Section 3. In Section 4 we discuss the related work in the same field. Section 5 contains the conclusion and the future work.

2 MOTIVATION

As mentioned in Section 1, we carried out an intensive study [1] on the existing methods for designing, implementing, and optimizing of ETL workflows. We analyzed several techniques w.r.t their pros, cons, and challenges in the context of metrics such as: autonomous behavior, support for quality metrics, and support for ETL activities as user-defined functions. Following is the summary of conclusions on open research and technological issues in the field of ETL:

(1) The support for semi-structured and unstructured data is very limited. Whereas, the variety of data format especially the unstructured and raw data is growing rapidly. Therefore, there is a need to extend the support for processing an unstructured data in an ETL workflow along with other data formats (e.g., video, audio, binary).
(2) There is a lack or no support for user-defined functions (UDFs) as ETL activities. Whereas, the volume and variety of Big Data require custom functionality in order to perform complex and intensive computations. The reason being, traditional DWs are not optimized enough to store huge volume and variety of data, therefore novel data warehousing architectures like *data lake* [12] or a *polystore* [3] are introduced. These DWs support different kind of data formats, which eventually lead to complex ETL workflows to populate such DWs. Designing ETL workflows for the Big Data is a challenging task because traditional ETL operators are not suitable for processing

Big Data and such tasks have to be implemented by UDFs. Therefore, there is a need to consolidate and fully support UDFs in an ETL workflow along with traditional ETL operators.

(3) Only a few methods emphasized on the issues of efficient, reliable, and improved execution of an ETL workflow. Whereas, today's need of real-time availability of data requires efficient ETL workflows that can quickly process and analyze huge amount of data. Therefore, to improve the execution performance of an entire ETL workflow, techniques based on task parallelism, data parallelism, and a combination of both for traditional ETL operators as well as UDFs are required.

(4) Most of the design methods require ETL developers to extensively provide input during the modeling and design phase of an ETL workflow, thus it can be error prone, time consuming, and inefficient. Hence, there is a need for an ETL framework that shall reduce the work of the ETL developer from a design and performance optimization perspective. The framework should provide recommendations on: (1) an efficient design for an ETL workflow according to the business requirements, (2) how and when to improve the performance of an ETL workflow without conceding other quality metrics.

The consequence of the aforementioned observation is that designing and optimizing ETL workflows for Big Data is much more difficult than for traditional data and is much needed at this point in time.

3 THE EXTENDABLE ETL FRAMEWORK

On the basis of conclusions discussed in Section 2, we present an extendable theoretical *ETL Framework*. A three-layered architecture of the *ETL Framework* is shown in Figure 1.

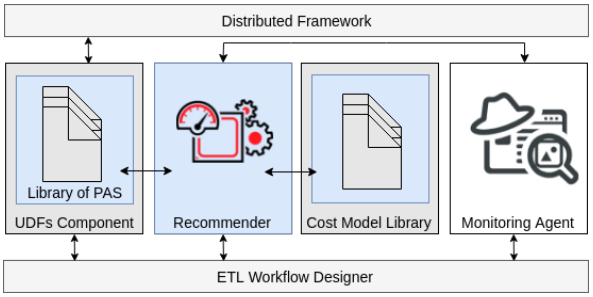

Figure 1: The overall architecture of the *ETL Framework*

The bottom layer is an *ETL Workflow Designer*, which may be any standard open source ETL tool for designing ETL workflows. This layer communicates with the middle layer, which is extendable and consists of the four components: (1) a *UDFs Component*, (2) a *Recommender*, (3) a *Cost Model*, and (4) a *Monitoring Agent*, described in detail in the following sub-sections.

The top layer in the architecture is the *Distributed Framework*. Its task is to execute parallel codes of UDFs in a distributed environment, in order to improve the overall execution performance of an ETL workflow.

3.1 A UDFs Component

The idea behind introducing this component is to assist the ETL developer in writing a parallelizable UDF by separating parallelization concerns from the code.

A UDF is a software program written in any programming, scripting, or procedural language. It allows the ETL developer to extend the functionality of an ETL tool that is outside the scope of the already provided built-in ETL operators. For example, the messy and noisy nature of Big Data demands new types of cleansing operators, such as outlier detection or de-duplication that specifically fit the ever-changing characteristics of the data. The same applies to the data analytics side where we find a zoo of algorithms such as classification, regression, clustering, collaborative filtering, and many more. A UDF can be used to implement aforementioned operators in a Big Data setup or to perform aggregations or any kind of run-time intensive computations on a data that may be necessary before loading into a data warehouse.

A *UDFs component* contains a library of *Parallel Algorithmic Skeletons* (PASs) or parallelizable code templates. These PASs are designed to be executed in a distributed environment, (e.g., a template for MapReduce or Spark to be executed in Hadoop).

The UDFs component requires a basic knowledge of distributed computing and parallelization aspects from the ETL developer.

Figure 2 shows the working of *UDFs Component*. The component provides the already parallelizable code for the list of commonly used Big Data operators (case-based PASs) to the ETL developer (e.g., sentiment analysis, de-duplication of rows, outlier detection) and a list of generic PASs (e.g., worker-farm model, divide and conquer, branch and bound, systolic, MapReduce). The ETL developer either chooses case based PAS or a generic PAS based on his/her requirements.

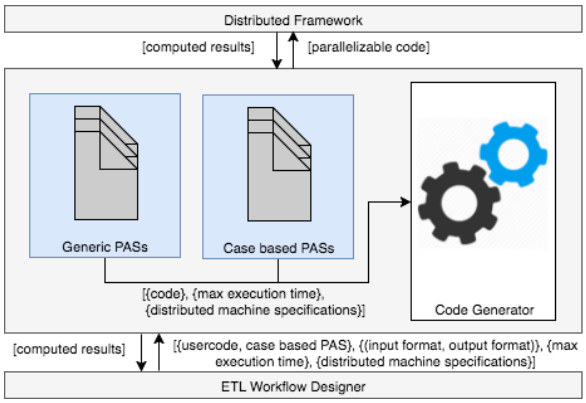

Figure 2: Extendable UDFs Component Architecture

As shown in Figure 2, a generic input to UDFs component is depicted as *[{usercode, case based PAS}, {(input format, output format)}, {max execution time constraint}, {distributed machine specifications}]*. For example, in case of case-based reasoning the ETL developer only has to provide the input and output data formats *{(input format, output format)}*, execution time constraint to run the ETL workflow (e.g., the ETL job must complete execution with in 'x' number of hours *{max execution time}*), and distributed machine specifications *{distributed machine specifications}*, if known. In case of generic PAS, the ETL developer has to provide the basic program for the chosen PAS *{usercode}*, an execution time constraint to run the ETL workflow *{max execution time}*, and distributed machine specifications *{distributed machine specifications}*. That is, for the MapReduce paradigm as a PAS, only Map and Reduce functions would be required. The MapReduce configurations (i.e., partitioning parameters, number of nodes) will be provided by the UDFs component. The *Code*

Generator then generates the configuration and a paralellizable code based on the ETL developer's input to the component about the distributed machine specifications, time constraints on the completion of the ETL workflow, and by the recommendation of the *Recommender* component in the proposed ETL framework. The specific configurations provided by this component are very critical to achieve the right degree of parallelism.

Once the configurations are generated, the code provided by the ETL developer and the distributed environment configurations will be executed in *Distributed Framework*. The computed results are then returned to the ETL workflow for the next steps in the workflow.

3.2 A Recommender

A Recommender includes an extendable set of machine learning algorithms to optimize a given ETL workflow (based on metadata collected during past ETL executions) and to generate a more efficient version of the workflow. Metadata may be collected with the help of *Monitoring Agent*, where it collects various performance statistics of different ETL workflows and provide them to a Recommender. Since, there are a few algorithms that can be applied to optimizing a workflow (e.g., *Dependency Graph* approach) [10, 11], *Scheduling Strategies* [7], the ETL developer would then be able to experiment with alternative algorithms and compare their optimization outcomes.

Recommender component also helps the ETL developer to choose the best possible PAS from a UDFs component based on the developer's input (c.f. Section 3.1) to the Recommender. To provide the optimal PAS to a UDFs component, it uses the *Cost Model* component.

3.3 A Cost Model

The algorithms used by *Recommender* need cost models. A Recommender can choose the appropriate cost model from a library of cost models in order to make optimal decisions based on the ETL developer's input to it.

The library of cost models may include cost models for monetary cost, performance cost, and both cost and execution performance optimization. Since most of the Big Data ETL workflows or UDFs for Big Data are executed in a cloud or a distributed framework, there would be cost models to evaluate the performance of workflows in a cloud computing environment [5, 6] and also to determine the best possible configuration of virtual machines both in terms of execution time and monetary cost [13].

Since the Recommender uses the Cost Model component to provide the optimal PAS to a UDFs component, the cost model would be able to select the optimal PAS based on the *Multiple Choice Knapsack Problem* (MCKP) [4]. For example, suppose an ETL workflow consists of n different computationally intensive UDFs and UDFs component may generate m parallel variants of each UDF, there are m^n combinations of code variants. Therefore, finding an optimal UDF may be mapped to MCKP.

3.4 A Monitoring Agent

Monitoring Agent allows to:

- monitor ETL workflow executions - e.g., number of input rows, number of output rows, execution time of each step, number of rows processed per second.
- identify performance bottlenecks - e.g., which tasks are being delayed or aborted, which tasks need to be optimized.
- report errors - e.g., task or workflow failures and the possible reasons.
- schedule executions - e.g., execution time of ETL workflows and creating a dependency chart for ETL tasks and workflows.
- gather various performance statistics - execution time of each ETL activity w.r.t rows processed per second, execution time of the entire ETL workflow w.r.t rows processed per second, memory consumption by each ETL activity.

This is a standard component of any ETL engine. However, we would store all of the aforementioned collected information in an ETL framework repository to be later utilized by *Recommender* and *Cost Model* in order to make recommendations to the ETL developer and to generate optimal ETL workflows.

4 RELATED WORK

There does not exist much research work in literature on ETL frameworks specifically for Big Data besides some cloud based distributed frameworks (e.g., *Amazon Web Services*[1] stack, *Google Cloud Platform*[2], and *Microsoft Azure*[3]). These cloud based distributed platforms provide several products that help in creating Big Data ETL data pipelines and solutions. However, the provided products are not fully autonomous as well as does not provide recommendations to the ETL developer for creating optimized data pipelines at run-time.

In research, there exists a few stand alone methods, data warehousing architecture, and utilities for the extraction and transformation phases in an ETL workflow for Big data.

In [2], the authors proposed a method to semantically extract the data from a variety of data sources e.g., text, video, email, audio in an ETL workflow. The discussed approach is focused on extracting the data and does not cover the compute-intensive transformation phase required for the 3Vs Big Data. Furthermore, to define the semantics of data it requires a human expert to define ontology, which is a tedious and a time consuming task.

A data warehousing architecture for Big Data is discussed in [9]. The proposed architecture uses HDFS for data storage, Talend Open Studio for ETL transformations, and Hive as a data warehouse. However, the work presented did not mention the difficulties to tackle the 3Vs of Big Data. In our paper, we addressed the issues like complex ETL workflows due to 3Vs of Big Data and how to solve the issues of compute-intensive UDFs. Finally we also provided a fully automated framework to create ETL workflows.

5 CONCLUSION

In this paper, we presented an extendable ETL framework in order to address the challenges posed by Big Data. We proposed this ETL framework on the basis of limitations and shortcomings in the currently existing ETL methodologies and tools. We proposed a UDF's Component to address the the issue of no or minimal support for UDFs and their optimization in currently existing ETL frameworks, which is an integral part to develop ETL transformations for Big Data. Furthermore, we proposed recommendation module that utilizes a library of cost models and retrieves information from a monitoring agent in order to provide recommendations to the ETL developer. The monitoring

[1] https://aws.amazon.com
[2] https://cloud.google.com/products
[3] https://azure.microsoft.com

agent module is proposed to assist the recommendation module as well as an end-to-end monitoring of ETL workflows.

We believe that the proposed ETL framework is a step forward towards a fully automated ETL framework to help the ETL developers optimize ETL tasks and an overall ETL workflow for Big Data with the help of recommendations, montoring agent, and UDFs provided by the tool.

Currently we are working on the first steps towards building a complete *ETL Framework* i.e., (1) a *UDFs Component* - to provide the library of reusable parallel algorithmic skeletons for the ETL developer and (2) *Cost Model* - to generate the most efficient execution plan for an ETL workflow.

ACKNOWLEDGMENTS

The research of Syed Ali has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence Doctoral College" (IT4BI-DC) and trivago N.V.

REFERENCES

[1] S. M. F. Ali and R. Wrembel. From conceptual design to performance optimization of ETL workflows: current state of research and open problems. *The VLDB Journal*, pages 1–25, 2017.

[2] S. K. Bansal. Towards a semantic extract-transform-load (ETL) framework for big data integration. In *Proceedings of International Congress on Big Data*, pages 522–529. IEEE, 2014.

[3] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. Zdonik. The BigDAWG Polystore System. *SIGMOD Record*, pages 11–16, 2015.

[4] T. Ibaraki, T. Hasegawa, K. Teranaka, and J. Iwase. The multiple choice knapsack problem. *Journal of Operations Research Society Japan*, pages 59–94, 1978.

[5] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *Transactions on Parallel and Distributed systems*, pages 931–945, 2011.

[6] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *International Conference on Cloud Computing Technology and Science*, pages 159–168. IEEE, 2010.

[7] A. Karagiannis, P. Vassiliadis, and A. Simitsis. Scheduling strategies for efficient ETL execution. *Information Systems*, pages 927–945, 2013.

[8] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob. Big IoT data analytics: Architecture, opportunities, and open research challenges. *IEEE Access*, pages 5247–5261, 2017.

[9] B. Martinho and M. Y. Santos. An architecture for data warehousing in big data environments. In *Proceedings of Research and Practical Issues of Enterprise Information Systems*, pages 237–250. Springer, 2016.

[10] A. Simitsis, P. Vassiliadis, and T. Sellis. State-space optimization of ETL workflows. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, pages 1404–1419, 2005.

[11] A. Simitsis, K. Wilkinson, U. Dayal, and M. Castellanos. Optimizing ETL workflows for fault-tolerance. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, 2010.

[12] I. Terrizzano, P. Schwarz, M. Roth, and J. E. Colino. Data Wrangling: The Challenging Journey from the Wild to the Lake. In *Proceedings of Conference on Innovative Data Systems Research (CIDR)*, 2015.

[13] V. Viana, D. De Oliveira, and M. Mattoso. Towards a cost model for scheduling scientific workflows activities in cloud environments. In *Proceedings of IEEE World Congress on Services*, pages 216–219, 2011.