Amalur: Next-generation Data Integration in Data Lakes

Rihan Hai Christos Koutras Andra Ionescu Asterios Katsifodimos
Delft University of Technology

ABSTRACT

Data science workflows often require extracting, preparing and integrating data from multiple data sources. This is a cumbersome and slow process: most of the times, data scientists prepare data in a data processing system or a data lake, and export it as a table, in order for it to be consumed by a Machine Learning (ML) algorithm. Recent advances in the area of factorized ML, allow us to push down certain linear algebra (LA) operators, executing them closer to the data sources. With this work, we revisit classic data integration (DI) systems and see how these fit into modern data lakes that are meant to support LA as a first-class citizen.

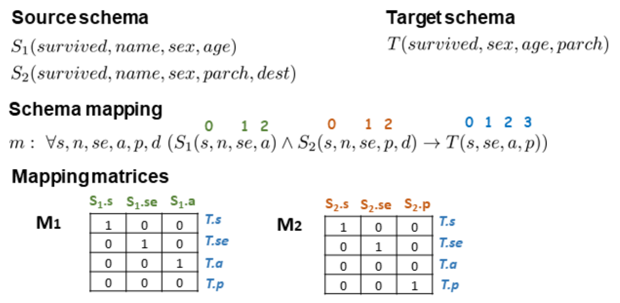
1 INTRODUCTION

Use case and problem. Consider data scientists who try to build an ML model, but relevant data is spread over multiple tables in the data lake. In the traditional way, they need to integrate source tables and export a single table as follows. They first perform data discovery or schema matching to find matched columns containing similar information. The next step is combining these source tables and creating the schema of target table T (also referred to as the mediated schema), which contains all the selected features needed for training. After generating schema mappings and linking entities, this process would result in the materialized target table T for exporting. The above DI process is tedious, even with the help of commercial data integration/preparation tools. Moreover, creating formal mapping specifications or refining mappings are far beyond the skill set of today's data scientists.

Enter Amalur. Many ML tasks are expressed in *linear algebra (LA)*. Given multiple datasets and downstream ML applications, our ultimate goal with Amalur is twofold: 1) It automates data integration tasks, and relieves data scientists from heavy manual labor or complex data transformation queries. 2) It supports LA operators and ML factorization in a data lake system, which reduces data redundancy and improves runtime efficiency. That is, exporting the data can incur huge computation costs as opposed to running those computations within the data lake. Instead, we can significantly improve the run-time efficiency of a LA pipeline if we avoid materializing the target table T by factorizing LA operators [1].

2 IN-LAKE LA PROCESSING

Two types of matrices. The final task of most traditional DI systems falls into either query reformulation (virtual integration) or target instance materialization (data exchange). In either case, we need column matching (via schema matching and mapping) and row matching (via record linkage) between the source tables and


Source schema
$S_1(survived, name, sex, age)$
$S_2(survived, name, sex, parch, dest)$
Target schema
$T(survived, sex, age, parch)$
Schema mapping
0 1 2
0 1 2
0 1 2 3
$m: \forall s, n, se, a, p, d\ (S_1(s, n, se, a) \wedge S_2(s, n, se, p, d) \rightarrow T(s, se, a, p))$
Mapping matrices
M1
S1.s S1.se S1.a
T.s T.se T.a T.p
M2
S2.s S2.se S2.p
T.s T.se T.a T.p

$$M_1 = \begin{array}{c|ccc} & S_1.s & S_1.se & S_1.a \\ \hline T.s & 1 & 0 & 0 \\ T.se & 0 & 1 & 0 \\ T.a & 0 & 0 & 1 \\ T.p & 0 & 0 & 0 \end{array} \qquad M_2 = \begin{array}{c|ccc} & S_2.s & S_2.se & S_2.p \\ \hline T.s & 1 & 0 & 0 \\ T.se & 0 & 1 & 0 \\ T.a & 0 & 0 & 0 \\ T.p & 0 & 0 & 1 \end{array}$$

Figure 1: Schema mapping and mapping matrices

the target table. A similar requirement applies when we facilitate LA rewrite rules. However, there are two differences. The selected subset of source data to be processed is in the form of matrices, and the operators to be decomposed are linear algebras. Thus, in Amalur we encode the column matching and row matching information also as matrices, i.e., *mapping matrices* and *indicator matrices*. Fig. 1 depicts an example of source schema (S_1 and S_2) and target schema T with user-selected columns. The corresponding schema mapping m is specified as a *tuple generating dependencies (tgd)*. m is also a full tgd, since all attributes of T are from S_1 and S_2, and m has no existentially quantified variables. Given m, Amalur generates the corresponding mapping matrices (M_1, M_2), which can be directly computed with the subset of source data in the matrix form. For row matching, we adapt and extend indicator matrices [1] to store the existing record linkage results.

Linear algebra rewrite opportunities. We simplify the in-lake processing of a linear algebra operator (part of an ML algorithm) in two steps. First, with the help of the aforementioned two types of matrices, we decompose and rewrite the LA operators, and push them down into individual source tables and compute the local LA results. Second, to compute the final result, we merge the local results and "stitch" them back as if the original LA operator was executed over a single target table. Notably, although such a procedure shares similarities with view-based query rewriting, its LA-based nature calls for new rewriting algorithms. Future challenges include (but are not limited to): 1) Existing LA rewriting rules (for both linear and non-linear ML models) are mainly developed based on joining two or multiple tables, and mainly equi-joins. Novel efficient algorithms are needed to tackle more complicated data integration scenarios. 2) The topic of data integration has well-studied systematic and theoretical results (e.g., schema mapping, query reformulation), leaving a large body of theoretical blank space to be filled for LA processing. 3) To support scalable ML, cost estimation is essential, which decides *when* the factorization should be performed.

REFERENCES

[1] L. Chen, A. Kumar, J. Naughton, and J. M. Patel. Towards linear algebra over normalized data. *PVLDB*, 10(11), 2017.