Yelp Data Analysis: Comparing Tools for Business Insight

Fantillo Joshua
Simon Fraser University Graduate School
Burnaby, Canada
joshua_fantillo@sfu.ca

Abstract

To determine important business trends and consumer preferences, this project compares different data processing techniques for efficiency. It does this by analyzing the Yelp dataset. Important discoveries show that Spark Dataframes and Spark SQL perform better than more conventional tools like Pandas, especially when handling big datasets. The report shows the significance of utilizing advanced Big Data methods for scalable data analysis and offers suggestions for local company marketing initiatives.

I. INTRODUCTION

To help local companies in their marketing plans, we are digging into the Yelp Dataset[1] in this project. Our focus covers a number of fundamental concerns:

Business Discovery: Determining particular business categories in various places in order to better understand the marketplace.

Evaluation of Services: Providing services that are continuously acclaimed highly, taking into account the interests of the customers.

Location Assessment: Identifying the areas that are highly rated for a given service.

The main goal is to determine the most effective data processing method for local companies to use so that they can benefit more from directing their advertising efforts toward tourists from other regions or the local population. This involves a thorough analysis of review trends from both groups using appropriate data processing tools, providing businesses with strategic insights on optimal allocation of their marketing budgets.

A key component of our approach is to compare various technological tools for handling this information. We use a comprehensive toolkit that includes Spark[2], SQL, Cassandra[3], and Pandas[4], all of which have been shown to be effective in managing big datasets. The project is more than just data analysis; it's also an analysis of comparison to determine which instrument, or set of tools, provides the best accurate and efficient conclusions for our goals.

Every step of the analysis is carried out on a reproducible computing system. This ensures that the processes we use are transparent and accessible to independent verification. To facilitate the replication of our findings by colleagues and other researchers, we provide thorough instructions for every method.

In the end, the project seeks to make it easier for nearby companies to decide what amount of money to devote on advertising. Through the examination of customer feedback patterns and the comparison of the effectiveness of different big data tools, we offer companies a way to make data-driven decisions that optimize both their impact and scope.

II. METHODOLOGY

A. Problem Addressed:

This project aims to extract meaningful insights from Yelp data, with a specific focus on identifying business types, determining the highest-rated services, assessing the best locations for certain services, and analyzing rating gaps between local and out-of-town reviewers. An extensive comparison of different data processing tools is also included in this study, which is crucial for local businesses planning their advertising strategies. Finding the most efficient and productive tools for examining

these Yelp dataset[1] features is the purpose. In the end, this comparative analysis will help businesses make well-informed decisions based on the insights gained from the data by helping them understand not only the insights derived from the data but also the best methodologies for large-scale data analysis.

Technical Setup and Tools Used:

Data Acquisition: We utilized three key files from the Yelp dataset[1]: business, user, and review. Accessing this data involved a straightforward process of downloading it from the Yelp dataset[1] page after registration and agreement to their terms of use.

Data Processing Tools:

- **Spark[2] Dataframes:** Chosen for its ability to handle large-scale data processing efficiently. Spark Dataframes offers a distributed environment that is crucial for processing the vast amounts of data in the Yelp dataset.
- **Spark[2] SQL:** Utilized for its capability to perform complex SQL queries on large datasets. Spark SQL combines the simplicity of SQL with the power of Spark, making it ideal for data analysis tasks in this project.
- **Pandas[3]:** A powerful data manipulation library in Python, Pandas is used for tasks that require a more nuanced approach, particularly when dealing with smaller subsets of data where its in-memory data processing is beneficial.
- **Cassandra[4]:** A distributed database known for its scalability and high availability. Cassandra is used for managing large datasets effectively, ensuring quick data retrieval and handling.

C. Rationale for Tool Selection:

The combination of Spark[2] Dataframes, Spark[2] SQL, and Cassandra[4] was chosen primarily for their scalability and efficiency in handling large datasets. In contrast, Pandas[3] was included to evaluate its performance with big data tools, offering a comparative analysis of traditional dataframes against more advanced big data technologies.

D. Hardware Used:

The project was executed on a Linux Virtual Machine equipped with 16GB of RAM and 8 processors, supplemented by the resources of the course's cluster[5]. This setup provided the necessary computational power and environment to handle the data-intensive processes of the project.

E. Data Organization:

The original dataset files were split up into 8 smaller files to optimize our processing efficiency and take full advantage of our 8-core setup. We were able to evenly distribute the workload among all processors with this approach, which resulted in faster processing and more effective data management.

III. SCRIPT OVERVIEW

get_businesses.py
This script retrieves business information from the Yelp dataset[1]. It filters businesses by a chosen category and location (city or state). Using Spark[2] Dataframes, it sorts the businesses in descending order based on a specified criterion and outputs the results in JSON format to a designated file.

get_most_liked.py
Designed to identify the most favored businesses in a specified location and category. The script employs Spark[2] Dataframes to generate a descending order list of top-rated businesses within the selected category and location, saving the output in JSON format.

get_most_popular.py
This script finds the highest-rated locations (cities or states) for a chosen business category. It uses Spark[2] Dataframes to calculate the average ratings for each location within the specified category, returning a list of locations sorted by their average rating.

get_local_popularity_df.py
Utilizes Spark[2] Dataframes to analyze and compare business ratings based on the reviewer's origin (local or out-of-town) for a specific location. It produces two separate outputs: one listing businesses favored by local reviewers and another for those preferred by out-of-town visitors.

get_local_popularity_sql.py
Similar to get_local_popularity_df.py, but this version leverages Spark[2] SQL for the analysis. It follows the same process of comparing local and out-of-town reviews for businesses in a specified location, with the results split into two corresponding files.

get_local_popularity_pandas.py
This script mirrors the functionality of get_local_popularity_df.py, but it uses Pandas[3] for data processing. It's particularly suited for smaller datasets due to Pandas'[3] in-memory processing capabilities.

get_local_popularity_cass.py
Conducts the same analysis as the previous scripts but utilizes Cassandra[4] in conjunction with Spark[2] Dataframes for data storage and retrieval. This script is ideal for handling large-scale data efficiently.

load_data_cassandra.py
This script is responsible for loading the Yelp dataset[1] into a Cassandra[4] cluster[5]. It uses Python3[6] and Cassandra[4] to batch-upload the data, setting the stage for subsequent Cassandra-based analysis.

load_data_cassandra_spark.py
Similar to load_data_cassandra.py, but this version uses Spark[2] in conjunction with Cassandra[4] for efficient data loading. It's particularly useful for handling large volumes of data.

IV. RESULTS

get_businesses.py
Averages 7.9 seconds over 10+ executions.

get_most_liked.py
Takes about 7.2 seconds on average over 10+ executions.

get_most_popular.py
On average, 6.8 seconds over 10+ executions.

Comparison of Local Popularity Scripts

get_local_popularity_df.py
Averages 40.2 seconds over 10+ executions.

get_local_popularity_sql.py
Averages 43.4 seconds over 10+ executions.

get_local_popularity_pandas.py
Averages 1:35.3 minutes per run over 10+ executions, and only tested on 5% of the dataset.

get_local_popularity_cass.py
Averages 1:05.2 minutes on average over 10+ executions.

Data Loading Scripts

load_data_cassandra.py
Averages 17:52.6 minutes over 5+ executions.

load_data_cassandra_spark.py
Averages 18:35.5 minutes on average over 5+ executions.

V. DISCUSSION

get_businesses.py
This script efficiently reads the business data file from the Yelp dataset[1], leveraging Spark's[2] capabilities to quickly load and filter data based on the selected location and/or category.

get_most_liked.py
Similar to get_businesses.py, this script focuses on the business data file, applying user-defined filters for location and/or category. Its efficiency is attributed to Spark's optimized data processing.

get_most_popular.py
Reads the business data file and aggregates average ratings for a specified category and location, outputting the highest-rated locations. The speed reflects Spark's[2] capability in handling aggregate functions.

Comparison of Local Popularity Scripts

These scripts share a common function: merging user, business, and review datasets to analyze whether businesses receive more favorable reviews from locals or out-of-town visitors. The DataFrame and SQL scripts show nearly identical execution times, while the Cassandra[4] script lags behind by about 15 seconds. A key factor in improving the efficiency of each script is the caching of the dataframes after loading and prior to filtering. This optimization alone cuts down the processing time by approximately 45 seconds for each script.

There's a potential for even greater efficiency. Theoretically, if a Cassandra[4] cluster[5] was set up on our local machine with matching specifications, we might see similar performance times across all scripts. The Cassandra[4] script primarily differs in its method of data retrieval, suggesting that this is the main contributor to its slightly longer execution time compared to the DataFrame and SQL scripts.

The experience with the Pandas[3] script highlighted some challenges specific to handling large datasets. Our datasets contain around 500,000 rows each, a volume that Pandas struggled to process efficiently. The maximum number of rows we successfully loaded was 40,000, which took about 45 minutes. Any attempt to process over 100,000 rows resulted in processing times exceeding 1.5 hours and eventually led to program crashes. As a result, we had to restrict the input to 15,000 rows, which Pandas completed in about 90 seconds. Notably, this volume only makes up 5% of our entire dataset, highlighting Pandas'[3] limitations when it comes to large datasets.

Data Loading Scripts

In our experiments with data loading scripts, we explored both Spark[2] and non-Spark methods. The initial non-Spark approach did not use batching, resulting in exceedingly long loading times. Introducing a batch size of 200 significantly improved the performance. Interestingly, this optimized non-Spark loader slightly outperformed the Spark[2] loader in terms of the time required to load and write the dataset. Although the differences in execution times were minimal, the non-Spark loader consistently finished a few seconds faster on average.

VI. CONCLUSION

Our analysis revealed a clear advantage in using Spark[2] and Cassandra[4] for data processing over traditional methods like Pandas[3], particularly evident in our local popularity assessment. The Spark[2] DataFrame method proved to be the most efficient, closely followed by Spark[2] SQL, with both significantly outperforming the standard Pandas[3] approach used for smaller datasets.

This efficiency gap highlights the importance of choosing appropriate tools for large-scale data analysis. For tasks involving extensive datasets, such as the full Yelp dataset[1], Big Data technologies like Spark[2] DataFrames, SQL and Cassandra[4], are not only preferable but necessary for effective and scalable data processing. In contrast, traditional methods like Pandas[3], while useful in smaller contexts, are less suitable for handling Big Data challenges.

VII. PROJECT SUMMARY

- Getting the data: 3
- ETL: 0
- Problem: 7
- Algorithmic Work: 1
- Bigness/Parallelization: 7
- UI: 0
- Visualization: 0
- Technologies: 2

REFERENCES

[1] Yelp, Yelp Open Dataset (2023), Retrieved November 20th, 2023 from https://www.yelp.com/dataset

[2] The Apache Software Foundation, Apache Spark (2018), Retrieved November 20th, 2023 from https://spark.apache.org/

[3] The Apache Software Foundation, Apache Cassandra (2023), Retrieved December 1st, 2023 from https://cassandra.apache.org/_/index.html

[4] NumFOCUS, pandas (2023). Retrieved November 20th, 2023 from https://pandas.pydata.org/

[5] ggbaker, CMPT 732 G1 Cluster (2023), Retrieved September 11th, 2023 from https://coursys.sfu.ca/2023fa-cmpt-732-g1/pages/Cluster

[6] Python Software Foundation, Python3, Retrieved September 11th, 2023 from https://www.python.org/downloads/