UPPSALA
UNIVERSITET

Project for Data Engineering I

March 2022

Ivar Blohm
Liang Cheng
Melissa Nicole Athanasiou
Xiaoxia Liu

1. Background

Reddit is a social media website. Subreddits represent topic-specific communities where users post, comment, and vote. The usually lengthy nature of subreddits makes Reddit an ideal source of data for studying the behavior of users. Since the majority of users remain anonymous, the website consists of a great number of publicly available comments that can be used for training and testing Natural Language Processing models, including sentiment classifiers (Turean and McKeown). The goal is to predict whether a reddit comment is positive, neutral, or negative.

2. Data Format

The test data is semi-structured and therefore JSON format is used which can be stored in NoSQL directly. JSON is used to exchange data between different platforms in a way that is compatible with both systems by universal programming concepts (Bassett L.). It does not depend on any format, it is based on characters for representing data such as numbers and words. The plain text can easily be converted to suit the needs of the server. Even though the format does not support binary data and numbers are stored in their decimal notation, this limitation will not affect the test data. However, empty vectors, empty lists and empty data frames are represented the same way which means that an additional interpretation is needed (Ooms J.).

The format of the training data is CSV. Some advantages of CSV format include the straightforward information schema in a tabular format. Moreover, the format is human and machine-readable. The basic, non-proprietary format of CSV and the generally small size makes it fast to handle (Carvalho P. et al.). However, this simplicity can cause some issues. For instance, the semantic and syntactic interpretation of CSV files can be difficult. Furthermore, it can be hard to get an overview of the structure and the content of a CSV file. Despite its good reputation, this format does not have a formal specification (Carvalho P., et al.) like the ability to distinct text and numeric values or a standard way to represent control characters and binary data.

3. Computational Experiments

3.1. System Architecture

The system is mainly structured in 3 parts: data source, storage layer and processing layer. Data source is the part that contains the raw data which is going to be processed, and the raw data here is reddit comment data (JSON type). There are two parts in the storage layer: HDFS and MongoDB. For HDFS, it’s used to store the raw data and training data, as for MongoDB, it’s used to store the outcome of data analysis. The processing layer consists of the Spark cluster, and there are 1 master and 3 workers in this cluster to process the data.

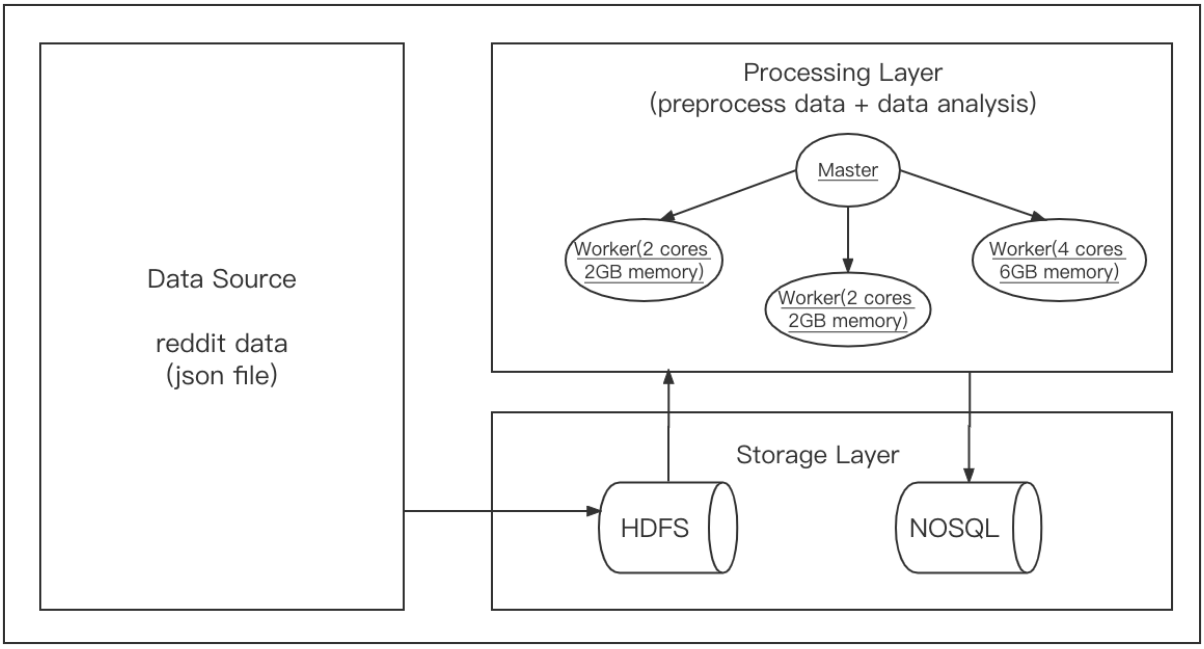

Figure 1: Overview of chosen system architecture

3.2. HDFS

The Hadoop Distributed File System is a highly fault-tolerant distributed file system, designed to run on commodity hardware. Because of its properties and that it works well together with Apache Spark it was suitable for this project. The JSON files, which were desired to be analyzed, were downloaded as zip-files and unzipped on the virtual machine before being uploaded to HDFS. For this project HDFS was only installed on the master node and not on the three workers. Since the size of the files were 1.6GB, 3.1GB and 11GB it wasn't necessary to install HDFS on the workers as well, the analysis part could be done with HDFS in standalone mode.

3.3. Spark

3.3.1. Spark Cluster

There are 1 master and 3 workers in the Spark cluster, the configuration of these three workers are worker1 (2 cores, 2GB memory), worker2 (2 cores, 2GB memory), worker3 (4 cores, 6GB memory).

3.3.2. Data Analysis

3.3.2.1. Data preprocessing:

There are three main parts in the preprocessing: lowercasing all text, dropping the useless words, and filtering the stop words. In sentiment analysis, we need to compare every word in the text. In some cases, the words containing lower or upper letter might have the same meaning but the program would recognize them as different words, so the word-lowercasing is

necessary. And then some of the comments would include useless text such as the url link (http://…), user name (@user), and so on. These things have no impact on predicting the sentiment, thus, we ought to delete them before the analysis. The last part is deleting the stop words. In this step, we make a list of the most common stop words and filter them out of the comment. Although this list doesn't cover all stop words, it should be thorough enough to achieve our goal.

3.3.2.2.Machine learning model:

In the sentiment analysis, we use Logistic Regression to do the classification (3 categories: positive, neutral, negative). First, we divide the preprocessed comment into words and compute the TF (Term Frequency). TF can be viewed as the occurrences of words in the whole text, theoretically speaking, the more the word occurs, the more important it is. And then, we calculate the IDF (Inverse Document Frequency), which is an importance adjustment factor for every word. In some cases, we can't conclude that the words are equally important because they have the same occurrence in the text, and in this situation IDF is able to let us know which of them is more significant. After these calculations, word list, TF and IDF are put in the Logistic Regression model to finish the classification.

3.3.3. Scalability Test on Spark Program

When looking at strong scalability the question under consideration is how the runtime decreases if more resources are added, for a fixed data size. When performing strong scalability the actual trade-off that needs to be considered is the cost of added hardware versus the decreased execution time. Furthermore, weak scalability examines how the runtime affects as a function of increasing hardware for a problem size that is growing. For this method, the results would indicate how much the workload could grow and still preserve the same execution time.

3.3.3.1 Horizontal scalability:

To test the strong and weak scalability horizontally, we set up four experimental groups with the number of executors: 1, 2, 3, 4, and each executor has 2 cores and 2GB memory. Every experimental group runs all three test datasets, and the results are shown below. The four plots correspond to data-loading time, model-training time, testing time (time for model to make prediction), and total running time for each experimental group.

In data-loading, the performance would be better once the extra executor is added. And the larger the data set size is, the more apparent this phenomenon is. When the dataset size is small enough such as smaller than 3.1 GB, the improvement probably would be too subtle to observe. As for model training, adding executors can shorten the training time greatly and the size of test data basically has no impact on it. Theoretically speaking, storing the big-size-data locally should have a negative effect on the performance of the system, but it doesn't go that way in this case, perhaps because the size of data we used is not large enough to have an actual impact. For the testing (prediction-making), unlike the other two parts, the difference between shortest time and longest time is even less than 0.05s, so neither the number of executors nor the size of test data can significantly affect the performance. In some cases,

adding an executor doesn't really shorten the testing time. The reason behind this probably is that the job of making predictions only needs a small amount of resources to run the computation, which could be much less than the resource assigned to each executor.

In short, we find out that the overall performance of the system can be improved by adding more executors though the situation might be different for each separated work. When the data size is fixed, each time one executor is added, the workload for each executor decreases. Under this situation, the running speed of this program increases, indicating it has good strong scalability. But the running time decreases at a different rate so the strong scalability is not perfect. As for the weak scaling, when both the data size and the number of executor increase, the operating efficiency is not maintained at a fixed level. Therefore, the system doesn't perform well in weak scaling.

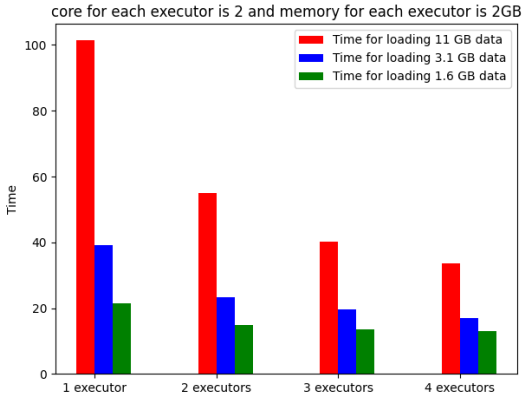

Figure 2: Scalability test of data-loading

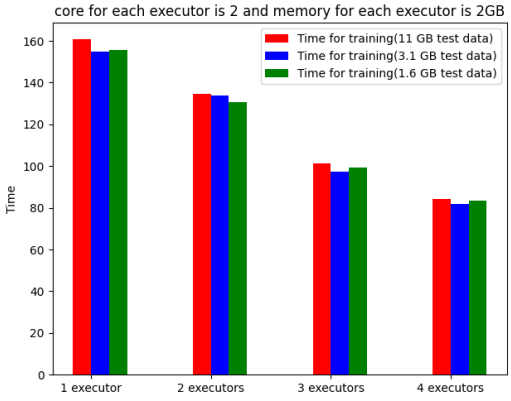

Figure 3: Scalability test of model-training

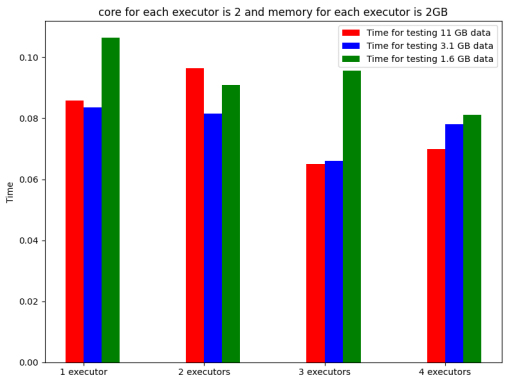

Figure 4: Scalability test of model-testing

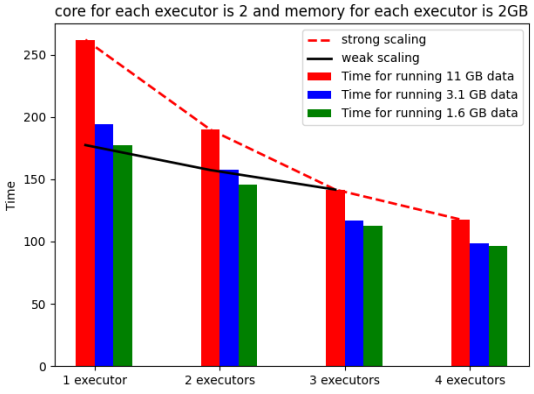

Figure 5: Scalability test of Spark program

3.3.3.2. Vertical scalability:

To test the strong and weak scalability vertically, we set up three experimental groups. All experimental groups have one executor but their assigned resources are different (2 cores + 2GB memory, 3 cores + 4GB memory, 4 cores + 6GB memory). This test also uses the same data as the horizontal scalability test, and the results are shown in the following figures. The four

plots correspond to data-loading time, model-training time, testing time, and total running time for each experimental group.

About the data-loading part, the performance is affected by both the data size and the resources assigned for each executor. When an executor has more cores and memory, the loading time is shorter, and the larger data size would lower the loading speed. As for model-training, the training times are basically the same as long as the executor has the same resources, which means that the test data size has no impact on the performance of this part. In the testing part, the situation is the same as the testing in horizontal scaling, even the resources of an executor or data size varies greatly, the testing time doesn't really follow any specific pattern to change.

Based on the four charts, we are able to conclude that the more resources each executor has, the better performance they have. When the data size is fixed and the amount of resource of each executor increases, it's clear that the running efficiency is improved but the different increasing rate also indicates that the strong scalability is not ideal. And the inconsistent change when both the data size and the amount of resource increase tells us that it's not good at weak scaling.

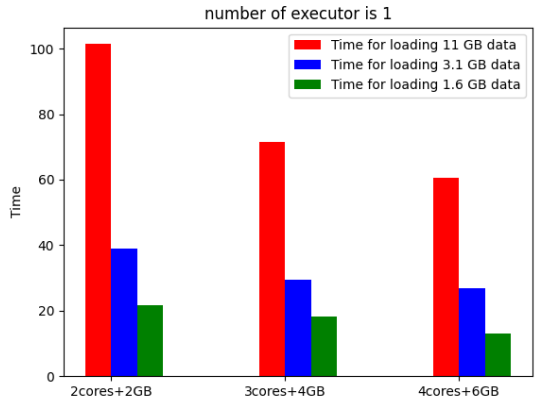

Figure 6: Scalability test of data-loading

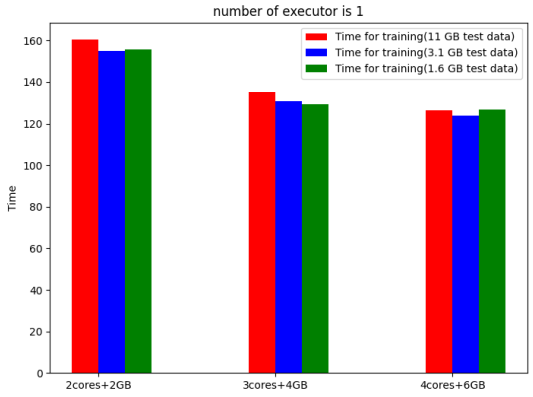

Figure 7: Scalability test of model-training

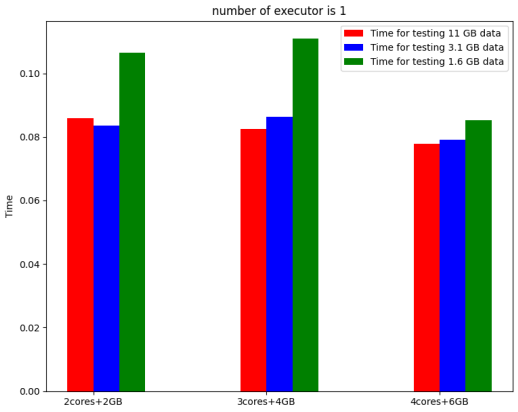

Figure 8: Scalability test of model-testing

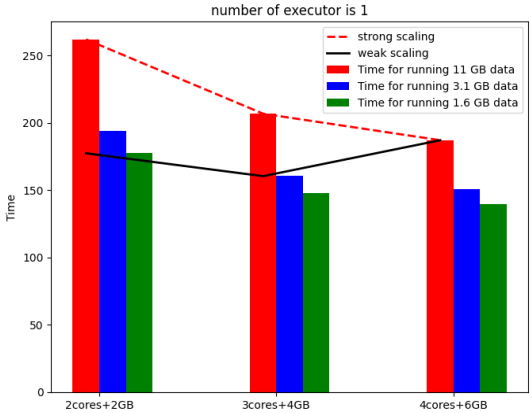

Figure 9: Scalability test of Spark program

3.4. MongoDB

3.4.1. Overview of MongoDB

In MongoDB, we created a database “reddit” to store the cleaned data. We imported the output from different test data into its corresponding collections, which are analogous to tables in relational databases. Thus, we have three collections.

Then we renamed the JSON file and imported it into MongoDB. We can check the number of data and the data format in MongoDB. The documents are stored in the form of field-and-value pairs. MongoDB will automatically generate “_id” as a primary key.

```
test> use reddit
switched to db reddit
reddit> db.result201001.countDocuments({})
2646197
reddit> db.result201001.findOne()
{
  _id: ObjectId("6239dd4838ee412c65b5517e"),
  body: 'Ok so people can donate charity, but how does people playing a video game come into the equation?',
  prediction: 4
}
```

Figure 10: structure of data in MongoDB

3.4.2. Comparison of Different Datasets

Dataset	Records	Used time for import to MongoDB (s)	Import time per 1000 records (s)	Execution Time (s)*
201001	2,646,197	271.421	0.103	3.247
201011	5,256,863	642.607	0.122	35.647
201204	17,559,861	1,513.763	0.086	84.892

** Use case for execution time (finds counts for each prediction values) (eg. result201001): db.result201001.aggregate([{ $group: { _id: "$prediction", count: { $sum: 1} } }]).explain("executionStats")*

We can see that the time required to import into the database is basically stable. It is about 0.1s per 1000 records. Hence, we can ensure the operating efficiency.

3.4.3. Availability

A replica set in MongoDB is a group of mongod processes that maintain the same dataset. Replica sets provide redundancy and high availability. We created a three-member replica set. Since three member replica sets provide enough redundancy to survive most network partitions and other system failures (“Deploy a Replica Set — MongoDB Manual.”).

```
rs0 [direct: primary] test> show dbs
admin   81.9 kB
config  98.3 kB
local    446 kB
reddit  4.45 GB
rs0 [direct: primary] test> rs.conf()
{
  _id: 'rs0',
  version: 1,
  term: 1,
  members: [
    {
      _id: 0,
      host: 'host-192-168-2-82-de1:27017',
      arbiterOnly: false,
      buildIndexes: true,
      hidden: false,
      priority: 1,
      tags: {},
      secondaryDelaySecs: Long("0"),
      votes: 1
    }
```

Figure 11: Configuration of MongoDB after replica set has a primary

3.4.4. Scalability

MongoDB supports horizontal scaling through sharding. Sharding is a method for distributing data across multiple machines (“Sharding — MongoDB Manual”).

We discussed Sharding Architecture in theory but didn’t implement it, since we can query data efficiently right now. To implement the Sharding Architecture, we also need more VMs. We want to set up a sharded cluster with: one Mongos, three Shards (we split our datasets on the result of prediction, which value can be 0, 2, 4.) and Config servers. The Config servers and each Shards all have three replica sets.

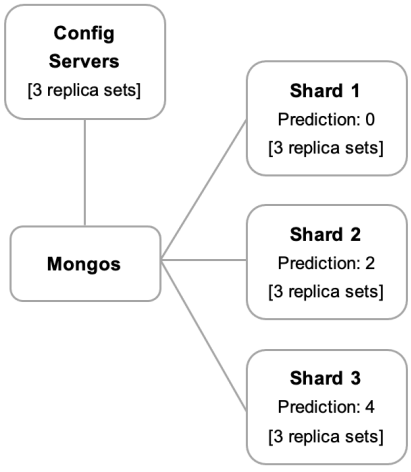

Figure 12: Sharding Architecture

4. Discussion and Conclusion

One of the largest challenges for this project (and similar tasks) is the choice and design of the framework to be used. One needs to consider aspects such as input data, wanted velocity of analysis, volume of data, storage and processing part. This project included several bumps in the road which resulted in a great insight on both Spark and HDFS. Since we were fortunate enough to use the cloud environment for free, an aspect that we didn't encounter is the cost of the framework. The results showed clearly that with more resources the execution time will decrease. But this will always be a trade-off with the cost of the framework, and that is a major part that one needs to consider.

Since HDFS was only installed on the master node and not on the three workers it provided some limitations. The consequence is that a file larger than the volume of the master node (20GB in this case) could not be entirely stored. If one would need to store large files there are two alternatives. The first alternative is to increase the volume/storage of the master node (vertical scalability). This often comes with a high cost for high-end machines and is limited by the capacity of the underlying host. The other alternative is to install HDFS on the workers (horizontal scaling), which is usually a more cost-effective way of scaling since the value of adding another hardware of the same type grows linearly. But for this project the analysis could still be successfully performed with files below 20GB, so installing HDFS only on the master node was sufficient.

Regarding the configuration of Spark cluster, we decided to build a cluster with one master and three workers (2 cores, 2 cores, 4 cores) because of the limited resources on the cloud. After the scalability test, we found that the system has a better performance under horizontal scaling in this eight cores case, so we set up 4 executors and each one with 2 cores for the Spark program. Based on the test result, we knew that the system can perform better if more executors are set or more resources are assigned, but the set-up we eventually chose should be the best solution under the fixed worker-resource.

Code for this project: https://github.com/lmchengliang/Data-Engineering/tree/main/Project

References

(1) Test Dataset is available at http://files.pushshift.io/reddit/comments/
(2) Training Dataset is available at http://help.sentiment140.com/for-students

Turcan, Elsbeth, and Kathleen McKeown. "Dreaddit: A Reddit dataset for stress analysis in social media." *arXiv preprint arXiv:1911.00133* (2019).

Severance, Charles. "Discovering javascript object notation." *Computer* 45.4 (2012): 6-8.
Alessia, D., et al. "Approaches, tools and applications for sentiment analysis implementation." *International Journal of Computer Applications* 125.3 (2015).

Connelly, Aidan, M. Palomino, and V. Kuri. "Lack of consensus among sentiment analysis tools: A suitability study for SME firms." (2021).

Carvalho, Paulo, et al. "Information visualization for csv open data files structure analysis." *IVAPP 2015-6th International Conference on Information Visualization Theory and Applications; VISIGRAPP, Proceedings, March*. 2015.

Gu, Lei, and Huan Li. "Memory or time: Performance evaluation for iterative operation on hadoop and spark." *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*. IEEE, 2013.

Bassett, Lindsay. *Introduction to JavaScript object notation: a to-the-point guide to JSON*. " O'Reilly Media, Inc.", 2015.

Ooms, Jeroen. "The jsonlite package: A practical and consistent mapping between json data and r objects." *arXiv preprint arXiv:1403.2805* (2014).

Krishnan, Hema & Elayidom, M.Sudheep & Santhanakrishnan, T.. (2016). MongoDB – a comparison with NoSQL databases. International Journal of Scientific and Engineering Research. 7. 1035-1037.

Rupali Kaur, Jaspreet Kaur Sahiwal (2019). A review of comparison between NoSQL Databases: MongoDB and CouchDB. International Journal of Recent Technology and Engineering (IJRTE).

“Deploy a Replica Set — MongoDB Manual.” MongoDB Documentation, https://docs.mongodb.com/manual/tutorial/deploy-replica-set/. Accessed 23 March 2022.

“Sharding — MongoDB Manual.” MongoDB, https://www.mongodb.com/docs/manual/sharding/. Accessed 24 March 2022.