Technical Perspective on 'BOSS - An Architecture for Database Kernel Composition'

Carsten Binnig
TU Darmstadt & DFKI

Composability is becoming an important characteristic of modern database systems, enabling flexible integration of storage, processing, and acceleration components. This shift allows specialized accelerators, various specialized storage backends, and different query execution engines for CPUs and GPUS to seamlessly work together to execute a workload. By embracing composability, database systems can better ensure adaptability in an increasingly complex data landscape rather than developing a zoo of specialized database systems — one for each workload. The VLDB 2024 paper 'BOSS - An Architecture for Database Kernel Composition' [3] is one of the first papers that tackles the composability of data systems with low added overhead and a holistic view of the full stack.

The BOSS Idea. Traditional database systems struggle with composability due to the impedance mismatch between different query processing models, data formats, and execution strategies. While specialized systems such as Apache Arrow (storage), Velox (CPU execution), and ArrayFire (GPU acceleration) offer efficient solutions within their domains, integrating them into a unified system typically requires extensive glue code, data transformation, and runtime overhead. BOSS addresses these challenges by designing a framework which comes with two key ideas: (1) queries are incrementally evaluated in stages (called Partial Query Evaluation), and (2) data is exchanged without copies (called Zero-Copy Data Transfer), ensuring high performance and flexibility.

Partial Query Evaluation. The partial query evaluation model in BOSS transforms query execution into a pipeline of processing stages, where each engine evaluates as many operations as it can efficiently execute while leaving the remaining work for subsequent stages. Instead of enforcing a monolithic execution model, queries are passed through a sequence of transformations, progressively reducing their complexity until they are fully evaluated. This approach allows each kernel to exploit its unique optimizations; e.g., a GPU kernel can perform fast parallel filtering and arithmetic operations, while a CPU kernel can handle complex joins and aggregations. The execution model ensures that if a kernel cannot process a particular query component, it simply forwards it to the next stage in the pipeline without breaking execution semantics.

Zero-Copy Data Transfer. To support efficient data transfer between components without heavy data transformations and data copies, BOSS introduces a unified in-memory representation. Typically, database systems suffer from performance bottlenecks when transferring data between different execution models due to the need for format conversions (e.g., from row-based to column-based layouts) or explicit serialization and deserialization. BOSS overcomes this by defining a shared data representation compatible with multiple processing paradigms. BOSS builds on the fact that data in many engines already uses physically the same in-memory data structures (i.e., C-arrays) and suggests Spans, lightweight wrappers around these in-memory data structures, to allow different kernels to access data directly without copying. Spans also include a reference management mechanism to ensure safe memory reuse, avoiding the complexities of garbage collection.

Other Directions in the Community. The importance of composability is also shown by the fact that the larger community and industry is working on related ideas and the fact that there are even workshops just dedicated to this topic[1]. One direction is open table formats such as Apache Iceberg[2], Apache Parquet [3], etc., which allow data engines to work on the same files, making query engines replaceable. Moreover, Apache Calcite [4] or the Postgres Query Compiler and Optimizer are examples of query frontends that are often reused for building a new database engine. Furthermore, as part of cloud data systems, which already today separate query compilation from query execution and storage, there are many other trends of breaking data systems down further into sub-components and separating the indexing layer as a separate service [1] or providing data shuffling as a service [2][5] that can be used by different distributed data systems.

Open Challenges and Impact. While composability is clearly becoming more and more an important topic, as outlined above, many challenges remain. Even when focussing only on tabular data and SQL, it is not really clear if composability will ever become true. One particular challenge is the semantic mismatch between different components, which mainly stems from the fact that different SQL query engines implement SQL as a query language very differently. For example, type inference or rounding numbers work very differently in different engines. As such, when running the same SQL query in different engines, different results might be produced, which is somewhat of a deal-breaker for composable systems. While clearly composability in parts (e.g., open data formats) is already today a huge success, it remains unclear if composability across the stack will ever happen – while being a great idea.

References

[1] Peter A. Boncz, Yannis Chronis, Jan Finis, Stefan Halfpap, Viktor Leis, Thomas Neumann, Anisoara Nica, Caetano Sauer, Knut Stolze, and Marcin Zukowski. 2023. SPA: Economical and Workload-Driven Indexing for Data Analytics in the Cloud. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 3740–3746. doi:10.1109/ICDE55515.2023.00302

[2] Matthias Jasny, Lasse Thostrup, Sajjad Tamimi, Andreas Koch, Zsolt István, and Carsten Binnig. 2024. Zero-sided RDMA: Network-driven Data Shuffling for Disaggregated Heterogeneous Cloud DBMSs. *Proc. ACM Manag. Data* 2, 1 (2024), 36:1–36:28. doi:10.1145/3639291

[3] Hubert Mohr-Daurat, Xuan Sun, and Holger Pirk. 2023. BOSS - An Architecture for Database Kernel Composition. *Proc. VLDB Endow.* 17, 4 (2023), 877–890. doi:10.14778/3636218.3636239

[1] https://cdmsworkshop.github.io/
[2] https://iceberg.apache.org/
[3] https://parquet.apache.org/
[4] https://calcite.apache.org/
[5] https://cloud.google.com/blog/products/data-analytics/how-distributed-shuffle-improves-scalability-and-performance-cloud-dataflow-pipelines