Integrated Architecture of SQL Engine and Data Analytics Tool with Apache Arrow Flight and Its Performance Evaluation

Yuichiro Aoki
Research and Development Group, Center for Technology Innovation - Digital Platform
Hitachi, Ltd.
Tokyo, Japan
email: yuichiro.aoki.jk@hitachi.com

Satoru Watanabe
Research and Development Group, Center for Technology Innovation - Digital Platform
Hitachi, Ltd.
Tokyo, Japan
email: satoru.watanabe.aw@hitachi.com

***Abstract*—Data analytics in enterprise systems requires huge amounts of data that are generally stored in databases. Conventionally, Structured Query Language (SQL) engines retrieve the data from the database and data analytics tools, such as Python® scripts, are used to analyze them. In this case, whenever the data moves from the database to the data analytics tools, the data needs to be serialized/deserialized in traditional Open Database Connectivity (ODBC). This is one of the bottlenecks in data analytics performance. In addition, the data needs to be joined for advanced data analytics, and joining the data in the SQL engines takes a lot of time. This is another bottleneck. To remove these bottlenecks, we propose a new architecture integrating the SQL engine and the data analytics tool that reduces the number of data serializations/ deserializations and caches joined results to improve performance of data analytics. Evaluation results show that the data transfer throughput using Apache Arrow/Arrow Flight is 13.1-37.4 times faster than that of a conventional data analytics tool using ODBC. Moreover, this architecture runs 2.4 times faster with the caching mechanism than without it.**

Keywords-relational database; SQL engine; ODBC; Apache Arrow; Apache Arrow Flight; data analytics.

I. INTRODUCTION

Data analytics in enterprise systems requires huge amounts of data that are generally stored in databases. Conventionally, Structured Query Language (SQL) engines, such as Relational DataBases (RDBs), retrieve the data from the database using traditional Open Database Connectivity (ODBC) [6], and data analytics tools like Python® scripts analyze the retrieved data. In such cases, whenever the data moves from the database to the data analytics tool, the data is serialized in the database and deserialized in the data analytics tools. Serialization/deserialization demands many memory copies and takes a lot of time. Thus, serialization/deserialization is a bottleneck in data analytics performance.

In addition, the data needs to be joined for complicated data analytics. Join operations in the SQL engines take a lot of time and are another bottleneck.

In this paper, to address these issues, we propose a new architecture for a data analytics system that integrates the SQL engine and the data analytics tool. It uses Apache Arrow Flight for data transfer between them. Apache Arrow Flight is a brand-new parallel data transfer framework [1]. It uses a column-oriented data format based on Apache Arrow [12]. If the SQL engines, data transfer framework, and the data analytics tools use the same column-oriented data format, the data does not need serialization/deserialization. Thus, the proposed system might be faster than a traditional data analytics system that uses ODBC. In addition, we propose JOIN Result Cache to reduce the number of join operations in the SQL engines. It also improves the performance of the data analytics.

Moreover, we evaluate two types of performance. One is the data transfer performance of the SQL engines using ODBC and Apache Arrow Flight. The other is the performance of JOIN Result Cache.

The rest of the paper is organized as follows. In Section II, we review related work. In Section III, we describe the overview of the proposed architecture of the data analytics system. We show the performance evaluation results in Section IV. In Section V, we make a discussion about the performance, followed by conclusion and future study in Section VI.

II. RELATED WORK

In this section, we review the SQL engines that use Apache Arrow Flight. Dremio [2] is an open-source SQL engine for cloud data lakes. Dremio uses both ODBC and Apache Arrow Flight as a connection with the data analytics tools. Dremio discloses its performance with ODBC and Apache Arrow Flight [3] and Apache Arrow Flight performs 15 times faster than ODBC. In addition, Dremio compares benchmarking results using Transaction Processing Performance Council – Decision Support (TPC-DS) generated data on PrestoDB [4]. On average, Dremio is 3-4 times faster than PrestoDB. However, both results show the overall performance of SQL queries, and the performance of data transfer itself with ODBC and Apache Arrow Flight was not compared.

Li et al. [5] implemented data transfer functionalities using Apache Arrow Flight on the DB-X (currently known as noisepage) database management system [7]. They measured the data transfer throughput using Apache Arrow Flight and Remote Direct Memory Access (RDMA) over

 ISBN: 978-1-61208-891-4

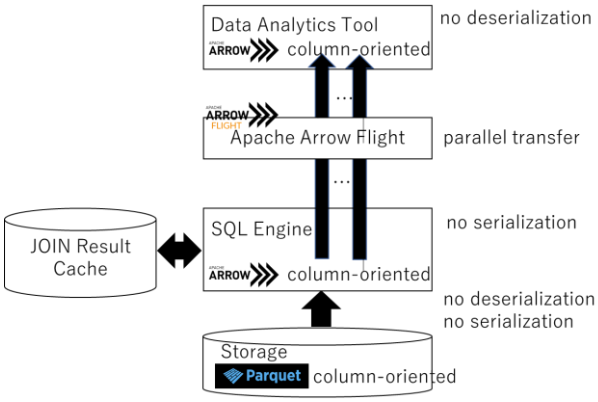

Figure 1. Proposed architecture.

Ethernet. RDMA is slightly faster than Apache Arrow Flight. However, they do not show the overall data analytics system design.

Magpie [8] measured speedups of SQL queries with or without caching data in Apache Arrow Flight. Apache Arrow Flight with caching data runs 2-3 times faster than that without it. However, data transfer time with or without Apache Arrow Flight was not evaluated.

ImmVis [9] is an open-source framework for immersive analytics. It is suggested that data transfer performance may improve if ImmVis uses Apache Arrow Flight. However, the data transfer performance of Apache Arrow Flight was not evaluated.

InfluxData [10] refers to Apache Arrow, Apache Arrow Flight, and Apache Parquet. Data transfer time using Apache Arrow Flight was shown. However, no performance comparison was conducted.

NVIDIA® RAPIDS [11], DataBricks [14], Google BigQuery [16], and Snowflake [17] use Apache Arrow internally. However, they do not use Apache Arrow Flight.

In-database analytics analyzes the data in the RDBs and only the results are transferred to the users [18]. However, in huge results case, the data transfer time is also problematic.

In contrast, we propose a new architecture for a data analytics system that can rapidly transfer the data between the SQL engines and the data analytics tools. In addition, we show the data transfer throughput of both ODBC and Apache Arrow Flight on various SQL engines.

III. PROPOSED ARCHITECTURE OF DATA ANALYTICS SYSTEM

A. Proposed Architecture

Figure 1 illustrates the proposed architecture. It uses a column-oriented Apache Arrow data format internally. The SQL engine and the data analytics tool are connected via Apache Arrow Flight. Apache Arrow/Arrow Flight decreases the number of serializations/deserializations. In addition, the data is stored in a column-oriented Apache Parquet format in storage [15]. This also decreases the number of serializations/deserializations in collaboration with Apache Arrow in the SQL engine. In this architecture, no serialization and deserialization of the data occurs from bottom to top.

Moreover, the proposed architecture has JOIN Result Cache next to the SQL engine. JOIN Result Cache reduces the number of join operations.

TABLE I. PERFORMANCE EVALUATION ENVIRONMENT

CPU	Intel® Core™ i7-8665U (4 cores / 8threads)	
Memory	32GB	
Storage	1TB SSD	
Host OS	Windows 10 Pro 2004	
Virtual Machine (VM)	Oracle VM VirtualBox 6.1.14	
	VM CPU	4 processors
	VM Memory	16GB
Guest OS	CentOS 7.8	

B. Apache Arrow Flight

Apache Arrow Flight is a brand-new data transfer framework, and 1.0.0 was released in 2020 and 5.0.0 in 2021 [1][13]. Apache Arrow Flight uses Apache Arrow column-oriented in-memory data format [12]. If the SQL engines hold the data in Apache Arrow format in memory, Apache Arrow Flight can use the data for transfer without serialization in the SQL engine. In addition, Apache Arrow Flight transfers the data in parallel using gRPC. The gRPC is an open-source high performance RPC framework using Hyper Text Transfer Protocol/2 (HTTP/2). HTTP/2 enables multiple HTTP requests to be sent on a single Transmission Control Protocol (TCP) connection without waiting for the corresponding responses. Thus, Apache Arrow Flight enables faster data transfer between the data analytics tools and the SQL engines than traditional database connectivity, such as ODBC.

C. JOIN Result Cache

This architecture has JOIN Result Cache beside the SQL engine. Some conventional systems cache the join results without precomputing them. However, this architecture pre-computes them on the basis of join query history before join queries are issued, and caches them. For example, a join query is precomputed if it has the same columns as a previous join query but has different tables. Such cases often appear in daily tabulation of Point-Of-Sales (POS) system. Different daily sales tables have the same column names. The tables are inferred from the history of table usage in previous join queries, because in daily tabulation, tables are often mechanically named in day order, such as sales20210803, sales20210804, etc. Thus, join can be precomputed. The SQL engine uses the results if they are cached. As a result, we could improve the data analytics performance and shorten the Turn-Around Time (TAT) of data analytics.

IV. PERFORMANCE EVALUATION

To prove the effectiveness of the new architecture, we evaluate the data transfer time using ODBC or Apache Arrow Flight. In addition, we make an evaluation of JOIN Result Cache.

 ISBN: 978-1-61208-891-4

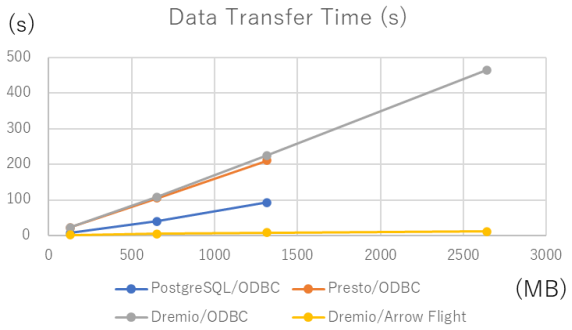

Figure 2. Data transfer time from SQL engine to Python® script.

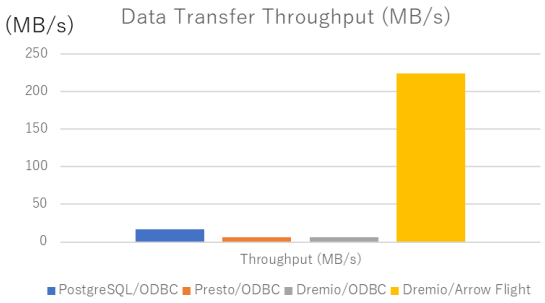

Figure 3. Data transfer throughput from SQL engine to Python® script.

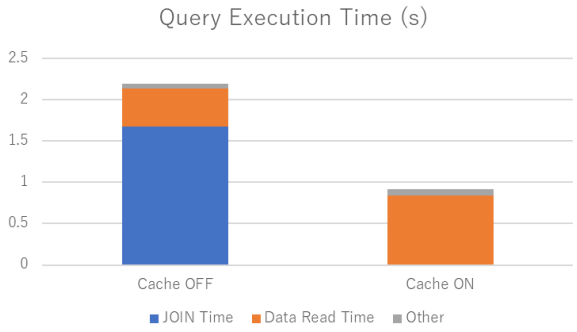

Figure 4. Effect of JOIN Result Cache.

A. *Performance Evaluation Environment*

In this section, we briefly overview the performance evaluation environment. Table I shows details of the environment. We use a virtual machine on Windows 10 for performance evaluation.

B. *Performance Evaluation Targets and Methods*

In this section, we briefly explain performance evaluation targets and methods. We created Python® scripts as an example of the data analytics tool. We imported pyodbc 4.0.30 as connections with the SQL engines in the Python® scripts. It is a Python® wrapper of ODBC. The Python® scripts executed an SQL query in the execute() method and fetched the data from the SQL engines in the fetchall() method. We measured the execution time of this fetchall() method as the data transfer time in ODBC cases.

We also imported pyarrow 2.0.0 as connections with the SQL engines in the Python® scripts. It is a Python® wrapper of Apache Arrow and Apache Arrow Flight. We calculated the data transfer time from the difference between script execution time and SQL engine server's Central Processing Unit (CPU) execution time because Apache Arrow Flight is implemented in C++ libraries of pyarrow and does not use the fetchall() method.

We selected open-source SQL engine Dremio 4.9.1 Community Edition that is enabled to use both Apache Arrow Flight and ODBC [2]. In addition, we also selected PostgreSQL 9.2.24 and Presto 0.250 as ordinary SQL engines that use ODBC.

We generated dummy datasets using Python® Faker package. The datasets have 5 columns (employee ID, first name, last name, age, and education history) and 2.5M, 12.5M, 25M, and 50M rows, respectively. The sizes of the datasets are about 130MB, 650MB, 1315MB, and 2642MB, respectively.

We used SQL queries "SELECT * FROM dataset" for performance measurement. Though they are very simple, we focus on the data transfer time, not the data processing time inside the RDBs, such as GROUP BY operations.

In addition, we measured the effect of the JOIN Result Cache on Dremio. We prepared other thtree datasets with the dataset described above. Two of them are inner-joined and other two are also inner-joined. Lastly, these two inner-joined tables are inner-joined. A Reflection (materialized view in Dremio) of the resulting table was calculated to simulate JOIN Result Cache. The resulting table was queried as "SELECT * FROM table". Its size is about 256 MB.

C. *Performance Evaluation Results*

Figure 2 shows data transfer time from the SQL engine to the Python® script in seconds. Each line means PostgreSQL/ODBC, Presto/ODBC, Dremio/ODBC, and Dremio/Apache Arrow Flight, respectively. We cannot measure the data transfer time of PostgreSQL/ODBC and Presto/ODBC in the 2642MB dataset case because they stopped executing the query with error messages.

In this figure, Presto/ODBC and Dremio/ODBC take almost the same measurement time. PostgreSQL/ODBC runs faster than them. Dremio/Apache Arrow Flight is the fastest in these four measurements.

Figure 3 shows data transfer throughput from the SQL engine to the Python® scripts in MB/s. Each bar means, from left to right, PostgreSQL/ODBC, Presto/ODBC, Dremio/ODBC, and Dremio/Apache Arrow Flight, respectively. The data transfer throughput of Dremio/Apache Arrow Flight is 224MB/s. It is 13.1 times faster than PostgreSQL/ODBC, 36.0 times faster than Presto/ODBC, and 37.4 times faster than Dremio/ODBC.

Figure 4 shows the effect of JOIN Result Cache. In Cache ON case, JOIN time (blue bar) disappeared. As a result, query execution time decreases by 2.4 times.

 ISBN: 978-1-61208-891-4

V. DISCUSSION

ODBC needs data to be serialized and deserialized before and after the data transfer. On the other hand, Apache Arrow Flight uses the Apache Arrow column-oriented in-memory data format that the SQL engine (Dremio) uses internally. Thus, Apache Arrow Flight does not need the data to be serialized before the data transfer. In addition, the Python® script uses the pyarrow module, and the transferred data is retained in Apache Arrow format. Thus, the data do not need to be deserialized. We suppose that this is why Apache Arrow Flight outperforms ODBC.

However, we suppose that, in many cases, the data analytics tools written in Python® use pandas DataFrame, which is not column-oriented. Thus, the data needs to be deserialized when it is analyzed. This could be another overhead of the data transfer. In a simple experiment, for example, the data deserialization of 1315MB Apache Arrow array to pandas DataFrame in Python® takes about 2.6s. This could worsen the data transfer performance by 33.8%. If the time of deserialization linearly depends on the size of the data, 1TB data needs about 40 additional minutes for deserialization. This overhead deteriorates the TAT of data analytics.

After serialization/deserialization disappears, join time is a performance bottleneck. In Figure 4, 76% (blue bar) of the query execution time is join time. The JOIN Result Cache removes join query processing and outperforms Cache OFF case by 2.4 times. The difference between Magpie and the proposed architecture is that the former has cache in Apache Arrow Flight and the latter has cache out of Apache Arrow Flight. It affects the maintenance cost of the system.

The Cache OFF case only reads smaller datasets (296MB) from the storage. However, the Cache ON case reads larger joined table (512MB). That is the cause of the difference in data read time (orange bar) in Figure 4.

In addition, the data analytics system using the proposed architecture can cross the cloud boundaries, because it does not use specific hardware, such as RDMA. This means that data analytics users can distribute the data among usual clouds where the data analytics tools are not installed.

Additionally, if the system resides in one cloud, we can use memory-mapped files in place of file Input/Output (I/O) system calls between storage and the SQL engines. When files are mapped into memory, data in the files is read from and written to the mapped files as if it were in a memory. I/O system calls are usually much slower than memory read/write. Therefore, memory-mapped files can speed up read/write performance of the SQL engines. Thus, in addition to Apache Arrow Flight, memory-mapped files enable us to improve the system performance more and shorten the TAT of data analytics.

VI. CONCLUSION

We proposed a new architecture for a data analytics system using column-oriented Apache Arrow/Arrow Flight. We compared the data transfer throughput performance between the data analytics tool and the SQL engine using ODBC and Apache Arrow Flight. We found that Apache Arrow Flight transfers the data 13.1-37.4 times faster than ODBC because serialization/deserialization of the data is eliminated. In addition, JOIN Result Cache accelerates the query by 2.4 times using precomputed join results. Thus, our proposed architecture can improve the TAT of the data analytics.

In future work, we will design and implement such a data analytics system using Apache Arrow and Apache Arrow Flight. It may reduce the data analytics time and help data analytics users to gain new insights from the data more rapidly.

REFERENCES

[1] The Apache Software Foundation, “Arrow Flight RPC” [Online]. Available from: https://arrow.apache.org/docs/format/Flight.html [Retrieved: August, 2021].

[2] Dremio, “Set Your Data Free” [Online]. Available from: https://www.dremio.com [Retrieved: April, 2021].

[3] P. Shrivastava, “Eliminating Data Exports for Data Science with Apache Arrow Flight” [Online]. Available from: https://www.dremio.com/eliminating-data-exports-for-data-science-with-apache-arrow-flight [Retrieved: April, 2021].

[4] S. Leontiev, “Think Presto Is Fast? Dremio is 3,000 Times Faster.” [Online]. Available from: https://www.dremio.com/dremio-vs-presto [Retrieved: April, 2021].

[5] T. Li, M. Butrovich, A. Ngom, W. S. Lim, W. McKinney, and A. Pavlo, “Mainlining Databases: Supporting Fast TransactionalWorkloads on Universal Column-oriented Data File Formats,” arXiv:2004.14471, 2020.

[6] Microsoft®, “What Is ODBC?” [Online]. Available from: https://docs.microsoft.com/en-us/sql/odbc/reference/what-is-odbc?view=sql-server-ver15 [Retrieved: April, 2021].

[7] Database Research Group at Carnegie Mellon University, “noisepage” [Online]. Available from: https://github.com/cmu-db/noisepage [Retrieved: April, 2021].

[8] A. Jindal, et al., "Magpie: Python at Speed and Scale using Cloud Backends," in Proc. CIDR’21, 2021.

[9] F. A. Pedroso and P. D. P. Costa, "ImmVis: Bridging Data Analytics and Immersive Visualisation," Proc. VISIGRAPP 2021, vol.3, pp.181-187, 2021.

[10] P. Dix, “Apache Arrow, Parquet, Flight and Their Ecosystem are a Game Changer for OLAP” [Online]. Available from: https://www.influxdata.com/blog/apache-arrow-parquet-flight-and-their-ecosystem-are-a-game-changer-for-olap/ [Retrieved: June, 2021].

[11] NVIDIA®, “RAPIDS” [Online]. Available from: https://developer.nvidia.com/rapids [Retrieved:]2020.06.04

[12] The Apache Software Foundation, “Apache Arrow” [Online]. Available from: https://arrow.apache.org/ [Retrieved: April, 2021].

[13] Wes McKinney, “Introducing Apache Arrow Flight: A Framework for Fast Data Transport” [Online]. Available from: https://arrow.apache.org/blog/2019/10/13/introducing-arrow-flight/ [Retrieved: April, 2021].

[14] L. Jin, “Introducing Pandas UDF for PySpark” [Online]. Available from: https://databricks.com/blog/2017/10/30/introducing-vectorized-udfs-for-pyspark.html [Retrieved: June, 2021].

[15] The Apache Software Foundation, “Apache Parquet” [Online]. Available from: https://parquet.apache.org/ [Retrieved: May, 2021].

 ISBN: 978-1-61208-891-4

[16] Google, “Download table data in the Arrow data format” [Online]. Available from: https://cloud.google.com/bigquery/docs/samples/bigquerystorage-arrow-quickstart?hl=en [Retrieved: June, 2021].

[17] H. Kapre, “Fetching Query Results From Snowflake Just Got a Lot Faster With Apache Arrow” [Online]. Available from: https://www.snowflake.com/blog/fetching-query-results-from-snowflake-just-got-a-lot-faster-with-apache-arrow/ [Retrieved: June, 2021].

[18] J. Taylor, “In-database analytics,” [Online]. Available from: http://www.decisionmanagementsolutions.com/wp-content/uploads/2015/06/In-database-Analytics-Decision-Management-Solutions.pdf [Retrieved: August, 2021].

 ISBN: 978-1-61208-891-4