DBMS Final Project: Faster Query Performance on YeSQL

萬庭佑
r10631039@ntu.edu.tw
National Taiwan University
Taipei, Taiwan

劉宗翰
r11921094@ntu.edu.tw
National Taiwan University
Taipei, Taiwan

呂冠輝
r10942129@ntu.edu.tw
National Taiwan University
Taipei, Taiwan

王奕方
r11946012@ntu.edu.tw
National Taiwan University
Taipei, Taiwan

ABSTRACT

YeSQL [5] is an SQL extension that enhances the usability, expressiveness, and performance of Python User Defined Functions (UDFs) within existing database systems. This framework supports Python UDFs fully integrated with relational queries as scalar, aggregator, or table functions with several performance enhancements. In our report, we mainly focus on the performance of processing large CSV files using YeSQL UDF. We identified that the existing functions were not optimized for efficiently handling CSV files in the range of a few gigabytes. As a solution, we proposed a data conversion approach to address this issue. Our proposed way is able to efficiently convert large CSV files to Parquet format using the Apache Arrow library, and enable YeSQL to directly query on Parquet data via SQLITE virtual table. This approach significantly improved query speeds, leveraging the powerful features offered by the Parquet format. Experimental results show that our strategy improve the query speeds up to 94x, and it's also effective if we take account of the Parquet data conversion.

1 INTRODUCTION

The current landscape of data processing is characterized by an abundance of diverse data sources and the increasing demand for handling complex processing tasks on massive volumes of data. In response to these challenges, Python User Defined Functions (UDFs) have emerged as a popular solution. However, existing data processing systems have limitations in terms of the usability, expressiveness, and performance of Python UDFs. This motivates the need for a more efficient and versatile approach.

In this context, we introduce YeSQL, an SQL extension designed to address the limitations of Python UDFs and provide a more usable, expressive, and preferment solution. YeSQL offers significant enhancements over traditional UDF implementations by leveraging the power of Python for data processing tasks. By integrating YeSQL with existing database systems, users can harness the flexibility and expressiveness of Python while benefiting from improved performance and usability.

To demonstrate the capabilities of YeSQL, we have conducted a comprehensive implementation and evaluation process. Our implementation focuses on addressing the challenges associated with handling large CSV files in database management systems. We observed that existing CSV-to-Parquet conversion libraries often suffer from performance issues and high memory footprint when dealing with large files. To overcome these limitations, we employed the Apache Arrow library, leveraging its streaming reader function to incrementally read record batches from CSV files and efficiently manage memory resources.

The results of our experiments are promising. We successfully converted large CSV files, including a 14.6GB file with billions of rows, to Parquet format using our proposed approach. The conversion times were significantly reduced, resulting in efficient processing.Furthermore, we integrated the sqlite-parquet-extension into YeSQL, allowing direct querying of Parquet data within the database engine. This integration demonstrated substantial improvements in query speed, thanks to the powerful column statistics features provided by Parquet. Our experiments revealed that queries executed on Parquet data completed in significantly less time compared to traditional file UDFs, showcasing the advantages of YeSQL in terms of query optimization.

In conclusion, our introduction of YeSQL as an SQL extension aims to overcome the limitations of Python UDFs in existing data processing systems. The combination of YeSQL's enhanced usability, expressiveness, and performance, along with the efficient conversion of large CSV files to Parquet format, demonstrates the potential of our approach to significantly improve data processing workflows. By leveraging the strengths of Python, Apache Arrow, and Parquet, we provide a comprehensive solution that addresses the challenges of handling diverse data sources and complex processing tasks in the current data processing landscape.

2 YESQL OVERVIEW

YeSQL is designed to work in synergy with existing systems, such as the SQLite DBMS. One of the key advantages of YeSQL is its ease of implementation on top of the SQLITE API, which provides native support for C UDFs. This native support allows for the seamless integration of YeSQL with SQLite, enabling the efficient execution of UDFs transformed from Python CFFI Wrapper.

Python UDFs are widely utilized in various application scenarios, catering to both resource-intensive database servers and resource-constrained edge devices. For data analysts and data scientists, running complex UDFs on server-based DBMSs is often preferred. However, in edge computing applications, a function-shipping paradigm that brings computation closer to the data collector can offer significant benefits. For example, deploying a UDF on an embedded system, co-located with the data collection process, such as a small compute device like a Raspberry Pi monitoring vital metrics in a

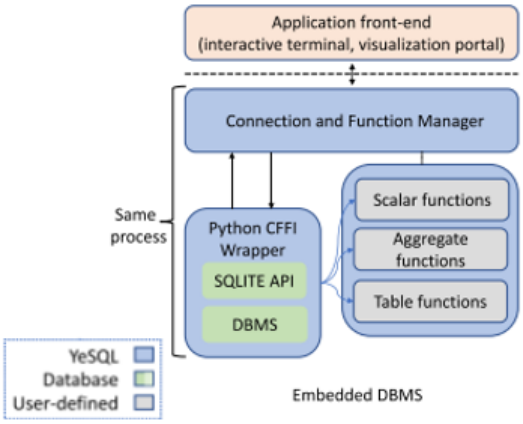


Figure 1: YeSQL architecture in embedded DBMS

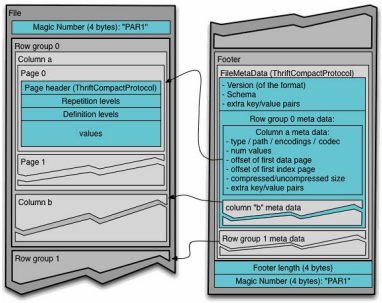


Figure 2: Parquet format (from Apache Parquet)

wind turbine generator, enables efficient data reduction computations. In such scenarios, the use of a UDF running on an embedded system, within the same process, becomes an advantageous design choice.

YeSQL inherits the conventional classification of UDFs into scalar, aggregate, and table functions. The Python CFFI wrapper serves as the bridge between Python and the database engine, facilitating seamless data exchange. With an embedded DBMS like SQLite, the Python CFFI wrapper allows UDFs to be submitted as callback functions, ensuring smooth integration between the database engine and the Python UDF. Moreover, the SQLITE API offers native support for extended-SQL functionality through C UDFs, further enhancing the capabilities and compatibility of YeSQL.

By leveraging the native support for C UDFs in the SQLITE API and the seamless integration with Python through the Python CFFI wrapper, YeSQL provides a powerful and versatile platform for executing UDFs within existing database systems. This enables users to harness the benefits of Python UDFs while leveraging the efficiency and reliability of established DBMSs like SQLite. Whether in resource-intensive server-based environments or resource-constrained edge computing scenarios, YeSQL offers a flexible and efficient solution for executing UDFs and unlocking the full potential of data analysis and processing.

3 PARQUET VIRTUAL TABLE

In this section, we introduce the Parquet file format and the benefits of using this format when executing queries, and talk about the virtual table mechanism that using in YeSQL system for accessing the foreign data contents.

3.1 Parquet format

The Parquet format is one of the most widely used columnar storage formats in the big data systems and even for "smaller" data processing libraries like Python's Pandas. The data organization shown at Figure 2 consists of row-groups, column chunks and pages. Given a structured data with thousands rows and ten columns, we can horizontally partition the rows in this data into several row-groups (default 128MB). Within those row-groups, the columns in this data are vertically partitioned to column chunks. The actual data is stored in the column chunks, called (data) pages. The pages contain the metadata such as minimum value, maximum value and the number of the value in that page. For semi-structured data (JSON, XML), there are repetition levels and definition levels stored in the pages to reconstruct nested schema. In Parquet, a wide range of encoding algorithms are available for various value types, including bit-packing, run-length encoding, delta encoding, and delta strings, which reduce the storage footprint.

The metadata in the pages is really helpful. It provides the column statistics (min, max, count) which helps in avoiding the reading of irrelevant data. Another advantage is the schema in Parquet is defined in the footer, reducing the overhead of inferring the schema from the files when creating tables in SQL queries. These powerful features make converting files, especially CSV files, to Parquet beneficial for query performance. For example, if we have a query `SELECT * FROM table WHERE x = 123`, as `table` stored in a Parquet file, the query reads the footer first and retrieves the metadata information for each row-group, such as `"Row-group 0: x: [min:10, max:90]"`, `"Row-group 1: x: [min:100, max:200]"` and `"Row-group 2: x: [min:120, max:150]"`. This metadata can be leveraged to skip unnecessary row-groups (e.g., skipping Row-group 0 in this example), thereby avoiding unnecessary table scans.

3.2 Virtual table mechanism

To read the data that is not in the current DBMS, SQLITE provides a virtual table interface that allow access to foreign data. The YeSQL framework also utilizes this interface to achieve polymorphic UDFs. For example, a polymorphic UDF for selecting data from a file: `SELECT * FROM file("data.csv");`
with the SQLITE API would be:

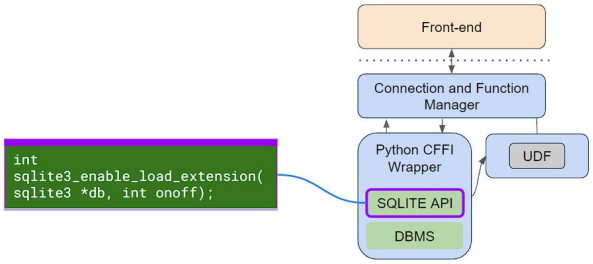
```
create virtual table if not exists
temp.vt_name using file("data.csv","automatic_vtable:1");
select * from temp.vt_name;
drop table if exists temp.vt_name;
```

We supposed to create a UDF for reading the parquet file, which is not supported by YeSQL:
`SELECT * FROM parquet("data.parquet");` To create a virtual table implementation, we need to write all necessary methods in

`sqlite3_module` such as `xCreate` for creating a new instance of a virtual table, `xOpen` for creating a new cursor for accessing (read and/or writing) a virtual table created by `xCreate`, `xNext`, `xColumn` and `xEof` for moving the cursor and accessing the values of the virtual table. These methods can be implemented as Python functions and submitted as callback functions, like:

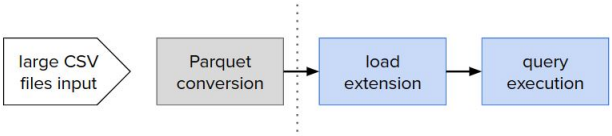
```
ffi.callback(
"int(sqlite3_vtab *pVTab, sqlite3_vtab_cursor **ppCursor)",
xOpen)
```

Where the first argument represents the `xOpen` C API in SQLITE, and the second one is the Python UDF.



4 IMPLEMENTATION

Our implementations are displayed at the following diagram



The first thing we need to address is the "large" CSV files conversion. While there are existing libraries available for converting CSV to Parquet ([1], [3]), we observed that it's time consuming when dealing with CSV files that are several gigabytes in size. This is likely due to the need to load the entire input file into memory, which can lead to device crashes or performance issues.

To avoid such high memory footprint, we employ the function `pyarrow.csv.open_csv` from Apache Arrow, which returns a streaming reader that incrementally reads record batches from a CSV file. By reading data in batches, we can effectively manage memory usage. Once we have obtained the batches of data, we create a Pyarrow table using `pyarrow.Table.from_batches` and write this table to a parquet file. This approach allows us to handle large CSV files efficiently without overwhelming memory resources.

Once the conversion is complete, we have to make YeSQL available for reading Parquet file through the SQLITE virtual table. Instead of directly crafting a function, we utilize the existing extension sqlite-parquet-vtable [2] to achieve parquet virtual table. This extension provides support for row-groups filtering based on strings and numerics, as long as the SQLite type matches the corresponding Parquet type. It turns out that we can skip the irrelevant row-groups as described in Section 3. To enable YeSQL to load extension, we wrap the SQLITE C API using CFFI like:

```
ffi.cdef("""int  sqlite3_enable_load_extension(sqlite3
*db, int onoff);""")
```

This allows YeSQL to call the `select load_extension("lib")` SQL function to load the sqlite-parquet-vtable extension. The overall system modification can be represented as the figure below

5 RESULTS

We measured the parquet-YeSQL performance using the dataset from *Statistics Canada: Census Profile, 2016 Census* [4]. We choose one extremely large CSV file (14.6GB/billions rows) called `census.csv` and one relatively small CSV file (1.2GB/ten millions rows) called `census_a.csv` to convert to Parquet, and the conversion time require 208.8 (sec) and 18.2 (sec) respectively.

We then investigated the query performance using `census` dataset. Given a simple query that averages a certain row `C14` (equivalent to `C15` in the CSV file), with a condition on `"Canada"`, the query result is displayed in Figure 3

```
r10942129@bl524-server1:~/YeSQL$ python3 csv2parquet.py
execution time is:208.8
r10942129@bl524-server1:~/YeSQL$ rm demo.db
r10942129@bl524-server1:~/YeSQL$ pypy2.7-v7.3.6-linux64/bin/pypy YeSQLite/m
term.py -f udfs -d demo.db
mTerm - version 1.0
running on Python: 2.7.18, APSW: MSPW ver 0.01, SQLite: 3.41.2, YeSQL: 1.9
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
mterm> select load_extension('/home/r10942129/Downloads/libparquet');\
null
--- [0|Column names ---
[1|load_extension('/home/r10942129/Downloads/libparquet')
Query executed and displayed 1 row in 0 min. 0 sec 23 msec.
mterm> create virtual table vt using parquet('./data/census.parquet')\
;
Query executed in 0 min. 0 sec 234 msec.
mterm> select avg(C14) from vt where C3='Canada';
814712.110147
--- [0|Column names ---
[1|avg(C14)
Query executed and displayed 1 row in 0 min. 4 sec 960 msec.
mterm>
```

Figure 3: `census.parquet` Screenshot.

Remarkably, the `select` query completed in less than 5 seconds, showcasing the remarkable efficiency of Parquet. The query only scanned the relevant row-groups where the column metadata matched the specified "Canada" condition, further highlighting the advantageous features of Parquet.

For CSV file in `census` dataset, we use the `file` UDF in YeSQL, and run the same `select` query to see the execution time
It took about "9 minutes" to complete the query as shown in Figure 4. Notice that even we added the whole processes that included in Figure 3, the execution time was less than 4 minutes (conversion + query time). This demonstrates a significant time saving of over 50% compared to using CSV. Although it could be optimized by some query techniques, such as indexing, to accelerate the query speed, the demonstration here indicates that ease of use in Parquet and still get excellent performance without the need for additional query optimization tips.

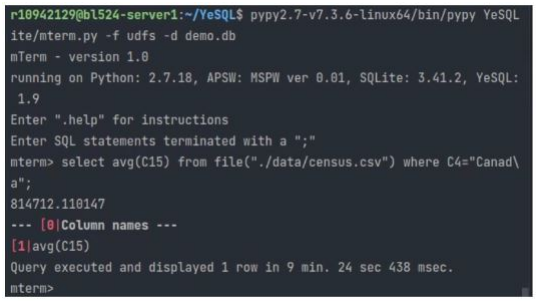


Figure 4: `census.csv` Screenshot.

When using a relatively small dataset `census_a`, the Parquet virtual table also outperforms the `file` UDF in YeSQL which reading the CSV files.

Again, we use the `select` query that average one specific column with condition on `"Canada"`. The query result using Parquet virtual table is displayed in Figure 5

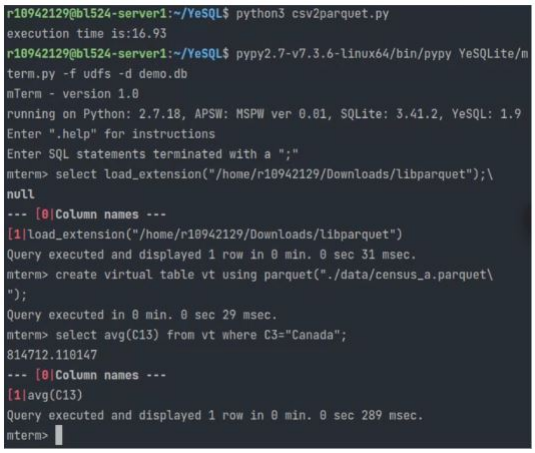


Figure 5: `census_a.parquet` Screenshot.

And the query result using `file` UDF is in Figure 6

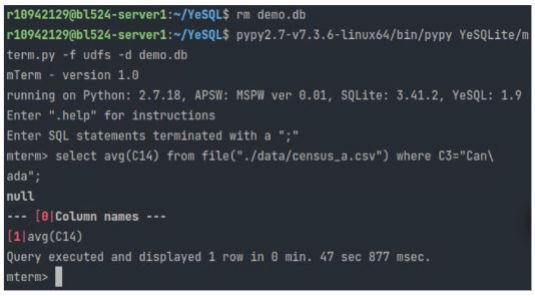


Figure 6: `census_a.csv` Screenshot.

The `select` query completed in 289 (msec) using Parquet, which is significantly fast compared to the `file` UDF (47 seconds). Even we count the whole processing time that dealing with Parquet (less than 18 seconds), the performance is still good.

The experiments conducted in [5] presented practical use cases for measuring the performance of an end-to-end pipeline using three pipelines: *zillow*, *flights*, and *text-mining*. For our analysis, we selected the *flights* pipeline. This dataset consists of a large number of columns (110), two small tables used in joins, and more operators, 23 operators on the larger table, 3 joins, and 1 filter. We didn't expect the significant performance improvements through Parquet conversion, since the CSV files here are quite small (at most 1.7MB), and the purpose of the query statement is mainly involved merging three tables without intensive analytical work, and this could mitigate the useful column statistics feature in Parquet. The execution time of the original query is in Figure 7

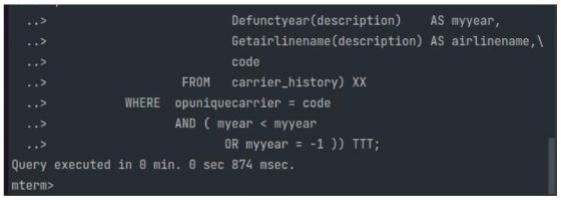


Figure 7: `flights.csv` Screenshot.

The results of our Parquet conversion method is in Figure 8

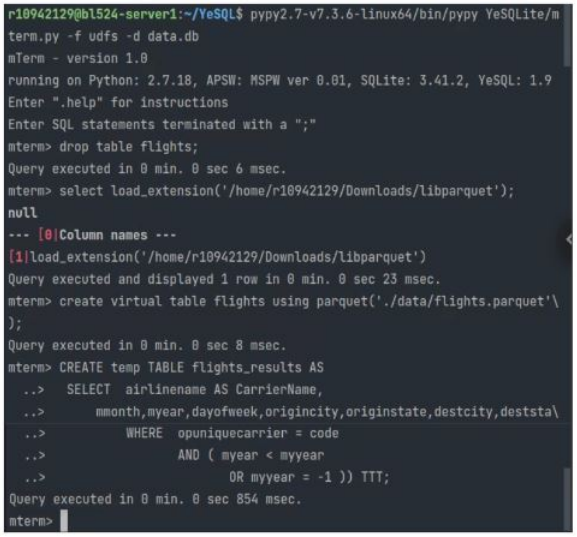


Figure 8: `flights.parquet` Screenshot.

The query execution shown in Figure 7 involves the creation of an additional table to define the schema, followed by the use of the `file(flights.csv)` UDF to insert the tuples into the table. For our execution in Figure 8, we create the virtual table interface without creating an additional table as before. The results in `flights.parquet` only show a slight improvement compared to the `flights.csv`. It seems that the performance cannot be significantly enhanced in this work, as we mentioned before. However,

the previous experiments still demonstrate the effectiveness of our proposed method.

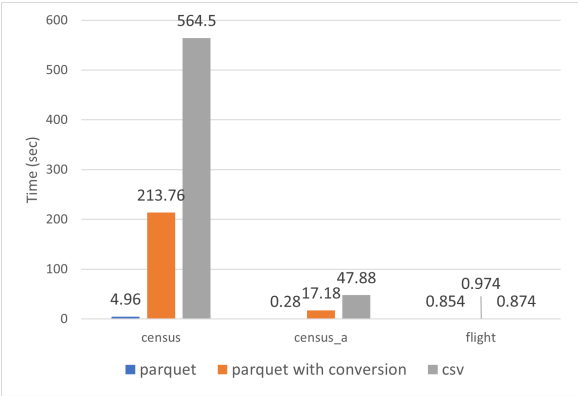


Figure 9: Parquet virtual table performance

Figure 9 shows the experiments conducted on the three data set on the remarkable efficiency of Parquet. The select query on `consus` results in a time saving of over 50% with Parquet. This highlight the advantageous features of Parquet, as it only scanned relevant row-groups based on the specified conditions, without the need for additional query optimization techniques. Similarly, in the `census_a` data set, the select query using Parquet virtual table significantly outperformed the file UDF. This showcases the superior performance of Parquet and its suitability for efficient data analysis workflows.

Although the `flight` data set consisted of relatively small CSV files, the conversion to Parquet still demonstrated slight performance improvements compared to CSV. This can be attributed to the limited analytical work involved in the query statement. Nevertheless, these findings validate the effectiveness of the proposed method and highlight the overall capabilities of YeSQL in handling diverse data sets.

6 CONCLUSION

In this report, we successfully import the sqlite-parquet-extension to YeSQL and make it available for reading Parquet. The results show that such virtual table interface enable us to read Parquet data, and significantly improve the query speed due to the powerful column statistics features. The file conversion using Apache Arrow is also efficient. It turns out that our proposed method can indeed improve the query performance when dealing with large CSV files within YeSQL.

7 WORK DISTRIBUTION

Work distribution

- Discussion: 王奕方、劉宗翰、萬庭佑、呂冠輝
- Searching for Solutions: 王奕方、呂冠輝
- Presentation & Report: 萬庭佑、呂冠輝

REFERENCES

[1] 2018. csv2parquet. https://github.com/cldellow/csv2parquet/
[2] 2018. sqlite-parquet-vtable. https://github.com/cldellow/sqlite-parquet-vtable
[3] 2021. csv2parquet. https://github.com/domoritz/csv2parquet
[4] Statistics Canada. 2016. Census Profile. https://www12.statcan.gc.ca/census-recensement/2016/dp-pd/prof/details/download-telecharger/comp/page_dl-tc.cfm?Lang=E
[5] Yannis Foufoulas, Alkis Simitsis, Lefteris Stamatogiannakis, and Yannis Ioannidis. 2022. YeSQL: "You Extend SQL" with Rich and Highly Performant User-Defined Functions in Relational Databases. *Proc. VLDB Endow.* 15, 10 (jun 2022), 2270–2283. https://doi.org/10.14778/3547305.3547328