CS 411 project track 2 final report

Che-Lin Huang

Wen-Chen Lo

Mengfei Liang

Muxia Yi

University of Illinois at Urbana-Champaign

1. Introduction

This project aims at finding a suitable algorithm to handle millions of data in 5 seconds. So the performance of dealing with large data files is significantly important. This report will introduce how we implement this algorithm and discuss problems occurred during the implementation. In addition, this report comprehensively evaluated our system by comparing with q-text, a system performs similar functions with ours. Finally, this report briefly summarizes our results and discusses future work.

2. System designing

2.1. Overall architecture of the system

The overall architecture of our system is designed as figure 1.

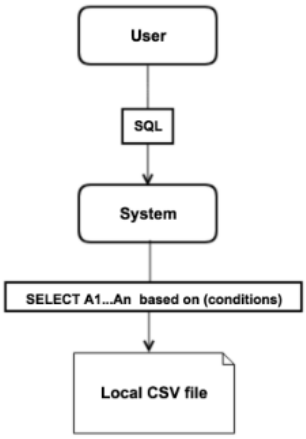

Figure 1. Overall architecture of system

2.2 System product

We use python as our programming language. In order to manipulate data, we import pandas and feather library which provide efficient data type such as data frame to store attributes and data. The environment of the system does not matter much. In general, any personal computer would be able to manage large data through our application, no matter which operating system it runs on. However, we still recommend running our code on an Intel i5 CPU and 8 GB RAM machine to get the best experience.

Our system is easy to handle. First, download our code and our sample CSV films and save all of them into the same disk in your computer. Second, run the code with python or python IDE. Make sure you have already installed the python 3.6 or above version. Third, input your SQL query after the sentence "Type your SQL and press Enter:" appears and wait for the system to print out results.

Our code is available at Github:
https://gitlab-beta.engr.illinois.edu/clhuang2/Ad-hoc-Data-Computing

3. Implementation

Our system provides a command line interface which enables users to input queries so as to retrive data that meets their requirements. The implementation can be divided into several parts: preprocessing, query validation, translate SQL queries, and query optimization.

3.1. Preprocessing

Before our system is available for any query, we preprocess the data first to tackle the inefficient processing of CVS files during queries. In this phase, we read all CSV files one by one and store them into Feather files for later use. Feather file is a kind of high performance and lightweight binary file format for storing data frames which can reduce the I/O overhead for our system.

3.2. Query Validation

We implement a basic validation check for any incoming query which can help to increase system robustness. If any syntax error is detected when parsing the command, our system would prompt the error message and prevent further data processing.

3.3. Translating SQL queries

After receiving user's sql command, our system would translate the SQL commend which consists of SELECT attribute, FROM which tables and WHERE the conditions are.

Besides those three key words, our system also supports the following keywords:

1. Boolean Connector: AND, OR, NOT.

2. mathematical relation as: =, <>, <, <=, >, >=, LIKE, and NOT LIKE

For example: WHERE B.city = Champaign AND B.state = IL AND R.stars = 5 AND P.label = inside

3. Join tables: JOIN

Here is a complete query example:

If users want to find the review id with its stars and the useful level who has at least 4 stars and the level of useful is more than 20 from a CSV film named review-1m, just input: SELECT review_id,stars,useful FROM review-1m.csv WHERE stars >= 4 AND useful > 20.

Step 1: Read all the csv files into dataframes, then save them into feather format files.

Step 2: Read user input and saves parts of the SQL command separately, such as fileList, attributeList, tableprefixList.

Step 3: If user changes the file name to others like using B to represent the file business.csv, replace the orginal file name and save it into tableprefixList as the prefix of each attribute. Or, just save the original file name into the tableprefixList.

Step 4: Read the one or over one feather files needed for the manipulations into dataframes, then into dictionaries.

Step 5：Filter the contents of each tables according to the conditions after keywords WHERE so as to minimize the size of dictionary.

Step 6：Split the dictionary which saves table data into several dataframes, then process the join manipulations on all the dataframes, convert them into one dataframe.

Step 7: Select those attributes specified in the SQL command from this dataframe. End of the code.

3.4. Query Optimization

We have implemented several optimizations in order to reduce the response time. First, when the system load data from the disc, it will trim off the data before doing any manipulation. As soon as the system finishes query parsing, it will immediatly project the required columns into the data frames which can significantly reduce the data size if the required columns are just a small subset of all. in addition, we all know that Join operation can be very time consuming and become the bottleneck of the system. Our system adopt an optimization strategy which try to perform the selection operation in advance and postpone the join operation as possible as we can.

4. Evaluation

4.1. Perform systematic experiments

Sample Queries were performed by using the Yelp Challenge dataset, consisting of four CSV films which are reviews, businesses, photos, and checkins.

1. SELECT review_id,stars,useful FROM review-1m.csv WHERE stars >= 4 AND useful > 20

Result: Returns 1105 results in 2.862765073776245 seconds

2. SELECT B.name FROM business.csv B JOIN review-1m.csv R ON B.business_id = R.business_id JOIN photos.csv P ON B.business_id = P.business_id WHERE B.city = Champaign AND B.state = IL AND R.stars = 5 AND P.label = inside

Result: Returns 6593 results in 3.5023698806762695 seconds

3. SELECT DISTINCT B.name FROM business.csv B JOIN review-1m.csv R JOIN photos.csv P ON B.business_id = R.business_id AND B.business_id = P.business_id WHERE B.city = Champaign AND B.state = IL AND R.stars = 5 AND P.label = inside

Result: Returns 13 results in 2.794286012649536 seconds

4.2. Compare with other similar systems

When compared with other systems which also supports SQL command, our system shows an excellent performance.

Q - Text as Data is able to handle data in CSV and TSV files and any other tabular text files directly. When dealing with ordinary files, q - Text as Data treats them as tables. In addition, q - Text as Data can support all SQL constructs including WHERE, GROUP BY, JOINs. Based on its function, it is a great system to deal with big data. However, it takes extremely amount of time to run large files. For example, running the first query 'SELECT review_id,stars,useful FROM review-1m.csv WHERE stars >= 4 AND useful > 20' in Q - Text as Data would takes 54 seconds, nearly 1 minute. But our system can conduct query within 5 seconds when the data is as large as millions — it is more efficient.

5. Discussion

5.1 Summary

Our system has two significant advantages:

1. Easy to handle.

Being easy to use is a great advantage of our system. First, our system can run on many kind of operating systems. Second, users do not need any specific knowledge to process their data in order to

use our system. All they have to do is to store their data into CSV files and perform SQL queries on those files.

2. Fast speed.

Our system is quite efficient. As introduced in Section 4.1, our system can carry out queries when the film is more than 1 million rows within 5 seconds. Which saves users much time under most of circumstances.

However, our system has some important disadvantages as well. First, our system only supports SELECT-FROM-WHERE query format and JOIN keyword. Aggregate functions such as MAX cannot be carried out with our system. Second, our system does not support subqueries. Also, only basic Boolean operations and arithmetic operators as well as LIKE keywords can be done within our system. In summary, our system is very strict with syntax of queries which users input.

5.2 Future work

There is still much room for our system to get improved.

a) Aggregation functions and subqueries. The next step for our team is to make our system able to support aggregation functions and subqueries, which means we need to handle more complicated SQL command structure.
b) Efficient indexing. Using index can completely speed up the query. When doing our project, our team tried different kinds of index to speed up our queries. If given more time in the future, our team would try to implement B-tree to achieve range search more efficiently. What we really see is that how index plays an indispensable and miraculous role in improving the performance of data processing. Such technique can be also helpful in the future whenever we need to do something with data.

Through this project, we can learn how the database works under the hood, instead of just knowing how to use it. Such hands-on experience can help us better understand how to design a good database system, how to process the data more efficiently, and how query strategy (algorithm) can significantly increase the performance.

6. References

[1] Garcia-Molina, H., Ullman, D.J., Widom, J. (2009). Database system: The complete book, Second edition. *Pearson education Inc.*
[2] q-Text as Data, available at: http://harelba.github.io/q/