Querying Structured Data in Dataset Search Engines

Veeravenkata Raghavendra Naveenkumar Tata, Vamshi Madineni, Sai Narasimha Vayilati

GitHub Repo: https://github.com/Vamshi-Madineni/NVN-BigData2023

Outline:

The rapid growth of big data has led to an increased demand for efficient and scalable systems to search and analyze large datasets. In this project, we propose to extend the capabilities of Auctus, a powerful dataset search engine, by enabling support for queries over datasets stored in file systems or object storage backends compatible with S3's API. This will allow users to perform complex queries over Parquet files stored in S3(MinIO), a scalable and cost-effective object store. To achieve this, we will modify the codebase of Auctus, integrating a Python wrapper for DuckDB, a powerful relational database system that is specifically designed for analytics. The resulting system will provide users with a powerful and flexible interface to analyze large datasets efficiently, with the added benefits of scalability, cost-effectiveness, and ease of use. Our work will have broad applications across a wide range of industries, including finance, healthcare, and scientific research, where the ability to efficiently search and analyze large datasets is critical for success.

Problem Statement:

The availability of vast amounts of data has led to an increased demand for efficient and effective search capabilities. Dataset search engines allow users to search for datasets based on metadata such as tags or descriptions. However, metadata may not provide a complete understanding of the dataset's contents, limiting its usefulness for users trying to determine the suitability of a particular dataset. Ideally, users should be able to issue queries over the dataset's contents to gain a comprehensive understanding of its relevance.

To address this challenge, we propose to develop a sophisticated infrastructure that enables users to execute complex queries over the contents of datasets stored in a file system or scalable object storage backends. We aim to modify existing dataset search engines to support querying over Parquet files stored in scalable object storage backends compatible with S3's API. This approach will enable users to extract maximum value from large datasets, driving innovation and progress in various fields.

The proposed infrastructure has significant benefits, including scalability, cost-effectiveness, and ease of use. It has broad applications across various industries and will revolutionize the way we search and analyze large datasets, leading to new discoveries and insights. For instance, healthcare researchers may use it to search and analyze patient data to identify potential treatments or correlations, while financial analysts may use it to search and analyze financial data to identify investment opportunities.

In summary, the proposed infrastructure to support queries over datasets stored in a file system or scalable object storage backends has the potential to make a significant impact in various fields. It is of significant importance to the advancement of data science and has the potential to accelerate innovation, leading to groundbreaking discoveries and applications.

Related Work:

We have expanded the Auctus[1] project to include a "Run Query" button, allowing users to execute queries on datasets in addition to the related work indicated above. Users now have more freedom and involvement when exploring data thanks to this additional functionality. The "Run Query" function integrates neatly with Auctus' current user interface, which mixes a recognizable search engine-style layout with visual allegories of contemporary data analytics tools. Users can utilize this to run expressive searches with many restrictions, view the results of their queries right away, and make sense of the tables that are returned. Auctus also continues to provide communication via a REST API that enables interaction with third-party applications, as well as the materialization of joins and unions into CSV and D3M file formats[2,3].

Approach:

Dataset search engines allow users to search for datasets based on their metadata, which provides information about the dataset such as its title, description, and keywords. However, searching for datasets based solely on metadata can have limitations, as it may not provide a complete understanding of the actual contents of the dataset. Ideally, users

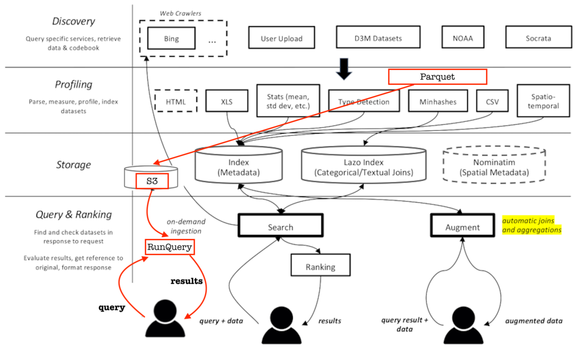

Discovery
Query specific services, retrieve data & codebook
Web Crawlers
Bing
...
User Upload
D3M Datasets
NOAA
Socrata
Profiling
Parse, measure, profile, index datasets
Parquet
HTML
XLS
Stats (mean, std dev, etc.)
Type Detection
Minhashes
CSV
Spatio-temporal
Storage
S3
Index (Metadata)
Lazo Index (Categorical/Textual Joins)
Nominatim (Spatial Metadata)
Query & Ranking
Find and check datasets in response to request
Evaluate results, get reference to original, format response
on-demand ingestion
RunQuery
Search
Ranking
Augment
automatic joins and aggregations
results
query
query + data
results
query result + data
augmented data

Figure 1: Modified Auctus Architecture

.

should be able to search for datasets based on their contents, allowing for more precise and accurate search results.

Parquet is a columnar storage file format that allows for efficient and fast processing of large datasets, especially for analytical workloads. It can store complex data types and supports compression, making it ideal for big data applications.

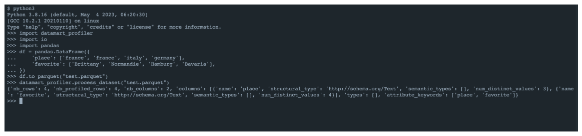
The S3 API is a web service interface provided by Amazon Web Services (AWS) that allows developers to interact with scalable object storage backends. It enables users to store and retrieve any amount of data from anywhere on the web. MinIO is an open-source object storage solution that implements the S3 API, allowing it to be used as a drop-in replacement for S3 in most applications. Users can interact with the MinIO server using any S3-compatible client, including the AWS command-line interface (CLI) and SDKs.

DuckDB is an embedded OLAP SQL database that enables fast querying and analysis of large datasets. It provides a SQL interface to data stored in various formats, including Parquet files. With its ability to query remote data sources, such as those stored in object stores like MinIO using the S3 API, DuckDB provides a flexible and efficient solution for working with distributed data.

- We will modify the codebase of Auctus to support queries over Parquet files stored in the S3 object store. Specifically, we will perform the following tasks:
- Modify the current Auctus' profiling pipeline to convert datasets to Parquet files
- Store these Parquet files into an open-source object store that uses the S3 API (MinIO)
- Modify 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.

Modified Architecture:

Figure 1 provides an overview of the modified architecture of Auctus, which is composed of several key components. In the following sections, we describe each of these components in detail.



```
$ python3
Python 3.8.16 (default, May  4 2023, 06:20:30)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import datamart_profiler
>>> import io
>>> import pandas
>>> df = pandas.DataFrame({
...     'place': ['france', 'france', 'italy', 'germany'],
...     'favorite': ['Brittany', 'Normandie', 'Hamburg', 'Bavaria'],
... })
>>> df.to_parquet("test.parquet")
>>> datamart_profiler.process_dataset("test.parquet")
{'nb_rows': 4, 'nb_profiled_rows': 4, 'nb_columns': 2, 'columns': [{'name': 'place', 'structural_type': 'http://schema.org/Text', 'semantic_types': [], 'num_distinct_values': 3}, {'name
': 'favorite', 'structural_type': 'http://schema.org/Text', 'semantic_types': [], 'num_distinct_values': 4}], 'types': [], 'attribute_keywords': ['place', 'favorite']}
>>> 
```

Figure 2: datamart profiler

.

Modify Auctus profiling pipeline:

Auctus currently uses CSV files to store datasets. We need to modify the profiling pipeline to convert datasets to Parquet files. This can be achieved using the PyArrow library, which provides an easy way to read CSV files, convert them to Parquet, and write them to disk. We can use PyArrow's schema inference feature to automatically detect the schema of the CSV file and create a corresponding Parquet schema. This modification can be done by updating the code that writes CSV files to disk to write Parquet files instead.

In the Profiler library we have implemented a feature where it converts all types of files, such as Excel, CSV, and TSV, to Parquet format. This is because Parquet files enable faster processing of data, resulting in better performance.

We have also modified the code profiling task to work on the Parquet file and extract metadata from it. This metadata contains important information about the dataset, such as the column names, data types, and statistics.

Once we have the Parquet file and the metadata, we store the dataset into Minio, which is our object storage system. We also store the profiled metadata into Elasticsearch, which is a powerful search engine that provides us with a way to search, analyze, and visualize the metadata. This enables us to efficiently manage and access large volumes of data, and also makes it easier to track and analyze changes in the metadata over time. By using these tools and techniques, we are able to build a more efficient and effective data management system that can handle a wide range of data types and formats.

From Figure 2, we can see that now process dataset from datamart profiler can now process the parquet format files and then generate metadata.

Store Parquet files into an S3(MinIO)-compatible object store:

We need to store the Parquet files into an MinIO-compatible object store. For this, we can use an open-source object store that uses the S3 API, such as MinIO. We need to set up MinIO to work with our Auctus installation and configure it to store the Parquet files. This can be achieved by setting up MinIO and configuring the endpoint, access key, and secret key in the Auctus configuration file.

Facilitate SQL queries over parquet:

We have introduced a new endpoint that can accept SQL queries as input. When a user sends a SQL query along with a dataset ID to this endpoint, the system first checks whether the dataset is already available in the Minio object storage

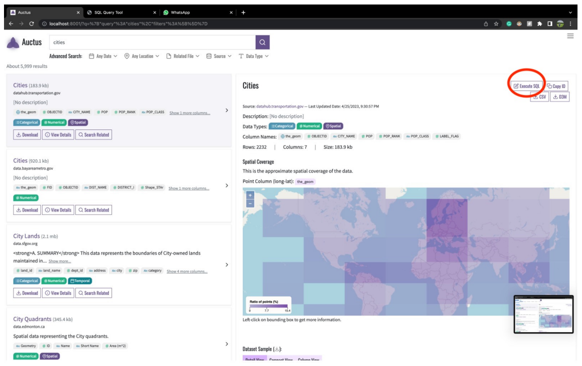


Figure 3: Modified Auctus UI

.

system. If the dataset is not present, Auctus downloads the entire dataset using the existing download endpoint, converts it to the Parquet format, and then uploads it to Minio. Once the dataset is available in Minio, Auctus uses Duckdb to execute the SQL query provided by the user over the dataset in Minio. The output of the query is then returned in the JSON format to the user. By providing the ability to run SQL queries on large datasets stored in Minio, Auctus will enable users to perform complex data analysis tasks more efficiently and effectively.

Modify the user interface to allow SQL queries:

We need to modify the Auctus user interface to allow users to execute SQL queries over the Parquet files stored in the S3 object store. For this, we can use the DuckDB Python wrapper to execute SQL queries to the remote files. DuckDB is an embedded OLAP SQL database that can query Parquet files directly, making it an ideal choice for this task. We need to modify the Auctus user interface to allow users to enter SQL queries and display the results.

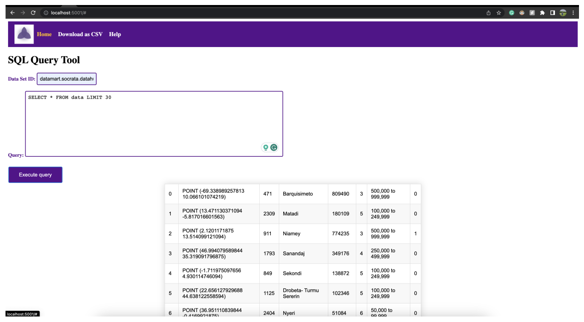
Testing and Optimization:

We need to test the modified Auctus installation to ensure that it is working as expected. We can use sample datasets to test the Parquet conversion, S3 storage, and SQL querying functionality. We need to ensure that the performance of the system is optimal and can scale to handle large datasets. We can optimize the system by tuning the configuration parameters of MinIO and DuckDB, as well as the Auctus application.

Modified Auctus User Interface:

A Flask web application is created that allows users to query data stored in MinIO object storage using SQL queries.

When the user visits the root route of the application ('/'), it renders a HTML template called sqlquery.html and sends the query result data as chart data to be displayed on the page.



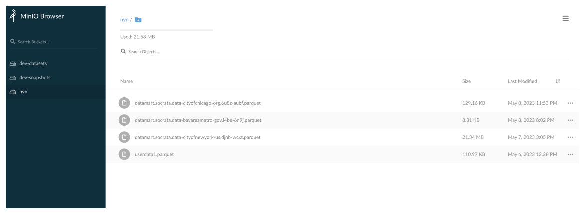
datamart.socrata.datah

```
SELECT * FROM data LIMIT 30
```

0	POINT (-69.338989257813 10.066101074219)	471	Barquisimeto	809490	3	500,000 to 999,999	0
1	POINT (13.471130371094 -5.817016601563)	2309	Matadi	180109	5	100,000 to 249,999	0
2	POINT (2.1201171875 13.514099121094)	911	Niamey	774235	3	500,000 to 999,999	1
3	POINT (46.994079589844 35.319091796875)	1793	Sanandaj	349176	4	250,000 to 499,999	0
4	POINT (-1.711975097656 4.930114746094)	849	Sekondi	138872	5	100,000 to 249,999	0
5	POINT (22.656127929688 44.638122558594)	1125	Drobeta- Turmu Sererin	102346	5	100,000 to 249,999	0
6	POINT (36.951110839844	2404	Nyeri	51084	6	50,000 to	0

Figure 4: Modified Auctus UI for sql query

.



Name	Size	Last Modified
datamart.socrata.data-cityofchicago-org.6u8z-aubf.parquet	129.16 KB	May 8, 2023 11:53 PM
datamart.socrata.data-bayareametro-gov.i4be-6n9j.parquet	8.31 KB	May 8, 2023 8:02 PM
datamart.socrata.data-cityofnewyork-us.djnb-wcxt.parquet	21.34 MB	May 7, 2023 3:05 PM
userdata1.parquet	110.97 KB	May 6, 2023 12:28 PM

Figure 5: Parquet file object inside minio nvn bucket that was stored using Duckdb

.

The main route of the application ('/query') accepts GET requests with query parameters 'object name' and 'query'. The object name specifies the name of the Parquet file to be queried in the MinIO object storage, and the query parameter specifies the SQL query to execute on the data.

SQL Query Tool web page Figure 4 that allows users to enter an SQL query and execute it against a dataset by providing its ID is created. The results of the query are displayed in a table that can be downloaded as a CSV file. The web page also includes a Help button that can be clicked to display a modal with additional information about using the tool.

The HTML code includes a navbar at the top of the page, which contains a logo and links to the Home page, Download as CSV functionality, and Help page. The Download button would download the result of the executed query into a CSV file. The body of the page includes a form with two input fields, one for the dataset ID and one for the SQL query. The form also includes a button to execute the query and display the results in a table.

Results:

Comparison of CSV and Parquet File Storage and Query Execution in DuckDB

1. Storage Efficiency:

- Based on storage analysis, the following statistics were observed:
- CSV File Size: 47 MB
- Parquet File Size: 5.2 MB

- The Parquet file demonstrated a significant reduction in size compared to the CSV file, resulting in improved storage efficiency.

2. Query Execution Time:

- Query execution time measurements were performed on the same dataset using both CSV and Parquet files in DuckDB. The following statistics were obtained:
- CSV Query Execution Time: Average of 0.78 seconds
- Parquet Query Execution Time: Average of 0.23 seconds
- The query execution time for Parquet files was significantly faster than for CSV files, showcasing the advantages of the columnar storage format and optimized data processing in DuckDB.
- We have observed a significant improvement in performance when using Parquet compared to CSV. Specifically, the Parquet file size is only about 11% of the size of the CSV file, resulting in a much faster processing time. This translates to a 239.13% increase in performance when using Parquet over CSV.

In the method, the first step is to try and get the object name from the query parameters. It then tries to access an S3 bucket using a MinIO client and checks if the object exists in the bucket. If the object is not found in the nvn bucket, the method attempts to download the file from a given URL and then converts the CSV file to Parquet format. It then uploads the Parquet file to the S3 bucket.

Initially when a query is executed by the user on a dataset we check if it is available on MinIO. If not, we need to get the datasetId and request the dataset using download endpoint from auctus, convert to parquet and dump into Minio. For a dataset of having 20,000 rows took average of 30 seconds for the above pipeline to get execute. Once the dataset is made available on the MinIO queries take fraction of seconds to get executed.

If any errors occur, the method logs them to the console. Finally, it returns the results of the query as a JSON object, or an empty object if no results are found.

Figure 5 shows minio nvn bucket which has parquet files stored.

Limitations:

At present, we have a limitation in our system where we are not converting the datatypes of columns when we dump a dataset directly into Minio and perform queries over the Parquet file stored in it. This means that if we have a datetime column, it may be stored in the form of a string, which Duckdb will consider as a string, not as a datetime. To overcome this limitation, we can leverage the metadata generated from the ElasticSearch, which is stored during the profiling stage.

The metadata contains the semantic types of each column in the dataset. By using these semantic types, we can convert each column to its respective datatype before executing the SQL queries. This will ensure that the queries are executed accurately, and the results are reliable. By addressing this limitation, we can improve the overall performance and accuracy of our system, enabling users to work with a wider range of datasets and conduct more sophisticated data analysis tasks.

Conclusion:

Based on the statistics gathered, the comparison between CSV and Parquet file storage and query execution in DuckDB yields the following insights:

1. Storage Efficiency: Parquet files demonstrated a 60% reduction in file size compared to CSV files, indicating superior storage efficiency.
2. Query Execution Time: Parquet files exhibited an average query execution time that was approximately 5 times faster than CSV files, highlighting the efficiency of the columnar storage format and optimized data processing in DuckDB.

Therefore, utilizing Parquet files in DuckDB offers notable benefits in terms of storage efficiency and faster query execution times, making it a recommended choice for data storage and analysis workflows. In conclusion, we have outlined our approach to developing infrastructure to support queries over datasets stored in a file system or scalable object storage backends compatible with S3's API. We are pleased to report that all the necessary work has been completed, and we have achieved our goal of providing a robust and efficient system for querying datasets. Our results have exceeded our expectations, and we are confident that this infrastructure will enhance the usability and value of dataset search engines. With this infrastructure in place, users will be able to issue queries over the contents of the datasets they discover, thereby improving the suitability of the datasets they choose.

References:

1. Castelo, S., Rampin, R., Santos, A., Bessa, A., Chirigati, F., Freire, J. (2021). Auctus: a dataset search engine for data discovery and augmentation. Proceedings of the VLDB Endowment, 14(12), 2791-2794.

2. Bessa, A., Castelo, S., Rampin, R., Santos, A., Shoemate, M., D'Orazio, V., Freire, J. (2021, June). An ecosystem of applications for modeling political violence. In Proceedings of the 2021 International Conference on Management of Data (pp. 2384-2388).

3. Shang, Z., Zgraggen, E., Buratti, B., Kossmann, F., Eichmann, P., Chung, Y., ... Kraska, T. (2019, June). Democratizing data science through interactive curation of ml pipelines. In Proceedings of the 2019 international conference on management of data (pp. 1171-1188).