Big Data - Auctus

Shvejan Shashank,[1] **Sai Krishna Prathapaneni** [2]

[1,2] Department of ECE, New York University
[1] ssm10076@nyu.edu, [2] sp7238@nyu.edu

Abstract

This project aims to enhance the capabilities of dataset search engines by enabling users to query the contents of datasets they discover. While current search engines allow users to issue queries over metadata associated with datasets, this may not provide sufficient information for determining the suitability of a given dataset. To address this, the project proposes to develop an infrastructure that supports queries over datasets stored in a file system or scalable object storage backends using S3's API. Specifically, the project involves modifying Auctus' (Group Accessed: 2023) codebase to convert datasets to Parquet files, storing them in an open-source object store, and modifying the user interface to allow users to execute SQL queries over the Parquet files stored in the S3 object store using the DuckDB Python wrapper. With the current system we achieve more than 300 times execution speeds to process the datasets.

Introduction

Dataset search engines are specialized search engines that help researchers discover and access relevant datasets for their research. Dataset search engines typically use web crawlers and other data-mining techniques to identify datasets that are publicly available on the internet. Dataset search engines typically allow users to search for datasets using keywords and filters, such as data type, geographic location, and time period.

In recent years, the importance of datasets in scientific research has increased, leading to the development of several popular dataset search engines such as Google Dataset Search (Noy, Burgess, and Brickley 2019) , Kaggle Datasets, government dataset search engines like nyc.gov data engines, Zenodo and Auctus (Group Accessed: 2023). The needs for robust, scalable search engines have never been more needed. one such attempt is give through Meta search engine (Mahmud, Rabbi, and Guy-Fernand 2016) where they have used NLP to extract context from the web. Auctus on other hand provides user to execute programs with efficiency and and robust set of features for the user.

The current version of Auctus performs profiling on the dataset by converting it into CSV files and performs queries on the data using pandas data frames. While this method is reliable and gets the job done in most cases, converting the data into CSV files and then into pandas data frames is a very time-consuming and computationally intensive task. This method is also not scalable to handle heavy requests/-data sets. Also, sometimes just looking at the metadata that Auctus provides might not be enough for the user to decide the suitability of the dataset for their requirements. In this project, we will develop a new feature to Auctus to allow the users to execute SQL queries right from the Auctus platform allowing them to dig deeper into the datasets and validate their suitability. In the following table 1 we present the results with vaex (Breddels et al. 2019), Polars (Ma et al. 2020) and dask (Rocklin 2015) data reader/processors systems. The experiment uses 3 csv files to read and concat them together to generate the common data table across different systems. The speed of excecution of dask comes from the distributed parallel computing setup.

Data reader	speed of execution
Vaex	297 ms
Polars	46.9 ms
dask	15.6 ms
pandas	297 ms
DuckDB	367 μs

Table 1: table showing speed of CPU executions across different systems. System uses i5 processor with 8gb of RAM

Methodology

In this project, we will be developing the infrastructure to allow the users to run SQL queries on datasets right from the Auctus website. Parquet file (Dem and Nadeau 2013) is designed to run heavy data computations much faster and more efficiently than the existing method of using Pandas Dataframes. We will also be using cloud-hosted S3 Buckets on MinIO (Periasamy 2016) to store these parquet files. This combination will form a robust and scalable architecture that can handle heavy data processing and query in a much more efficient way.

Another important element of this architecture is DuckDB (Raasveldt, Mühleisen, and Nes 2019). Duck DB brings the powerful features of SQLite into big data analytics. With no external dependencies and with the ability to be completely

```
%%time
pandas_df = pd.concat([pd.read_csv(f) for f in glob.glob('csv_files/*.csv')])

CPU times: user 223 ms, sys: 9.8 ms, total: 233 ms
Wall time: 230 ms
```

Figure 1: CSV+Pandas Dataframes time taken: 230 ms

```
%%time
con.sql(" from db.parquet")

CPU times: user 367 µs, sys: 280 µs, total: 647 µs
Wall time: 654 µs
```

Figure 2: Duck DB + Parquet time taken:367us

embedded in a single database file. It provides a lightweight and very efficient way to query and process the data. Duck DB can also be configured to with S3 buckets seamlessly allowing it to retrieve the Parquet files stored in MinIO. This architecture provides the best in class performance to perform OLAP operations right from the Auctus website.

Methodologies and architecture:

The new proposed feature in Auctus will include a new "Run SQL" button in the button group of Auctus. Running the backend queries with the data set.

Data fetching

The first time the user clicks on the "Run SQL" button for a specific dataset, the data is fetched from the data source and stored in the S3 bucket hosted in minio using a BytesIO buffer. The dataset initially in the CSV format will be parsed to the parquet format before storing it on the s3 bucket. This parquet file remains saved in the s3 bucket to allow the user to run SQL queries directly on them.

DuckDB

DuckDB performs better on SQL queries especially in the parquet format because it has optimizers that can automatically identify and extract the necessary columns and filters from the query. This means that there is no need for manual effort to identify and specify these details. These extracted details are then automatically utilized in the Parquet reader, which reads and processes data stored in the columnar format. This results in faster and more efficient processing of the SQL queries, leading to improved performance of the Auctus database search engine. Our implementation of DuckDB uses the Pyarrow engine to perform DuckDB queries on Parquet files

MinIO

Minio is an open-source object storage server that is compatible with Amazon S3. It provides a simple, scalable, and cost-effective solution for storing and accessing large amounts of data. In the context of the Auctus project, Minio stores the datasets downloaded from external sources and provides a backend for executing SQL queries on these

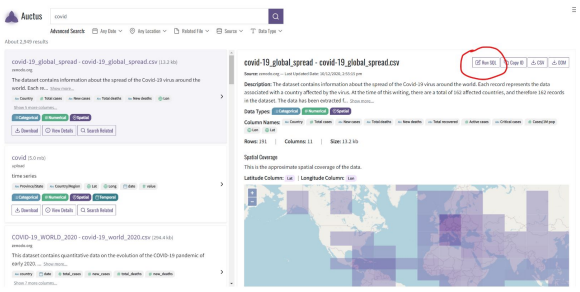

Figure 3: UI changes

Country VARCHAR	Total cases VARCHAR	New cases VARCHAR	Total deaths VARCHAR	New deaths VARCHAR
China	81054.0	46.0	3261.0	6.0
Italy	59138.0	5560.0	5476.0	651.0

Figure 4: Execution results before profiling

datasets. One of the most important reasons for considering MinIO is that Minio is designed to be highly scalable, both in terms of capacity and performance. It can handle large amounts of data and can scale up or down as needed to meet changing demands.

Data profiling

One of the biggest challenges with this Project was to profile the datasets Since the majority of the data sources provide the data in CSV format and these CSV files do not have information about the data types for the columns. For example, let us consider the following case below with two examples of datasets that are parsed without data profiling, as we can see from the column types, all the columns are "Varchar", This could be a problem while writing SQL queries because we can't write queries like

```
1   select * from DATASET where Total cases
        < 100
```

because all the columns are Varchar. Similarly, in figure 6, we can't perform date-related queries on the "date" column as it is of type Varchar.

To handle this, we used the DataMart Profiler to profile the dataset and then automatically convert each column into the correct data type. Now after applying the data profiler, we can observe the results in the figures 5 7 below where each column converted to their most relevant datatype.

Even after applying the Data profiler, the data profiler can sometimes can miss our identifying some data types, This could very frequently happen when dealing with date/time columns, as the date time columns can be represented in many ways, it gets difficult for the data profiler to automatically identify and parse these formats. To handle this case, we have developed a feature where the user can manually change the data types of the columns based on their needs.

Country VARCHAR	Total cases BIGINT	New cases BIGINT	Total deaths BIGINT	New deaths BIGINT
China	81054	46	3261	6
Italy	59138	5560	5476	651

Figure 5: Execution results before profiling with timestamp

country VARCHAR	date VARCHAR	total_cases VARCHAR	new_cases VARCHAR	total_deaths VARCHAR
Afghanistan	2020-03-24	74.0	34.0	1.0
Albania	2020-03-24	123.0	19.0	5.0

Figure 6: Execution results after profiling without timestamp

For example, let us look at the following case where the data mart profiler failed to parse the date-time column which can be observed in Figure 8, in the figure below, the date column is profiled as varchar as the back-end was not able to parse that specific data format.

With our new feature, the user can simply click on the edit button, to get a pre-constructed sql query to change the column type by manually specifying the format, the SQL query could look something like the following query which makes the change to the date column as shown in 9

```
update DATASET set date = (select CAST(
    STRPTIME(date, '%m/%d/%Y %I:%M:%S %p
    ')as TIMESTAMP) as date from DATASET)
    ;alter table DATASET alter date type
    timestamp;
```

In addition, by allowing the user to change data types, the tool becomes more extensible and robust. Users can easily adapt the tool to work with various datasets, regardless of the original data types or formats. This flexibility allows users to easily import and process large and complex datasets, providing a more comprehensive view of the data.

Furthermore, this feature enables users to perform more in-depth data analysis and visualization, as they can now work with different data types to identify trends and patterns that may have been previously overlooked. This not only improves the accuracy and reliability of the data analysis, but also enhances the overall value and usefulness of the tool for the end user.

Results and Future scope

The proposed system successfully enhances the capabilities of the Auctus dataset search engine by enabling users to query the contents of the datasets they discover. By allowing users to execute SQL queries over the Parquet files stored in the S3 object store using the DuckDB Python wrapper, the system provides a much more efficient and reliable method

country VARCHAR	date TIMESTAMP	total_cases BIGINT	new_cases BIGINT	total_deaths BIGINT
Afghanistan	2020-03-24T00:00:00.000	74	34	1
Albania	2020-03-24T00:00:00.000	123	19	5

Figure 7: Execution results after profiling with timestamp

date VARCHAR	state VARCHAR	positive BIGINT
02/12/2021 12:00:00 AM	VA	544209
07/13/2020 12:00:00 AM	VA	71642
02/13/2020 12:00:00 AM	VA	0

Figure 8: Date column is processed as Varchar

for data processing and query handling.

In comparison to the traditional method of querying datasets in CSV format and performing profiling on the data using pandas data frames, the proposed system is significantly faster and more scalable. As shown in Figure 1, the CSV+Pandas Dataframes method took 230 ms to complete the same task that was accomplished in just 367 us using the Duck DB + Parquet method.

Furthermore, the proposed system is more efficient at handling heavy requests/data sets, and it allows users to dig deeper into the datasets and validate their suitability by executing SQL queries directly on them. Experiments could be extended to using other querying engines like SQL servers, pyspark engines and other sql engines to compare the performances over parquet files.

References

Breddels, M. A.; Valtchev, V.; Goulooze, S. E.; ter Horst, H.; and Schoenmakers, R. 2019. Vaex: Out-of-Core DataFrames for Python, Visualization and Machine Learning. *Journal of Open Source Software*, 4(40): 1436.

Dem, J. L.; and Nadeau, J. 2013. The Design of Apache Parquet: A Columnar Storage Format. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*, 1217–1228.

Group, V. Accessed: 2023. Auctus. https://auctus.vida-nyu.org/.

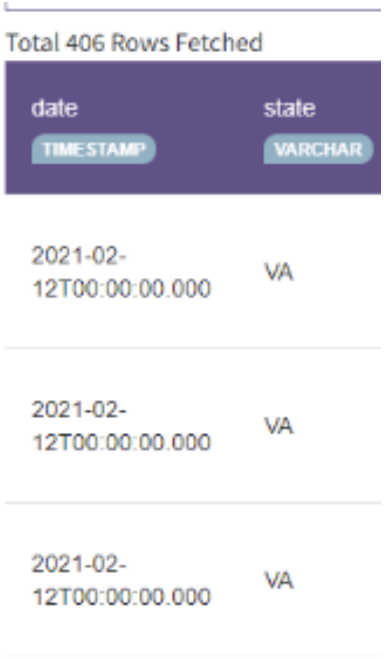

Figure 9: Date column changed after profiling to Timestamp

Ma, I.; Koloski, T.; Huynh, T.; and Verbeek, W. 2020. Polars: Data frames for Rust. *Journal of Open Source Software*, 5(52): 2511.

Mahmud, S. H.; Rabbi, M. F.; and Guy-Fernand, K. N. 2016. An Agent-based Meta-Search Engine Architecture for Open Government Datasets Search. *Communications*, 4: 21–25.

Noy, N.; Burgess, M.; and Brickley, D. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *28th Web Conference (WebConf 2019)*.

Periasamy, A. B. 2016. MinIO: Cloud Native Object Storage. In *Proceedings of the 2016 International Conference on Cloud Computing and Big Data Analysis*, 29–33.

Raasveldt, M.; Mühleisen, H.; and Nes, N. 2019. DuckDB: an embeddable analytical database. *Proceedings of the VLDB Endowment*, 12(12): 2216–2219.

Rocklin, M. 2015. Dask: Parallel computation with blocked algorithms and task scheduling. *Proceedings of the 14th Python in Science Conference*, 2015: 130–136.