Optimizing Across Relational and Linear Algebra in Parallel Analytics Pipelines

Asterios Katsifodimos
TU Delft

Talk Abstract

Advanced data analysis typically requires some form of preprocessing in order to extract and transform data before processing it with machine learning and statistical analysis techniques. Pre-processing pipelines are naturally expressed in dataflow APIs (e.g., MapReduce, Flink, etc.), while machine learning is naturally expressed in linear algebra with iterations, using programming abstractions such as R's Dataframe or Python's Pandas.

Data scientists nowadays perform scalable end-to-end data analysis programs by either $i)$ using parallel dataflow APIs for the complete program, which introduce impedance mismatch hinder programmers' productivity or by $ii)$ using multiple programming paradigms (e.g., SQL and Dataframes) and systems (e.g., Hadoop and Python Pandas). However, using multiple paradigms and systems prevents optimization opportunities such as parallelization, sharing of physical data layouts (e.g., partitioning) and data structures, among different parts of a data analysis program.

My talk will be split in two parts. In the first part I will briefly introduce Lara [3], a deeply embedded language in Scala, which extends Emma [1, 2], in that it enables authoring scalable programs using two abstract data types, namely `DataBag` and `Matrix`. Lara aims at alleviating the impedance mismatch of authoring linear algebra operations on top of collection-processing APIs. Instead, it allows programmers to author complete end-to-end programs in "native" syntax that can be automatically parallelized. Most importantly, Lara enables joint optimizations over both relational and linear algebra.

In the second part of my talk I will discuss a concrete optimization which can be applied in the context of analysis programs comprising both linear and relational algebra operations. More specifically, I will present BlockJoin [4], a distributed join algorithm which emits block-partitioned results to subsequent linear algebra operations such as matrix multiplications. BlockJoin applies database techniques known from columnar processing, such as index-joins and late materialization, in the context of parallel dataflow engines, in order to minimize very expensive shuffling costs, achieving up to 6x speedups.

References

[1] A. Alexandrov, A. Kunft, A. Katsifodimos, and V. Markl et. al. Implicit parallelism through deep language embedding. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015.

[2] A. Alexandrov, A. Salzmann, G. Krastev, A. Katsifodimos, and V. Markl. Emma in action: Declarative dataflows for scalable data analysis. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2016.

[3] A. Kunft, A. Alexandrov, A. Katsifodimos, and V. Markl. Bridging the gap: Towards optimization across linear and relational algebra. In *Proceedings of the 3rd ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*. ACM, 2016.

[4] A. Kunft, A. Katsifodimos, S. Schelter, T. Rabl, and V. Markl. Blockjoin: Efficient matrix partitioning through joins. *Proceedings of the VLDB Endowment*, 10(13), 2017.