The Gluten Open-Source Software Project: Modernizing Java-based Query Engines for the Lakehouse Era

Akash Shankaran[1,†], George Gu[2,†], Weiting Chen[2,†], Binwei Yang[3,†], Chidamber Kulkarni[4,†], Mark Rambacher[5,†], Nesime Tatbul[5,†] and David E. Cohen[5,*,†]

[1]Intel, Seattle, Washington, USA
[2]Intel, Shanghai, China
[3]Intel, Portland, Oregon, USA
[4]Intel, Vancouver, Canada
[5]Intel, Boston, Massachusetts, USA

Abstract

Year-on-year, exponential data growth, and the corresponding growth in machine learning's appetite to process that data is transforming the industry's data management discipline. In response, the data lakehouse architecture has emerged. The transformative nature of the lakehouse architecture and the need to enable a diverse set of query engines to access data that resides in a lakehouse is motivating a refactoring of capabilities in these query engines. Industry's response is the composable data management system (CDMS). This paper introduces the Gluten open-source software (OSS) project – an embodiment of the CDMS concept. Gluten is a Java Native Interface (JNI) bridge that enables Java-based query engines to offload/accelerate processing to native acceleration libraries, such as the Meta-led Velox OSS project.

Keywords

Spark-SQL, Gluten, Velox, Substrait

1. Introduction

This paper introduces the open-source software (OSS) project, Gluten [1], a Java Native Interface (JNI) based bridge between query engines written in Java and database acceleration libraries such as the Velox OSS project. Query engines that integrate Gluten embody the composable data management system (CDMS) concept. Currently, Gluten uses the Substrait.io OSS project [2] to enable the Spark-SQL query engine to employ the Velox acceleration library [3]. Work is under way to generalize the approach so that any query engine that incorporates Gluten can generate a query plan use Substrait to transform this plan to a canonical form. A transformation is provided that maps the canonical plan onto the targeted acceleration library's plan (e.g. a Velox plan). Execution of the plan is then offloaded to the library.

Although Gluten is intended to apply to any SQL query engine written in Java, initial work focuses on Apache Spark and that framework's Spark-SQL Java application [4]. Our work on Spark-SQL-over-Gluten serves as the motivating scenario for this paper. This work includes integration of the Substrait and Velox OSS projects into Gluten. With Substrait mappings of the Spark-SQL and Velox plans in place, the integration of Gluten has transformed Spark's SQL query engine into a CDMS implementation. This effort is early in its development, but is already producing competitive results in TPC-H/TPC-DS-like characterizations. This development effort and characterization work is covered in the section entitled "Spark-SQL-over-Gluten."

Concretely, the contributions of this paper are to provide background on the motivation for the Data Lakehouse, the technical architecture that has emerged, and the disruption adoption of this architecture is having on big data processing. No where is this disruption more evident than amongst the largest users of Spark-SQL. Insights into the Spark-SQL market are discussed along with how the leaders of Spark project have leveraged the Data Lakehouse architecture to their advantage. This, in turn, has served as a catalyst for introducing composability not just for Spark-SQL but to the broader set of Java-based query engines. Spark-SQL is used to illustrate the mechanics of composability, including early experimental results, Finally, the paper provides thoughts on how this composability can be extended to embrace the coming wave of heterogeneous processors and memories.

Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — Second International Workshop on Composable Data Management Systems (CDMS'23), August 28 - September 1, 2023, Vancouver, Canada
*Corresponding author.
†These authors contributed equally.
akash.shankaran@intel.com (A. Shankaran); george.gu@intel.com (G. Gu); weiting.chen@intel.com (W. Chen); binwei.yang@intel.com (B. Yang); chidamber.kulkarni@intel.com (C. Kulkarni); mrambach@gmail.com (M. Rambacher); nesime.tatbul@intel.com (N. Tatbul); david.e.cohen@intel.com (D. E. Cohen)

2. Background

Computing demand continues to grow exponentially, largely driven by "big data" processing on hyperscale data stores [5]. Increasingly, this data processing is in support of machine learning (ML), training models and subsequently serving up these models, to personalize digital content in an eCommerce setting, for example. This ML-centric, big data processing increasingly operates over custom, heterogeneous processors such as GPUs, TPUs, FPGAs, etc [6]. Taken together, these forces are motivating dramatic changes in the data management process of large companies.

2.1. The Emergence of the Data Lakehouse Architecture

The result of these changes has been the emergence of the Data Lakehouse architecture, a combination of traditional data warehouse functionality and more modern data lakes. A data lake stores raw, unstructured data on disaggregated storage. In contrast, a data warehouse is a repository for structured, filtered data that has already been processed for a specific purpose. In the emerging Lakehouse approach, an open table format is introduced into the data lake architecture. This enables the unbundling of the query engine from the data management facilities of the warehouse. These data management capabilities are refactored to operate over this open table abstraction. Introduction of these capabilities into the data lake is transformative, resulting in the Lakehouse [7].

What are the data management capabilities of a Lakehouse deployment? First and foremost is the disaggregation of log-structured storage from the servers over which processing is carried out [8]. This disaggregation allows for the independent scaling of the storage and compute resources. Second is the ingestion of virtually any type of data into the Lakehouse using a supported serialization format, for example the Apache Parquet [9]. Notably, the tables of a Lakehouse are mutable [10], [11], [12], [13], [14]; allowing for transactional updates, schema modifications, etc. From within the Lakehouse, this data is then projected into analytic services such as SQL query engines, search systems, stream processors, query editors, notebooks, and machine learning (ML) models through direct access, real-time, and batch workflows.

2.2. The Impact of the Lakehouse Architecture on Spark-as-a-Service Deployments

The Apache Spark open-source software (OSS) project started as a research project at the University of California - Berkeley AMPLab in 2009, and was open sourced in early 2010. In 2013, Matei Zaharia and several others from AMPLab founded Databricks whose charter is to provide a Cloud-only, Software-as-a-Service (SaaS) platform for working with Spark. Today, the Databricks service is globally available, running over Infrastructure-as-a-Service (IaaS) platforms operated by Alibaba Cloud, Amazon, Google, and Microsoft Azure [15].

In parallel, these same hyperscale companies operate their own Spark-SQL-as-a-Service offerings. Each of these offerings operate over their respective global infrastructure while Databricks is platform agnostic. This gives rise to a competitive environment across all participants.

2.3. The Databricks Lakehouse Changes the Competitive Landscape

The Apache Spark OSS project has historically been an engine of innovation that benefited the community as a whole. All of the Spark-SQL-as-a-Service offerings were based on this upstream Spark-SQL codebase. With the advent of the Lakehouse, Databricks has thrown down the gauntlet at its competition in this market segment.

Their Deltalake [12] initiative has been instrumental in bringing the Lakehouse architecture to market. As part of this initiative, Databricks introduced a proprietary Photon database acceleration library; giving them performance and efficiency advantages and changing the dynamics of the Spark community. In response to this competitive threat, others in this segment are motivated to find an alternative to the Databricks Photon library [16].

2.4. Private Offerings of Spark-SQL-as-a-Service

In addition to the Spark-SQL-as-a-Service market, some of the largest Cloud companies operate private Spark-SQL-as-a-Service offerings for internal constituents. These companies include ByteDance, eBay, JD.Com, LinkedIn, Maituan, Netflix, Pinterest, Stripe, etc.

The scale of these offerings has reached a point where they are motivated to find ways to realize economies-of-scale. The performance and efficiency gains afforded by the use of the Photon library is compelling. However, its proprietary nature is counter to their objectives. This is motivating interest across this segment in an open-source alternative to the Photon library.

2.5. Meta Deprecates Spark-SQL

Finally, Meta has been a driving force in the Spark community for many years [17], [18]. Recently, however, they have begun deprecating the use of Spark-SQL with PrestoDB's SQL taking its place. Meta is standardizing

on PrestoDB's SQL via the use of the Velox database acceleration native code library OSS project and that library's CoreSQL dialect [19]. PrestoDB-on-Spark refactors PrestoDB's functionality as a client library, similar to Google's F1 Query client library [20], [21].

3. Spark-SQL on Gluten

3.1. Offloading Spark Processing to a Native Database Accelerator

For analytical and machine learning workloads, the design of modern query engines is dominated by on-disk (e.g. Apache Parquet) and in-memory (e.g. Apache Arrow), columnar serialization formats [22], [9], [23]. While these workloads are memory/memory bandwidth bound, Spark workloads have become CPU-bound. Three companies realized the opportunity to transform Spark into a vectorized SQL engine and break through to its row-based data processing and JVM limitations. Today, Databricks, Intel, and NVIDIA each develop and maintain JNI-based database acceleration implementations that enable Spark-SQL to offload/accelerate Java code to a C++ library. These are the Photon, the Gluten, and the Spark-Rapids implementations respectively. Of these, only Gluten is an OSS project.

In the same timeframe that Databricks and NVIDIA were developing their solutions, Intel's Spark team was working on the Gazelle project, a predecessor to Gluten [24]. The Gazelle project focused on enabling Spark to exploit single instruction, multiple data (SIMD), specifically Intel's Advanced Vector Extensions (Intel AVX) technology. A key deficiency of Gazelle was its limited community participation. This meant that the development burden fell to Intel.

In the meantime, several vectorized SQL engines emerged with more active open-source communities. Among these, the Meta-led Velox project is a rising star, providing a vectorized database acceleration library [3]. While these vectorized engines are popular, prior to Gluten there was no support for an open-source software option in Apache Spark. Intel has adopted the Velox native OSS library to replace its own Gazelle library. Adoption of Velox has opened Gluten up to the larger, more vibrant Velox community of developers. The integration of the Gluten JNI bridge with Spark-SQL retains much of the Spark-SQL Executor's Java-based implementation. In contrast, Meta's PrestoDB-on-Spark implementation replaces Spark-SQL with a new Presto native C++ implementation that incorporates the Velox library. This new C++ SQL engine is then integrated with the Spark execution framework.

3.2. Gluten Implementation

As shown in Figure 1, Spark-SQL-on-Gluten replaces Gazelle with the Velox database acceleration library. The approach is similar to the one taken by Databricks in their Photon native library. The clear difference is software licensing. The Photon library is proprietary and only available with the Databricks Spark-as-a-Service platform. Gluten, on the other hand, is an Apache OSS licensed project. Gluten depends on OSS licensed projects such as the Apache Arrow, Substrate, and Velox projects. What follows is a brief sketch of the Spark-SQL-on-Gluten implementation.

3.2.1. Plan Conversion

Gluten uses Substrait to build a query plan tree. It converts Spark's physical plan to a Substrait plan for the targeted backend, and then shares the Substrait plan over JNI to trigger the execution pipeline in the Velox native library.

3.2.2. Fallback Processing

Gluten leverages the existing Spark JVM engine to check that an operator is supported by the native library. If not, Gluten falls back to the existing Spark-JVM-based operator. This fallback mechanism comes at the cost of column-to-row and row-to-column data conversions between the memory layouts of the two environments.

3.2.3. Memory Management

Gluten leverages Spark's existing memory management system. It calls the Spark memory registration API for every native memory allocation/deallocation action. Spark manages the memory for each task thread. If the thread needs more memory than is available, it can call the spill interface for operators that support this capability. Spark's memory management system protects against memory leaks and out-of-memory issues.

3.2.4. Columnar Shuffle

Gluten reuses its predecessor Gazelle's Apache Arrow-based Columnar Shuffle Manager as the default shuffle manager. A third-party library is responsible for handling the data transformation from native to Arrow. Alternatively, developers are free to implement their own shuffle manager.

3.2.5. Metrics

Gluten supports Spark's Metrics functionality. The default Spark metrics are served for Java row-based data

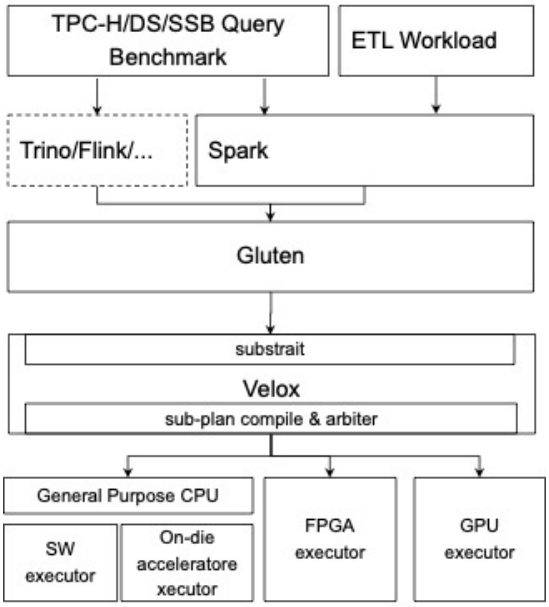

Figure 1: The Gluten Software Stack

Name	Hardware Platform
CPU Model	Intel® Xeon® Platinum 8480+
Micro-architecture	Sapphire Rapids
CPUs	224
Memory	1024GB
NIC	1x Ethernet Controller I225-LM 1x Ethernet interface
Disks	2x 1.5T INTEL SSDPE2KE016T8 1x 447.1G INTEL SSDSC2BB48 1x 447.1G INTEL SSDSC2KB48 7x 3.5T INTEL SSDPF2KX038TZ

Table 1
Hardware Configuration

Name	Software Platform
Operating System	Ubuntu 22.04.1 LTS
Linux Kernel	5.16.0-051600rc5-generic
JDK version	1.8
GCC version(Gluten only)	11
Spark version	3.3.1
Hadoop version	3.2

Table 2
Software Configuration

processing. Gluten includes additional metrics to provide developers a means of debugging the targeted native database acceleration library.

3.2.6. Shim Layer

To fully integrate with Spark, Gluten includes a shim layer whose role is to support multiple versions of Spark. Gluten supports Spark versions 3.2 and 3.3, with newer version support being added..

4. Comparative Performance Characterization

This section provides a comparative characterization of "Spark-SQL without Gluten" and "Spark-Gluten-Velox," executing on the latest Intel processor. Efforts to optimize Java-based query engines rely on JVM/JDK packages, whose capabilities differ from version to version. For example, JDK 17 includes a SIMD-based Vector API capability that enables efficient query engine vectorization. Widely used earlier versions such as JDK11 or JDK8 are missing this Vector API capability. Use of Gluten removes this JVM/JDK version dependency when optimizing Java-based query engines. These results demonstrate the performance benefits of offloading Spark-SQL processing to Gluten.

Two benchmarks (TPC-H-like and TPC-DS-like) are used to evaluate the performance of Gluten compared to Vanilla Spark which derives from TPC-H and TPC-DS benchmark with minor changes to accommodate Gluten and Velox implementations. The results show a significant improvement by using Gluten and Velox. In Figure 2, the result shows that Gluten outperforms Spark-SQL by 2.71X in the TPC-H-like characterization and by 2.29X in the TPC-DS-like characterization. As references, the Hardware and Software Configurations are listed in Table 1 and Table 2 respectively.

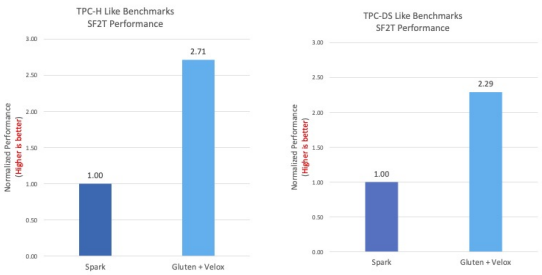

Figure 2: Comparative Characterization

The improvement can also be observed from CPU processor micro-architecture perspective. Figure 3 illustrates Gluten instruction path length reduces by 3.7X in the TPC-H-like query and by 2.5X in TPC-DS-like query against Spark-SQL. Gluten + Velox can also unleash the power of Intel AVX technology using SIMD instructions

within a vectorized SQL engine to break through the original row-based data processing and JVM limitations.

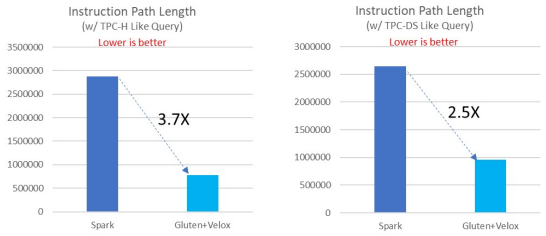

Figure 3: Instruction Path Length Comparison

Thanks to columnar based shuffle in Gluten, additional benefits observed are the size reduction of shuffle write and taking advantage of smaller packets in the network during spark shuffle phase. In both benchmarks, around 15-25 percent shuffle size gets reduced (Figure 4), which helps enhance the network utilization and improves performance in spark shuffle. Besides size reduction, columnar based shuffle can also take the advantages from the compression codec. Users can choose a suitable codec based on individual columnar data type to get higher compression ratio with better performance.

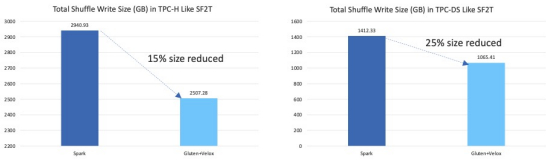

Figure 4: Shuffle Comparison

5. Roadmap and Future Work

We conclude the paper by sharing three key elements of the Gluten roadmap: (i) formalizing the use of the Substrait.io project, (ii) generalizing Gluten for use across several query engines, and (iii) enabling these Gluten-based query engines to target heterogeneous hardware.

5.1. Formalizing the Use of the Substrait.io Project

One method of vertical integration is to replace the top portion of the query engine framework [25]. This includes all of the frontend components: the user-facing interfaces, the query plan/optimizer, the distributed execution framework, etc. An adapter is introduced that allows these components to be replaced by a proxy. The new framework takes a query as input, produces a canonical plan, and then maps that plan onto the plan of various query engines. Each query engine provides an adapter that implements the transformation of the canonical plan to the query engine's plan. The query engine remains unchanged. Gluten is orthogonal to this approach as it operates on the plan produced by the top portion of the query engine framework. Composability at Gluten's level requires taking the optimized plan from the optimizer and mapping it on to heterogeneous hardware.

Analytical query engines targeted by Gluten have an internal schema that represents the planner/optimizer's relational algebra. The operators, functions, data structures, and data types represent the internal implementation of the engine. Vertical composability requires that this representation be externalized so that a plan instance produced by the optimizer can be transformed to canonical form. Gluten enlists the Substrait.io project to provide this function. Substrait defines a "Cross-Language Serialization for Relational Algebra," providing the methods to transform to/from its algebra.

Currently, Substrait mappings exist from the PrestoDB and Spark-SQL query engines. Ideally, a public, open-source repository of these mappings would be available in conjunction with the Substrait project. Currently, however, these mappings are internal to the PrestoDB and Gluten projects. Given such a repository, Substrait could define an application binary interface (ABI) analogous to the one provided by the Apache Arrow project. This ABI can then be supported by various native acceleration libraries. For example, support for the Substrait mappings provided by PrestoDB and Gluten is embedded in the Velox library. A Substrait ABI would allow formal support for Substrait in Velox. The mappings and their transformations could then be provided as external, shared libraries.

5.2. Gluten as a JNI Bridge to Database Acceleration Libraries for Any Java Query Engine

The Gluten JNI implementation is designed to support multiple database acceleration libraries. Currently, Kyligence provides a ClickHouse library while Intel employs the Velox library [26], [27]. The Kyligence implementation does not currently take advantage of the Substrait transformations. Support for Substrait is included in the Velox project. However, the lack of a Substrait ABI means the to/from Substrait transformations for PrestoDB and Spark-SQL are internal to Velox. Support for a Substrait ABI will allow for a mapping to provide a query-engine-specific Substrait schema, a shared library, and a means of registering the schema with Velox as part of the Velox initialization.

In Gluten's case, the framework is being refactored as a general Java-Native-Interface (JNI) implementation

that uses the Substrait algebra to map a Java-based query engine such as PrestoDB or Spark-SQL on to a native database acceleration library.Gluten supports the Kyligence and Spark-SQL query engines. Work is now underway to enable the Trino project to integrate Gluten and integration with Apache Flink is in the Gluten backlog [28], [29]. Trino and Flink will provide Subtstrait schemas and it is hoped that Kyligence will add support for Substrait to their roadmap.

5.3. Enabling Gluten to Target Heterogeneous Processors

The Gluten abstraction also affords the opportunity to target heterogeneous processors via the native accelerator library. For example, the Velox library is being extended to target heterogeneous hardware accelerators which may be based on General Purpose CPU, FPGA or GPU. The diverse heterogeneous accelerators could become out-of-box components for CDMS systems to achieve significant advantages on design flexibility, system elasticity, performance and power efficiency (Figure 1).

Currently, Velox provides a vectorization engine that targets general purpose CPUs: x86, IA, and ARM [30]. In contrast, the PyTorch framework provides the TorchDynamo and TorchInductor sub-project that together enable deployments to target heterogeneous processors such as an inductor OpenMP backend for general purpose CPUs and an inductor Triton backend for GPUs, including NVIDIA, AMD, etc [31], [32], [33]. The proposed extension to Velox will introduce an ABI that is analogous to TorchDynamo/TorchInductor. The idea is to provide baseline support with OpenMP and Triton. The Pytorch community has investigations underway to extend this to support IA-based on-die accelerators, FPGAs, and Habana [34]. We envision Velox pursuing a similar path as PyTorch's TorchDynamo/TorchInductor approach.

6. Conclusion

This paper provided background on the motivation for the Data Lakehouse and the disruption adoption of the Lakehouse architecture is causing. No where is this disruption more apparent than amongst the largest users of Spark-SQL. This market includes Databricks, founded by the creators of the Spark project and arguably the leaders of the Data Lakehouse movement. Their introduction of the proprietary Photon database acceleration library has been a catalyst for interest in the Spark-Gluten open source software (OSS) project and its use of the Substrait and Velox OSS projects. The methods used by Gluten to enable Spark-SQL to take advantage of these projects embodies vertical composability. Early experimental results demonstrate the promise of this approach. We believe Gluten can be generalized for use by any Java-based query engine. Further, we believe the use of Substrait and Velox allow for vertical composability to be extended to encompass the underlying heterogeneous hardware that is coming online. To that end, the paper provides insights into the Gluten roadmap along with plans to work with the Subtrait and Velox community to realize the Composable Data Management System vision.

Acknowledgments

Thanks to Masha Basmanova, Orri Erling, Deepak Majeti, Pedro Pedreira, and the rest of the Velox community. Thanks also to Paul Amonson, Lukasz Grab, Milosz Linkiewicz, Kelly Mckeighan, Cezary Sawicki, and the rest of the Intel folks working on the Gluten and Velox projects. Special thanks to Jim Younan, who passed away unexpectedly at the end of last year.

References

[1] Gluten, https://github.com/oap-project/gluten, 2023. Accessed: 2023-06-28.

[2] Substrait, https://substrait.io/, 2023. Accessed: 2023-06-28.

[3] Velox, https://github.com/facebookincubator/velox, 2023. Accessed: 2023-06-28.

[4] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, M. Zaharia, Spark SQL: Relational Data Processing in Spark, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2015, pp. 1383–1394.

[5] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, P. Villalobos, Compute Trends Across Three Eras of Machine Learning, CoRR abs/2202.05924 (2022).

[6] A. Gonzalez, A. Kolli, S. M. Khan, S. Liu, V. Dadu, S. Karandikar, J. Chang, K. Asanovic, P. Ranganathan, Profiling Hyperscale Big Data Processing, in: Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA), 2023, pp. 47:1–47:16.

[7] M. Zaharia, A. Ghodsi, R. Xin, M. Armbrust, Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics, in: Proceedings of the 11th Conference on Innovative Data Systems Research (CIDR), 2021.

[8] Z. Luo, L. Niu, V. Korukanti, Y. Sun, M. Basmanova, Y. He, B. Wang, D. Agrawal, H. Luo, C. Tang, A. Singh, Y. Li, P. Du, G. Baliga, M. Fu, From Batch Processing to Real Time Analytics: Running Presto® at Scale, in: Proceedings of the 38th

IEEE International Conference on Data Engineering (ICDE), 2022, pp. 1598–1609.
[9] Apache Parquet, https://parquet.apache.org/, 2023. Accessed: 2023-07-05.
[10] Apache Hudi, https://hudi.apache.org/, 2023. Accessed: 2023-07-05.
[11] Apache Iceberg, https://iceberg.apache.org/, 2023. Accessed: 2023-07-05.
[12] M. Armbrust, T. Das, S. Paranjpye, R. Xin, S. Zhu, A. Ghodsi, B. Yavuz, M. Murthy, J. Torres, L. Sun, P. A. Boncz, M. Mokhtar, H. V. Hovell, A. Ionescu, A. Luszczak, M. Switakowski, T. Ueshin, X. Li, M. Szafranski, P. Senster, M. Zaharia, Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores, Proceedings of the VLDB Endowment 13 (2020) 3411–3424.
[13] J. Camacho-Rodríguez, A. Agrawal, A. Gruenheid, A. Gosalia, C. Petculescu, J. Aguilar-Saborit, A. Floratou, C. Curino, R. Ramakrishnan, LST-Bench: Benchmarking Log-Structured Tables in the Cloud, CoRR abs/2305.01120 (2023).
[14] P. Jain, P. Kraft, C. Power, T. Das, I. Stoica, M. Zaharia, Analyzing and Comparing Lakehouse Storage Systems, in: Proceedings of the 13th Conference on Innovative Data Systems Research (CIDR), 2023.
[15] C. Power, H. Patel, A. Jindal, J. Leeka, B. Jenkins, M. Rys, E. Triou, D. Zhu, L. Katahanas, C. B. Talapady, J. Rowe, F. Zhang, R. Draves, I. Santa, A. Kumar, The Cosmos Big Data Platform at Microsoft: Over a Decade of Progress and a Decade to Look Forward, Proceedings of the VLDB Endowment 14 (2021) 3148–3161.
[16] A. Behm, S. Palkar, U. Agarwal, T. Armstrong, D. Cashman, A. Dave, T. Greenstein, S. Hovsepian, R. Johnson, A. S. Krishnan, P. Leventis, A. Luszczak, P. Menon, M. Mokhtar, G. Pang, S. Paranjpye, G. Rahn, B. Samwel, T. van Bussel, H. V. Hovell, M. Xue, R. Xin, M. Zaharia, Photon: A Fast Query Engine for Lakehouse Systems, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2022, pp. 2326–2339.
[17] B. Chattopadhyay, P. Pedreira, S. Agarwal, Y. J. Sun, S. Vakharia, P. Li, W. Liu, S. Narayanan, Shared Foundations: Modernizing Meta's Data Lakehouse, in: Proceedings of the 13th Conference on Innovative Data Systems Research (CIDR), 2023.
[18] M. Valdez-Vivas, V. Sharma, N. Stanisha, S. Li, L. Mi, W. Jiang, A. Kalinin, J. Metzler, Clockwork: A Delay-Based Global Scheduling Framework for More Consistent Landing Times in the Data Warehouse, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2021, pp. 3627–3637.
[19] P. Pedreira, O. Erling, M. Basmanova, K. Wilfong, L. S. Sakka, K. Pai, W. He, B. Chattopadhyay, Velox: Meta's Unified Execution Engine, Proceedings of the VLDB Endowment 15 (2022) 3372–3384.
[20] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, H. Ahmadi, D. Delorey, S. Min, M. Pasumansky, J. Shute, Dremel: A Decade of Interactive SQL Analysis at Web Scale, Proceedings of the VLDB Endowment 13 (2020) 3461–3472.
[21] B. Samwel, J. Cieslewicz, B. Handy, J. Govig, P. Venetis, C. Yang, K. Peters, J. Shute, D. Tenedorio, H. Apte, F. Weigel, D. Wilhite, J. Yang, J. Xu, J. Li, Z. Yuan, C. Chasseur, Q. Zeng, I. Rae, A. Biyani, A. Harn, Y. Xia, A. Gubichev, A. El-Helw, O. Erling, Z. Yan, M. Yang, Y. Wei, T. Do, C. Zheng, G. Graefe, S. Sardashti, A. M. Aly, D. Agrawal, A. Gupta, S. Venkataraman, F1 Query: Declarative Querying at Scale, Proceedings of the VLDB Endowment 11 (2018) 1835–1848.
[22] Apache Arrow, https://arrow.apache.org/, 2023. Accessed: 2023-07-05.
[23] X. Zeng, Y. Hui, J. Shen, A. Pavlo, W. McKinney, H. Zhang, An Empirical Evaluation of Columnar Storage Formats, CoRR abs/2304.05028 (2023).
[24] Gazelle Plug-in, https://github.com/oap-project/gazelle_plugin, 2023. Accessed: 2023-07-05.
[25] H. Gavriilidis, L. Behme, S. Papadopoulos, S. Bortoli, J. Quiané-Ruiz, V. Markl, Towards a Modular Data Management System Framework, in: S. R. Valluri, M. Zaït (Eds.), Proceedings of the 1st International Workshop on Composable Data Management Systems (CDMS), 2022.
[26] Clickhouse, https://clickhouse.com/, 2023. Accessed: 2023-07-05.
[27] Kyligence, https://kyligence.io/, 2023. Accessed: 2023-07-05.
[28] Apache Flink, https://flink.apache.org/, 2023. Accessed: 2023-07-05.
[29] Trino, https://trino.io/, 2023. Accessed: 2023-07-05.
[30] T. Kersten, V. Leis, A. Kemper, T. Neumann, A. Pavlo, P. A. Boncz, Everything You Always Wanted to Know About Compiled and Vectorized Queries But Were Afraid to Ask, Proceedings of the VLDB Endowment 11 (2018) 2209–2222.
[31] OpenMP, https://www.openmp.org/, 2023. Accessed: 2023-07-05.
[32] PyTorch/Torch/Dynamo, https://github.com/pytorch/pytorch/tree/main/torch/_dynamo, 2023. Accessed: 2023-07-05.
[33] PyTorch/Torch/Inductor, https://github.com/pytorch/pytorch/tree/main/torch/_inductor, 2023. Accessed: 2023-07-05.
[34] Habana Labs, https://habana.ai/, 2023. Accessed: 2023-07-05.