# Hybrid-Cloud SQL Federation System at Twitter

Chunxu Tang, Beinan Wang, Huijun Wu, Zhenzhao Wang, Yao Li,
Vrushali Channapattan, Zhenxiao Luo, Ruchin Kabra, Mainak Ghosh,
Nikhil Kantibhai Navadiya and Prachi Mishra

*Twitter, Inc., San Francisco, United States*

**Abstract**

Twitter runs a large-scale SQL federation system to fulfill the increasing need for data analytics alongside high scalability and availability. Recently, with Twitter's efforts in migrating ad-hoc clusters to the cloud, we evolved the SQL system into a hybrid-cloud SQL federation system, across Twitter's data centers and the public cloud, interacting with around 10PB of data daily.

In this paper, we present the design of the hybrid-cloud SQL federation system, including query federation, cluster federation, and storage federation. We identify challenges in a modern SQL system and how our system helps to address them with some important design decisions. Finally, we reflect on a qualitative examination of lessons learned from the development and maintenance of such a SQL system.

**Keywords**

SQL, cloud, query engine, big data

## 1. Introduction

Twitter runs multiple large Hadoop clusters of over 300PB of data, which are among the biggest in the world [1]. Billions of events are ingested into these clusters per minute [2]. Twitter's data platform exerts significant effort in pursuing system scalability and availability to fulfill the data analytics on such large volume data inventory and high throughput data flow.

At Twitter, a typical OLAP (Online Analytical Processing) workload mainly contains ad-hoc queries, empowering a wide range of use cases from internal tooling reporting to ads click-rate analysis. A SQL system needs to be capable of processing a large number of queries in parallel. Previously, we implemented an in-house SQL system in Twitter's data center (aka private cloud) with hundreds of worker nodes, accompanied by internal Twitter services such as monitoring and logging. At present, to enhance the experience and productivity, Twitter engineering is embarking on an effort to migrate ad-hoc clusters to the GCP (Google Cloud Platform), aka the "Partly Cloudy" [3]. Partly Cloudy extends Twitter's environment into the public cloud, as a first-class offering alongside on-premises platform services.

The hybrid-cloud environment brings challenges, leading to a fundamental architectural shift for an OLAP system. From our development and operational experience, a modern unified SQL

*ECSA'21: European Conference on Software Architecture, September 13–17, 2021, Virtual, originally Växjö, Sweden*
chunxut@twitter.com (C. Tang); beinanw@twitter.com (B. Wang); huijunw@twitter.com (H. Wu); zhenzhaow@twitter.com (Z. Wang); yaoli@twitter.com (Y. Li); vrushali@twitter.com (V. Channapattan); zluo@twitter.com (Z. Luo); rkabra@twitter.com (R. Kabra); mghosh@twitter.com (M. Ghosh); nnavadiya@twitter.com (N. K. Navadiya); prachim@twitter.com (P. Mishra)



CEUR Workshop Proceedings (CEUR-WS.org)

system should handle a series of challenges:

- **Querying heterogeneous data sources in the application layer.** With the growth of the business, more use cases emerged, leading to querying heterogeneous data sources, usually processed by different on-premises or cloud query systems with different configurations and interfaces. For example, data scientists from the Health team query data stored in HDFS (Hadoop Distributed File System), processed by HDFS-compatible SQL engines such as Hive [4], SparkSQL [5], and Presto [6], to analyze hate speech in the social media platform. Data engineers from the Ads team query data stored in GCS (Google Cloud Storage), processed by cloud query engines such as Presto on GCP, to validate data existence and accuracy. Infrastructure engineers from the Tooling team gain insights from the usage data stored and processed in MySQL and create shareable dashboards. Use cases may also involve querying and joining tables from various data sources. A modern SQL system should support querying heterogeneous data sources in a unified interface.
- **Horizontal scaling in the computation layer.** We have witnessed a boost in the number of daily queries sent into Twitter's SQL system in the recent few years. From our operational experience, vertical scaling cannot handle this large number of analytical queries which can cost a considerable amount of resources[1]. A modern SQL system usually prefers the horizontal scaling approach to serve analytical queries [7]. In addition, as an on-premises data center usually has a limited capacity, the horizontal scaling may need to cross data centers or on-premises/cloud environments. As a result, the SQL system needs to handle the challenges brought from horizontal scaling such as cluster orchestration, workload balancing, and fault tolerance.
- **Heterogeneous storage systems in the storage layer.** With the advent of the Big Data era, large-scale storage systems are developed to fulfill the requirements of archiving the scaling volume of data while also maintaining data availability and consistency. The variety of on-premises and cloud data storage systems also poses challenges for a modern SQL system. Maintaining heterogeneous storage systems is a major challenge we have faced in the development and maintenance of Twitter's SQL system. In a modern SQL system, no matter the dataset is stored in which on-premises storage cluster and/or which cloud storage system, query engines should access the dataset through a unified interface without memorizing the concrete physical paths of target datasets.

To overcome these challenges, Twitter engineering teams implement a hybrid-cloud SQL federation system, which processes around 10PB of data daily in production. This paper presents the evolution of the SQL system at Twitter including query federation, cluster federation, and storage federation.

The remainder of this paper is organized as follows. We describe the architectural design and implementation of the hybrid-cloud SQL federation system in Section 2, discuss related work in Section 3, and reflect on lessons learned in Section 4. Section 5 concludes the paper.

[1]From an analysis of a typical Twitter OLAP workload in three months, 19.2% of queries consume more than 1TB peak memory.

## 2. SQL Federation System Design & Implementation

### 2.1. Overview

Figure 1 depicts the architectural design of the hybrid-cloud SQL federation system at Twitter. There are three components: query federation, cluster federation, and storage federation.

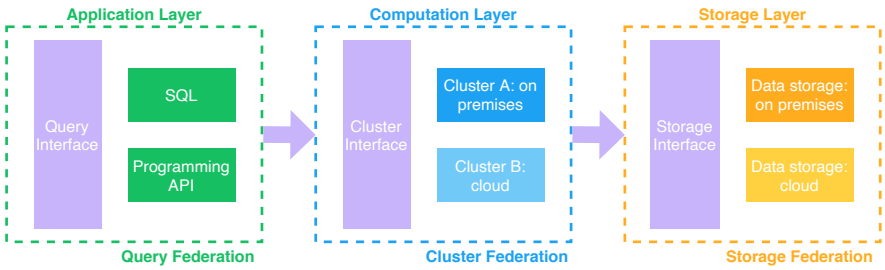



**Figure 1:** Overview of the hybrid-cloud SQL federation system in three layers.

**Query federation**. This exposes a unified query layer to customers such that one interface rules multiple query clusters for heterogeneous data sources. Query federation consists of a SQL component and a programming API component. At Twitter, the SQL component supports basic ANSI SQL semantics as well as some Twitter-specific features implemented into UDFs (user-defined functions). The programming API component enables auxiliary flexible programming features. User requests are eventually converted to SQL and passed to the cluster federation.

**Cluster federation.** This provides a unified cluster layer to the query federation, resolving the challenge of horizontal scaling. It exposes a single entry point, a router service, and hides the cluster details, which reduces the development and maintenance cost. The router service acts as the administrator of SQL engine clusters, helping to schedule queries across the clusters and balancing the workloads among the clusters. Fault tolerance is also improved by forwarding requests only to available clusters when a cluster fails and is offline.

**Storage federation.** This offers a unified view of datasets stored in different archival systems. At Twitter, we are heavily leveraging HDFS as the major on-premises distributed storage platform. In a cloud environment like GCP, we use GCS as the core storage system. The unified layer provides a unique path for each dataset stored in both on-premises and cloud, entirely getting rid of the burden of memorizing accurate physical locations for datasets.

### 2.2. Query Federation

The query federation fulfills three goals. First, it, as a user-facing front-end, converts user inputs to SQL and feeds SQL to the cluster federation. Second, it defines datasets in SQL such that users can locate data from different sources with a uniform approach. Third, it provides UI for interaction and visualization. We leverage Zeppelin [8] to implement the first and third goals, while the second goal is achieved with the help of Presto in the cluster federation. Figure 2 illustrates some SQL examples of query federation in a Zeppelin notebook. Apache Zeppelin is a web-based notebook service that enables interactive data analytics. In the figure, the first

query and the second one are pointing to the on-premise and cloud SQL clusters respectively, identified by a prefix to flag whether the query should be processed in Twitter's data center or public cloud. No extra configuration is required. Although as of the date of publication of this paper, users still have to explicitly mark the target data center, Twitter's data platform engineers are in the planning stage of rolling out a set of features that include automatic recognition with table metadata, data locality, and system performance.

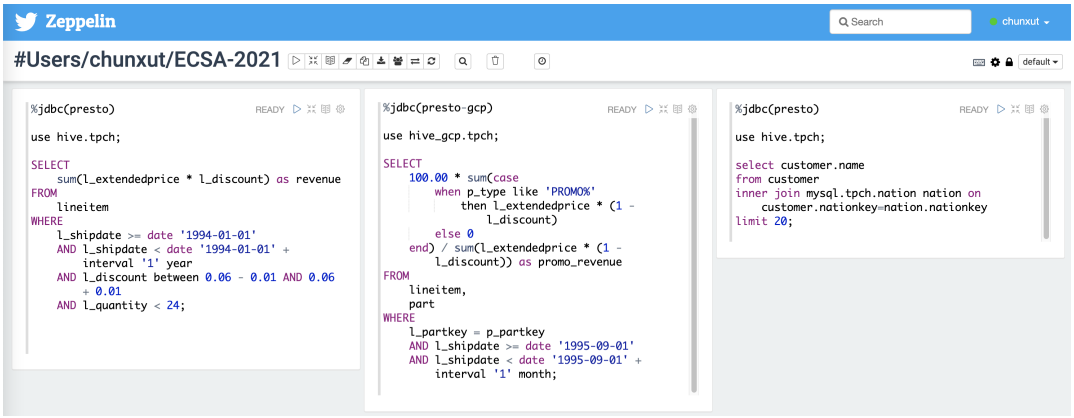



**Figure 2:** Three SQL query federation examples in a Zeppelin notebook. All SQL statements are from the TPC-H benchmark [9].

Besides accessing data within one data source, the third SQL statement in Figure 2 refers to a federated query, joining two tables from HDFS and MySQL. A federated query can refer to joining tables scattered in various data sources. Thus, a query processing engine that can access various data sources should be adopted in the SQL federation system. Presto is adopted for this scenario, which is a distributed SQL query engine targeting "SQL on everything". With a Connector API communicating with external data stores, data is fetched and then converted to the unified internal Presto data types, such that further query processing, such as joining tables, can be accomplished.

## 2.3. Cluster Federation

Figure 3 depicts the architectural design of cluster federation with the following components:

**Router.** The router service is the single entry point and the core of the cluster federation, which exposes a unified interface to the query tools, hides cluster details, and routes requests to concrete clusters. Meanwhile, it helps to balance the workloads among the clusters. Our prior SQL system suffered from imbalanced workloads as the clusters are exposed directly to clients. Some clients may send too many queries to a specific cluster, exhausting the compute resources of that cluster, but leaving other clusters idle. The current hybrid-cloud SQL federation system harnesses multiple routing algorithms including round-robin, random selection, and more complicated load-based approaches with the help of a query cost predictor.

**Query cost predictor.** This is a preditor service to forecast the CPU and memory resource usages of each SQL query. It applies machine learning techniques to learn from historical SQL

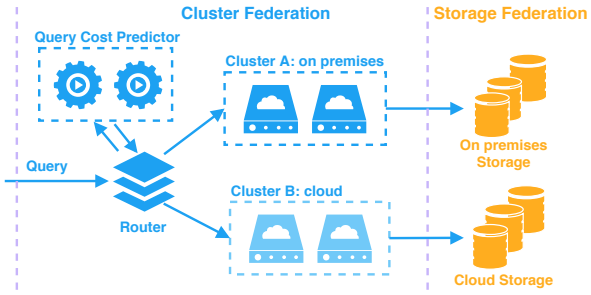



**Figure 3:** Architectural design of the cluster federation.

queries. The predictor details are beyond the scope of this paper and discussed in a separate paper [10].

**SQL engine cluster.** Presto is the query engine utilized in a SQL engine cluster. Each Presto cluster consists of a coordinator node and one or more worker nodes. A SQL engine cluster may be deployed in Twitter's data center or cloud. When it is deployed in Twitter's data center, it queries data stored in on-premises services such as HDFS. By contrast, when it is in the GCP, it queries data stored in the GCS. The SQL engine clusters do not query data across data centers due to performance concerns.

With the cluster federation, users only view logical clusters. When a cluster fails and is offline, the router will remove it from the available cluster list and will not route any requests to this cluster. When the cluster recovers from the failure and is back online, the router will find the cluster through service discovery, mark it as available, and route requests to this cluster. This also improves the availability and fault tolerance, mitigating the maintenance pain we have faced in the prior SQL system with separate clusters.

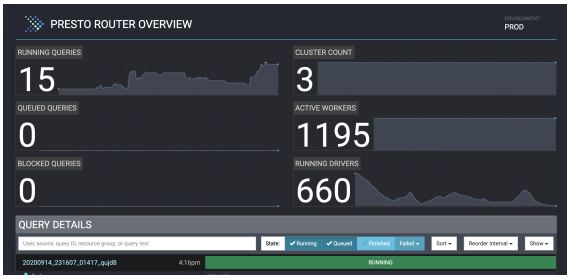



**Figure 4:** Unified UI for cluster federation.

To ease the administration of SQL engine clusters, we build an aggregated UI, shown in Figure 4, on top of the original Presto UI. The UI aggregates the status of all SQL engine clusters, sums the running queries, and monitors the active workers. Moreover, we can dive deeper into one specific cluster to investigate the performance metrics, collected into a unified UI shown in Figure 5. This panel visualizes metrics, including success rates, failures, cluster memory, running queries, etc., collected in the past two weeks.

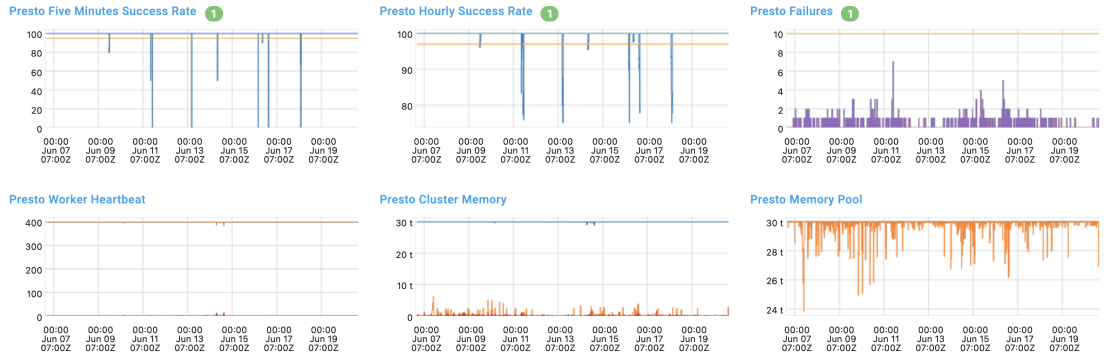



**Figure 5:** Monitoring and alerting of one SQL engine (Presto) cluster.

## 2.4. Storage Federation

To fulfill both scaling data and high availability requirements, Twitter engineers maintain storage clusters in both Twitter's data center and public cloud. Figure 6 depicts the high-level design of the storage federation platform, which is backed by hundreds of thousands of data replication jobs. This platform contains the unified view for data stored in on-premises HDFS clusters and a cloud storage system (GCS in the GCP).

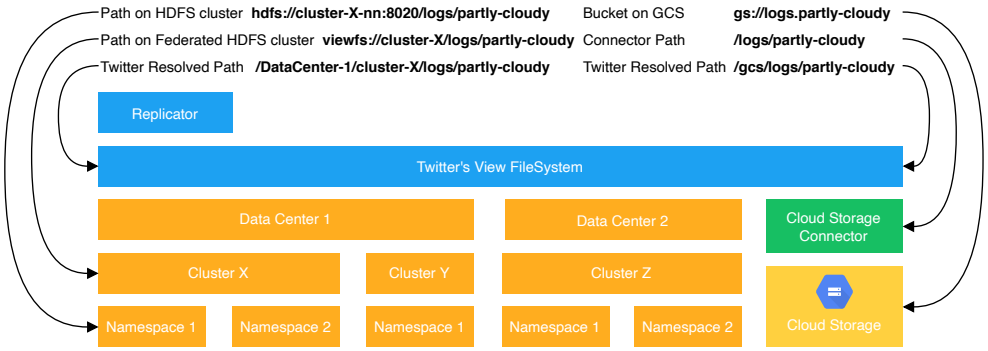


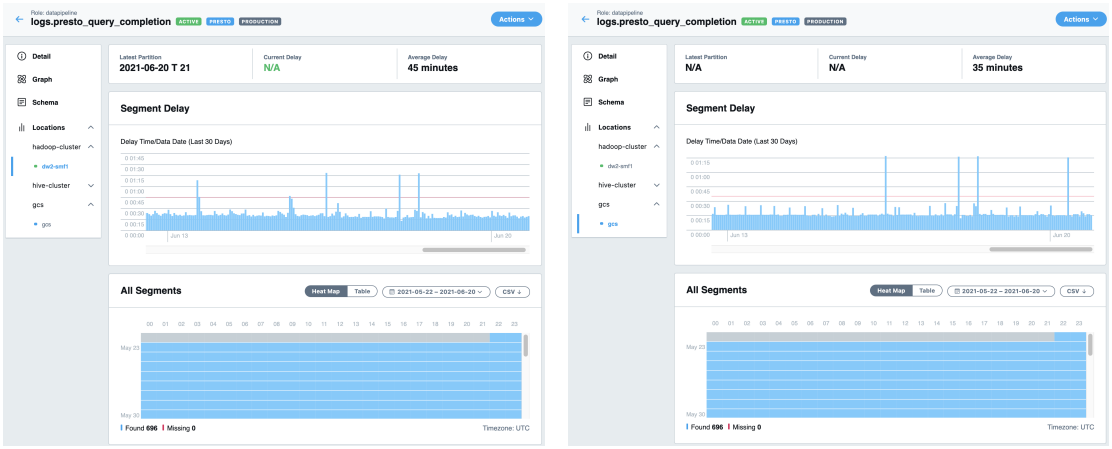

**Figure 6:** Architectural design of the storage federation.

**On-premises HDFS.** Twitter's data platform maintains multiple HDFS clusters across data centers, shown as the left part in Figure 6. Multiple namespaces are also required due to scalability and use case isolation requirements. We scale HDFS by federating these namespaces with user-friendly paths instead of long complicated URIs [11]. As shown in Figure 6, first, the original on-premises data path is *hdfs://cluster-X-nn:8020/logs/partly-cloudy* (*nn* refers to the namenode in HDFS), indicating the data resides in Cluster X in Data Center 1, under the namespace *logs*. Second, we leverage Hadoop ViewFs [12] to provide a single view across namespaces, starting with *viewfs://*. So the original path will become *viewfs://cluster-X/logs/partly-cloudy*. Finally, we extend the ViewFs and implement Twitter's View FileSystem, offering a unified user-friendly path (*/DataCenter-1/cluster-X/logs/partly-cloudy* in Figure 6) and enabling native

HDFS access. A replicator service is also created to help access data stored in different locations.

**Cloud storage (GCS).** Because of the large data volume and use case isolation, we are maintaining thousands of GCS buckets at Twitter. We also leverage the View FileSystem abstraction to hide GCS details behind the storage interface. The cloud storage connector is utilized to interact with GCS via Hadoop APIs. We apply the RegEx-based path resolution to resolve the GCS bucket path, by dynamically creating mountable mapping on-demand in Twitter's View FileSystem. As shown in Figure 6, similar to HDFS, the GCS bucket *gs://logs.partly-cloudy* is finally resolved as */gcs/logs/partly-cloudy*.

As a result, the storage federation only exposes standard unique paths of datasets, no matter they reside in the on-premises HDFS clusters or GCS. In addition, Twitter engineers maintain a metadata service, connected with these storage systems, aiming to provide the standard path of the closest target dataset to query engines. For example, in Figure 6, querying the same *partly-cloudy* dataset, if the query engine is in a Twitter's data center, the on-premises path */DataCenter-1/cluster-X/logs/partly-cloudy* will be returned. By contrast, if the query engine is in the cloud, the cloud path */gcs/logs/partly-cloudy* will be returned.




(a) Details of a dataset in on-premises HDFS.

(b) Details of a dataset in GCS.

**Figure 7:** Unified UI for datasets stored in HDFS and GCS.

To view dataset configuration details, Twitter engineers create a unified UI, shown in Figure 7, with segment support for files stored in various physical locations. Users can thus view different destinations for the same dataset. Specifically, Figure 7a illustrates details of the query log dataset of Presto stored in an on-premises HDFS cluster; Figure 7b points to details of the same dataset stored in the GCS. Figures also show segment delays and segment block information.

## 3. Related Work

With the increasing volume of data, many distributed SQL engines, targeted for analyzing Big Data, emerged in the recent decade. For example, Apache Hive [4] is a data warehouse built on top of Hadoop, providing a SQL-like interface for data querying and a warehousing solution

to address some issues of MapReduce [13]. Spark SQL [5] is a module integrated with Apache Spark, powering relational processing to Spark data structures. Presto [6], originally developed by Facebook, is a distributed SQL engine, targeting "SQL on everything". It can query data from multiple sources which is a major advantage over other SQL engines. Procella [14] is a SQL query engine, employed by YouTube, serving hundreds of billions of queries per day.

With the advent of the public cloud, some cloud-based commercial SQL products emerged in the recent decade. For example, Google BigQuery [15] (a public implementation of Dremel [16, 17]) offers a cloud-based, fully-managed, and serverless data warehouse. Similarly, Snowflake [18] provides a multi-tenant, transactional, and elastic system with full SQL support for both semi-structured and schema-less data. Amazon Redshift [19] applies a classic shared-nothing architecture with Vertica [20]-similar compression techniques, acting as a fully-managed PB-scale data warehouse solution in AWS. Azure Synapse Analytics [21] separates compute and storage for cloud-native execution, bringing together data warehousing and big data workloads.

## 4. Lessons Learned

In this section, we recount some of the qualitative lessons we have learned from the development and maintenance of the SQL federation system at Twitter.

**System monitoring and logging in a hybrid-cloud environment are vital.** Although our hybrid-cloud SQL federation system almost always works well, sometimes when the system goes wrong, it can be a headache to locate the root cause. We also observed architectural differences between on-premises and cloud environments, such as cluster provisioning and security enforcement. An important design decision we have made is implementing a real-time monitoring system with metrics collection and an injectable logging system to trace execution flows. The monitoring system provides a central platform to collect predefined and user-customized metrics, serves observability dashboards/alerts, and helps developers drill down to detailed metrics. Meanwhile, the injectable logging system provides APIs to inject logging points into application source code, collects the logs, and visualizes the execution flows.

**The on-premises capacity planning experience cannot be directly transferred to a hybrid-cloud environment.** During the migration of parts of on-premises workload to the cloud, we discovered that the capacity planning experience cannot be easily reused and shared across data centers, due to varied technical stacks and resource provisioning strategies. For example, one of our early migrated use cases requires around 50 machines in Twitter's data center but needs around 60 to get comparable performance, even though all these machines are sharing similar hardware configuration. This indicates the need for additional prototypes for capacity planning and extra tuning of service in a hybrid-cloud environment.

**SQL is still one of the most widely used languages in data analytics.** As a declarative language, SQL lets users focus on defining the data analytics tasks without worrying about the specifics on how to complete these tasks. Thanks to SQL's high expressiveness in queries and large existing customer bases, some execution engines previously without SQL support, such as Druid [22] and Beam [23], began to support SQL on top of their native query layers. In addition, some SQL variants, such as BigQuery ML [24], even introduced SQL into machine learning use cases. From our observation, SQL is still widely used in data analytics, although

challenged by some competitive alternatives such as Python. Python is more like a powerful supplement for SQL in data analytics with its concise styles and extreme popularity in machine learning, instead of a complete replacement.

## 5. Conclusion

We discussed the evolution of the hybrid-cloud SQL federation system in Twitter's data platform. With various demands for data analytics nowadays, we identified challenges faced within a modern SQL system in the application layer, computation layer, and storage layer. The presented hybrid-cloud SQL federation system overcomes these challenges by implementing query federation, cluster federation, and storage federation. We also discussed some lessons we learned from developing, deploying, and maintaining the system, which we believe can provide some deeper insights for building a large-scale interactive query platform.

## Acknowledgment

Twitter's SQL federation system is a complicated project that has evolved for years. We would like to express our gratitude to everyone who has served on Twitter's Interactive Query team, including former team members Hao Luo, Yaliang Wang, Da Cheng, Fred Dai, and Maosong Fu. We also appreciate Daniel Lipkin and Derek Lyon for their strategic vision, direction, and support to the team. Finally, we thank Erica Hessel, Alex Angarita Rosales, and the anonymous ECSA reviewers for their informative comments, which considerably improved our paper.

## References

[1] P. Agrawal, A new collaboration with Google Cloud, 2018. URL: https://blog.twitter.com/engineering/en_us/topics/infrastructure/2018/a-new-collaboration-with-google-cloud.html.

[2] L. VijayaRenu, Z. Wang, J. Rottinghuis, Scaling event aggregation at Twitter to handle billions of events per minute, in: 2020 IEEE Infrastructure Conference, IEEE, 2020, pp. 1–4.

[3] J. Rottinghuis, Partly Cloudy: The start of a journey into the cloud, 2019. URL: https://blog.twitter.com/engineering/en_us/topics/infrastructure/2019/the-start-of-a-journey-into-the-cloud.html.

[4] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, R. Murthy, Hive: A warehousing solution over a map-reduce framework, Proceedings of the VLDB Endowment 2 (2009) 1626–1629.

[5] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al., Spark SQL: Relational data processing in Spark, in: Proceedings of the 2015 ACM SIGMOD international conference on management of data, 2015, pp. 1383–1394.

[6] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, et al., Presto: SQL on everything, in: 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, 2019, pp. 1802–1813.

[7] J. Tan, T. Ghanem, M. Perron, X. Yu, M. Stonebraker, D. DeWitt, M. Serafini, A. Aboulnaga, T. Kraska, Choosing a cloud DBMS: Architectures and tradeoffs, Proceedings of the VLDB Endowment 12 (2019) 2170–2182.
[8] Apache Zeppelin, 2021. URL: https://zeppelin.apache.org/.
[9] TPC-H benchmark, 2021. URL: http://www.tpc.org/tpch/.
[10] C. Tang, B. Wang, Z. Luo, H. Wu, S. Dasan, M. Fu, Y. Li, M. Ghosh, R. Kabra, N. K. Navadiya, D. Cheng, F. Dai, V. Channapattan, P. Mishra, Forecasting SQL query cost at Twitter (in press), in: 2021 IEEE 9th International Conference on Cloud Engineering (IC2E), IEEE, 2021.
[11] R. Barga, Hadoop filesystem at Twitter, 2015. URL: https://blog.twitter.com/engineering/en_us/a/2015/hadoop-filesystem-at-twitter.
[12] Hadoop ViewFs, 2021. URL: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/ViewFs.html.
[13] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (2008) 107–113.
[14] B. Chattopadhyay, P. Dutta, W. Liu, O. Tinn, A. McCormick, A. Mokashi, P. Harvey, H. Gonzalez, D. Lomax, S. Mittal, R. A. Ebenstein, N. Mikhaylin, H. ching Lee, X. Zhao, G. Xu, L. A. Perez, F. Shahmohammadi, T. Bui, N. McKay, V. Lychagina, B. Elliott, Procella: Unifying serving and analytical data at YouTube, Proceedings of the VLDB Endowment 12 (2019) 2022–2034.
[15] Google BigQuery, 2021. URL: https://cloud.google.com/bigquery.
[16] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, Dremel: Interactive analysis of web-scale datasets, Proceedings of the VLDB Endowment 3 (2010) 330–339.
[17] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, H. Ahmadi, D. Delorey, S. Min, et al., Dremel: A decade of interactive sql analysis at web scale, Proceedings of the VLDB Endowment 13 (2020) 3461–3472.
[18] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, et al., The Snowflake elastic data warehouse, in: Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 215–226.
[19] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, V. Srinivasan, Amazon Redshift and the case for simpler data warehouses, in: Proceedings of the 2015 ACM SIGMOD international conference on management of data, 2015, pp. 1917–1923.
[20] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, C. Bear, The Vertica analytic database: C-store 7 years later, Proceedings of the VLDB Endowment 5 (2012).
[21] J. Aguilar-Saborit, R. Ramakrishnan, K. Srinivasan, K. Bocksrocker, I. Alagiannis, M. Sankara, M. Shafiei, J. Blakeley, G. Dasarathy, S. Dash, et al., POLARIS: the distributed SQL engine in Azure Synapse, Proceedings of the VLDB Endowment 13 (2020) 3204–3216.
[22] Apache Druid SQL, 2021. URL: https://druid.apache.org/docs/latest/querying/sql.html.
[23] Apache Beam SQL, 2021. URL: https://beam.apache.org/documentation/dsls/sql/overview/.
[24] M. Mucchetti, BigQuery ML, in: BigQuery for Data Warehousing, Springer, 2020, pp. 419–468.