Towards Unifying Query Interpretation and Compilation

Philipp M. Grulich[1] Aljoscha Lepping[1] Dwi Prasetyo Adi Nugroho[1] Bonaventura Del Monte[1]
Varun Pandey[1] Steffen Zeuch[1,2] Volker Markl[1,2]
[1]Technische Universitat Berlin, Germany [2]DFKI GmbH, Germany

Engineering high-performance query execution engines is a very challenging task as system engineers have to balance performance and productivity. On the one hand, engines have to provide high performance across diverse data processing workloads [1]. These may go beyond purely relational operators and involve stream processing, machine learning, and user-defined functions. On the other hand, engines have to provide high productivity for system engineers to enable the timely integration of new features. To this end, execution engines have to be easy to modify, test, and debug [1]. As a result, it is crucial to choose a suitable query execution engine architecture.

Over the last decade, *vectorized query interpretation* [2] and *query compilation* [7] have emerged as state-of-the-art architectures for high-performance query execution engines. Vectorized query interpretation extends the traditional Volcano processing model and passes vectors of records between precompiled operators that can be developed in ordinary imperative code. In contrast, *query compilation* translates queries into specialized code at runtime. This improves data/code locality and enables optimal query execution performance but introduces a high query startup time and system complexity. This is particularly evident from recent commercial systems, like Photon, Firebolt, and Velox, which tend to neglect this architecture. They argue that interpretation-based engines are much easier to develop, test, and debug [1, 8]. In the following, we discuss two fundamental challenges that hinder the adoption of query compilation in detail.

Challenge 1: Managing high engineering effort. As with any other software artifact, developing, debugging, and maintaining query execution engines requires developer time and costs. In contrast to interpretation-based engines, query compilers generate operator code after query submission, i.e., at runtime. This introduces a gap between implementation and execution, which makes the engines hard to build, debug, and maintain. For instance, the engineers at Databricks reported that developing a query compiler caused eight times higher engineering costs compared to an interpretation-based engine [1]. Furthermore, state-of-the-art query compilers use code generation frameworks such as LLVM [7] or build custom compilers [3, 6]. Using these compiler frameworks in an engine requires a deep understanding of compiler technology. In particular, they use internal representations that are close to assembly, which are, even for experienced engineers, cumbersome to analyze and debug. As a result, it becomes challenging to find suitable engineers that have expertise in both compiler technology and data management [8].

Challenge 2: Navigating a large design space. Modern data processing systems support an increasingly diverse set of data processing workloads. To this end, research introduced specialized query compilers for short-running queries [3, 6], stream processing [4], or user-defined functions [5]. These compilers follow different software architectures that make different trade-offs between compilation time, execution performance, and developer experience. However, there is currently no query compiler available that is suitable for a wide range of workloads. As a result, engineers must develop compilation-based engines from scratch and reinvent solutions for reoccurring tasks like the integration of compiler frameworks. Furthermore, system engineers must develop and maintain different query compilation backends to efficiently support diverse workloads. As a result, the selection of a suitable query compilation approach depends on specific workload requirements and remains challenging. Due to both challenges, recent commercial systems often follow interpretation-based architectures to reduce development costs. Similarly, most research groups cannot afford such engineering efforts. As a result, the consensus seems to emerge that although compilation-based engines offer optimal execution performance, vectorized-based engines are easier to adopt as they are easier to develop and maintain.

To tackle the challenges, we currently investigate novel approaches for the development of execution engines that provide the advantages of query compilation without sacrificing developer productivity. In particular, we focus our research on three aspects: First, we plan to revisit the implementation interface for data processing operators in query-compilation-based engines. Such an interface has to hide the complexity of code generation and enable engineers to write imperative code that is easy to develop, debug, and maintain. Second, we plan to investigate compiler techniques, such as tracing and partial evaluation, to translate imperative operators to a unified intermediate representation. This enables the engine to specialize query execution towards particular workload requirements, e.g., minimizing startup latency or maximizing execution performance. Third, we plan to provide the components of our query execution engine as extendable and composable modules. This will enable to build or extend data processing systems with our research outcome. Finally, we intend to use the resulting query compiler as the foundation for our data processing system NebulaStream [9].

REFERENCES

[1] Alexander Behm et al. 2022. Photon: A Fast Query Engine for Lakehouse Systems. In *SIGMOD*.
[2] Peter A Boncz et al. 2005. MonetDB/X100: Hyper-Pipelining Query Execution.. In *CIDR*.
[3] Henning Funke et al. 2022. Low-latency query compilation. *The VLDB Journal* (2022).
[4] Philipp M. Grulich et al. 2020. Grizzly: Efficient stream processing through adaptive query compilation. In *SIGMOD*.
[5] Philipp M. Grulich et al. 2021. Babelfish: Efficient execution of polyglot queries. *VLDB*.
[6] Timo Kersten et al. 2021. Tidy Tuples and Flying Start: fast compilation and fast execution of relational queries in Umbra. *The VLDB Journal* (2021).
[7] Thomas Neumann. 2011. Efficiently compiling efficient query plans for modern hardware. *VLDB*.
[8] Mosha Pasumansky et al. 2022. Assembling a Query Engine From Spare Parts. *VLDB* (2022).
[9] Steffen Zeuch et al. 2019. The NebulaStream Platform: Data and Application Management for the Internet of Things. In *CIDR*.