Photon: A High-Performance Query Engine for the Lakehouse

Alexander Behm
alex.behm@databricks.com
Databricks Inc.

ABSTRACT

We present Photon, a new native vectorized query engine powering the Databricks Runtime. Photon achieves state-of-the-art query execution times and industry-leading price performance on real workloads against data stored in the Parquet and Delta format, in situ over data lakes such as Amazon S3, ADLS, and GCS. Photon supports not only SQL but also the DataFrame APIs of Apache Spark [3], PySpark, and Koalas.

We developed Photon in response to the observation that organizations are shifting to a new paradigm called the *Lakehouse* [2], which combines the low cost, open access, and versatility of data lakes with the reliability, performance, and governance features of data warehouses. One reason for this trend is that the majority of enterprise data today (over 90% of bytes, in our experience) is already stored in data lakes. Users have reported that it is time-consuming and error-prone to copy and maintain subsets of this data in an external data warehouse while maintaining data freshness and accuracy. Instead, the Lakehouse enables applications to operate on the data in-situ throughout the data lifecycle (e.g., from raw ingested values to bronze/silver/golden) without compromising on fidelity or performance. The full data lineage can be tracked and errors can be corrected from the raw source data.

While in-situ processing simplifies the data architecture, it also challenges the data processing engine with a greater variety of data. On one end of the spectrum, users take great care to clean and organize their data for read performance by designing schemas and data indexes. On the other end of the spectrum, uncurated data may have suboptimal data layouts like small files, many columns, sparse or large data values, and no useful clustering or statistics. In addition, strings are convenient and prevalent, even to represent numeric data like integers and dates. Data is un-normalized, so string columns may additionally use placeholder strings for unknown or missing values instead of NULL. Schema information like NULL-ability or string encoding (e.g., ASCII vs. UTF-8) is typically absent.

These differences reflect that not all data is equally valuable, so simply optimizing all data is neither a cost-efficient practice nor a desirable outcome. Thus, our goal with Photon is to design an engine that is flexible enough to deliver good performance on arbitrary uncurated data, and excellent performance on data following Lakehouse best practices—like Hilbert clustering, reasonable file sizes, and appropriate data types—across a variety of use cases such as data science, ETL, ad hoc SQL, and BI.

We present our decisions with respect to Photon's design and implementation. First, we chose to design the engine around vectorized-interpreted query execution in lieu of code generation. Although we found some scenarios where the code generation model delivers better performance (e.g., queries with complex conditional expressions), our experience while prototyping both approaches and from working on past database engines was that the vectorized model was easier to build, profile, debug, and operate at scale. This allowed us to invest more time in specializations that narrowed or eliminated the performance gap between the two. Preserving abstraction boundaries such as query operators also facilitates collecting rich metrics to help end users better understand query behavior. Finally, vectorized execution enabled us to support runtime adaptivity, wherein Photon discovers, maintains, and exploits micro-batch data characteristics with specialized code paths to adapt to the various degrees of data quality in the Lakehouse for optimal performance.

Second, we chose to implement Photon in a native language (C++) rather than following the precedent of the existing Databricks Runtime engine, which is implemented in the JVM. One reason for this decision was that we were hitting performance ceilings with the existing JVM-based engine. For example, implementing operators that used SIMD instructions in the CPU or prefetched data from main memory was difficult without deep knowledge of the JVM compiler internals. Another reason for switching to native code was internal JIT compiler limits (e.g., on method size) that created performance cliffs when JVM optimizations bailed out. Finally, we found that the performance of native code was generally easier to explain than the performance of the JVM engine, since aspects like memory management were under explicit control.

Finally, Photon integrates with the Apache Spark-based Databricks Runtime to coordinate work and resources. Users can accelerate their existing workloads without code changes due Photon's hybrid execution capability. Queries can partially run in Photon and fall back to Apache Spark for still unsupported operations, while Photon features are being continuously added to reduce these transitions. This ability to partially roll out Photon has given us valuable operational experience in using Photon in the field.

Since its release, Photon has run tens of millions of production queries issued by hundreds of enterprise customers. Thus far, our customers have reported speedups of up to an order of magnitude with an average of 3× over the existing Databricks Runtime, which in turn provides significant performance improvements over open-source Apache Spark. As further validation, Databricks set an audited 100TB TPC-DS world record [1] in November 2021 with Photon and a Lakehouse system using the Delta Lake format on Amazon S3, showing that state-of-the-art SQL performance is attainable with open data formats and commodity cloud storage.

REFERENCES

[1] 2021. TPC-DS V3 Result Highlights, Databricks SQL 8.3. http://tpc.org/5013.

[2] Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. 2021. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. CIDR.

[3] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2–2.