Uppsala Univeristy

Data Engineering - 1TD169

Final Project

Author: Group DE-I_12
Fredriksson, Stina
Evelius, Axel
Dahlin, Erik
Recklies, Jannik

November 9, 2024

Contents

1 Background

As social media constantly generates large amounts of data it useful to extract information from the data. This can be done for a wide range of different purposes and with numerous of different ways to handle big data. The main focus is usually to understand trends and behaviours in connection to different topics online or for analyzing language. One example of this is the study of Yiming Zhu et al. (2022) which was conducted to understand how the Ukraine-Russia conflict is depicted on Reddit in terms of keywords, emojis and activity. It utilized the Pushshift Reddit API to perform the data collection. [1]

Another report which relates more concretely to the aim of this project is performed by Tom Willaert et al. (2021). They presented a tool to analyze the trend of frequent words on Reddit for different subreddits from 2005 to 2017. Similarly to the study mentioned previously, Pushshift Reddit was used to collect the data. The data visualization was done using a tool which stores the data using MongoDB. They showcase how the words such as "gamergate" and "redpill" have trended and spread over time through different subreddits. [2]

The dataset which this report is centered around is the "Webis-TLDR-17 Corpus", a collection of almost 4 million processed Reddit posts. The main purpose of this dataset is for usage in training a summarization deep learning model due to the summary attribute of Reddit posts. [3]

The aim of this project is to architect and create a horizontally scalable data processing solution which can be used to answer the following questions regarding the dataset:

- Which are the most popular subreddits?
- What are the most frequently occurring words?
- What are the authors using the most amount of bad words?
- What are the most frequent occurring words of the most popular subreddits?

The solution applied is a Apache Spark cluster with HDFS (Hadoop Distributed File System) for a distributed data storage over the cluster. The scalability of the solution was analyzed by testing horizontal strong and weak scalability properties.

Link to GitHub Repository:
`https://github.com/stinafredriksson/Data-Engineering1-Project`

2 Data Format

2.1 Dataset

The dataset which will be utilized in this study is the "Webis-TLDR-17 Corpus", the data format of this set is JSON. The JSON objects represents each a Reddit post and follows the schematic of

```
{"author":<string>
"body":< string>
"normalizedBody":<string>
"content":<string>
"content_len":<long>
"summary":<string>
"summary_len":<long>
"id":<string>
"subreddit":<string>
"subreddit_id":<string>
"title":<string>}
```

[3] JSON is a popular format and suitable for the dataset. The perks of JSON format is that the data is relatively easy for humans to read and write but also for computers to parse. It can store different types of data types consisting of array, boolean, null, number, object and strings. The data can also be stored nested which differs JSON from flat storage ways. Additionally, the JSON format is not dependent on the programming language handling the data. This quality together with the big range of applications which supports it makes the format very flexible. The drawback of JSON are primarily the non existent error handling and security concerns. [4]

This format type is suitable for the Reddit dataset because of the qualities mentioned earlier but also since the Reddit data is so called semi-structured data. Such kind of data must be stored in files rather than in traditional relational data bases. There are also a lot of APIs which are JSON compatible. An alternative formats to JSON is XML files. It has similar properties as JSON, however it is less concise and takes more time to process.[5]

2.2 Storage

Hadoop Distributed File System(HDFS) were chosen as the storage solution for its redundancy and scalability when working with large expanding datasets. The redundancy or fault tolerance comes from the replication factor in HDFS where the data is replicated and distributed on different nodes in case of node-failure. HDFS can be scaled by adding or removing Datanodes accommodating for bigger or smaller file sizes[6].

3 Computational Experiments

3.1 Design of the distributed system

HDFS were used based on the motivation in subsection 2.2 as the storage solution with YARN(Yet Another Resource Negotiator) to manage and allocate resources across the

cluster, YARN consists of two main components, scheduler and applications manager. Which together allows for efficient usage of the DFS-cluster resources. For the executor Apache Spark was chosen for its compatibility with DFS, computational speed, and scalability. Spark is also robust by using Resilient Distributed Datasets(RDD) and if any partition is lost it can be recomputed using data lineage.

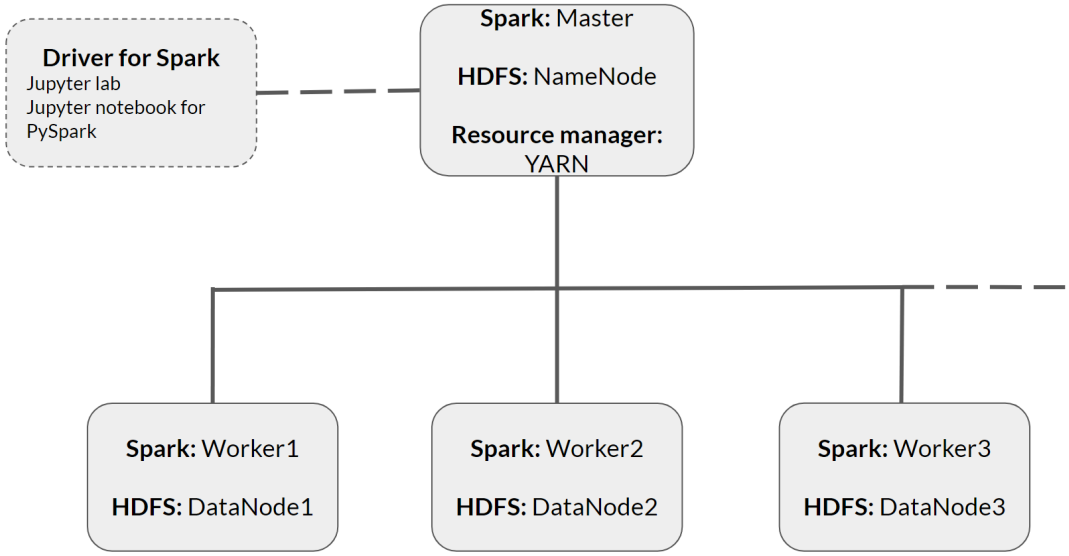

Figure 1: Our solution's architecture.

The architecture consists of a master node which hosts both Spark master and HDFS Namenode. In figure 1 the Spark workers and HDFS datanodes are both hosted on the worker nodes, this provide some advantages such as improved task execution as the data is stored together with Spark eliminating the need to transfer the data over the network plus a simplified infrastructure. It comes with drawbacks having the two applications running on the same instance, HDFS and Spark both are resource-intensive processes and by sharing resources can impact performance negatively. Required storage and computational power might not scale together hence demand for Spark workers and Datanode is different thus making co-hosting a waste of resources.

With all things considered the chosen architecture is to keep Datanodes and workers separate as the size of the dataset is static and can be stored approximately on 3 nodes with replication factor 2. This solution allows for independent scaling of each resource and automatic scaling down or up with workers depending on computational load on the cluster. The ceiling for the proposed solution is the limit of the master node, but can easily patched with secondary master and namenode. Additionally this increases system redundancy in case of master failure. The applications are submitted by a driver node running Jupyter notebook to Spark API loading data from HDFS. An additional python script was developed with the purpose to automate launching new workernode/datanode, whilst not fully integrated the functionality of launching an instance, finding a free name and IP is fully working. The instance is planned to launch without a specialized image and install required packages, for example Hadoop, JAVA and Spark. It should also copy the Hadoop configurations from the master node and required SSH properties.

3.2 Analyzing the Dataset

For the analysis we chose to answer the questions posed in the background section using a python script. The script uses the SQL functions in the pyspark library to query and group the data. The data is then ordered based on occurrence and displayed as a table. Since the analysis itself is not the main focus of the assignment we will not display all of the results here. For the curious soul, the results can be found on github.

We believe this is an effective script to test our scalability because it reads through and handles all of the data in our dataset, creating a time consuming process.

Question: What are the most frequent occurring words of the most popular subreddits?

The notebook for this question can be found on Github under Experiment and is called `popular_subreddits_popular_words.ipynb`. For the most popular subreddits, which was found in the question above, the most popular words were extracted. The aim was to get the words which were related to the subreddit and not only the most popular English words again. Because of this the words were filtered if they matched the words from the file `most_common_words.txt` which contains the 100 most common English words. The results are presented below but can also be found on Github in the file `subreddit_words.csv`.

subreddit	5 most common words
AdviceAnimals	being, think, got, really, into
AskReddit	got, back, into, after, really
atheism	god, think, believe, being, any
funny	into, being, think, got, -
gaming	game, games, play, really, any
leagueoflegends	game, team, play, really, think
relationship_advice	really, want, feel, our, am
relationships	really, want, feel, our, am
tifu	got, back, into, after, went
trees	got, smoke, really, after, back

Table 1: Most popular subreddits and their most popular words.

3.3 Analyzing Scalability

3.3.1 Horizontal scaling

In order to test the horizontal scaling of the architechture, we tested running the data analysis with 1 through 4 workers, and timed the spark session load time as well as the analysis time. In order to do this, we used an automatic "worker launching script" written in python to boot new workers from an image source before rerunning the analysis using that setup. These were the results:

Number of Workers	Load Time	Analysis Time
1	94,5s	481,7s
2	65,1s	356,9s
3	67,6s	231,7s
4	57,0s	286,2s

Table 2: Performance metrics with varying numbers of workers

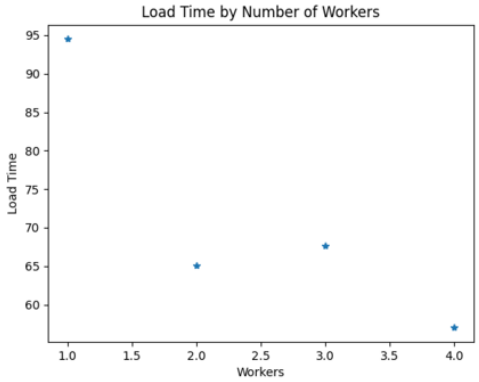

Figure 2: First figure

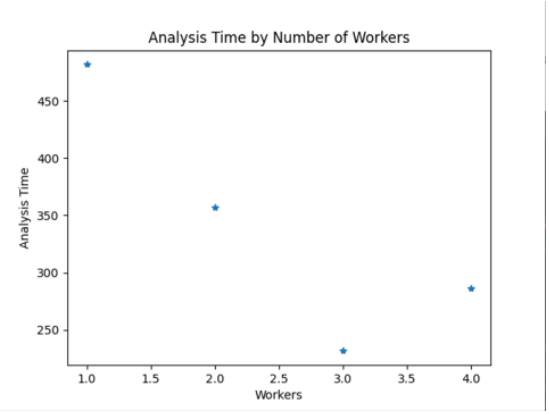

Figure 3: Second figure

3.3.2 Weak scaling

Experiments of weak scaling were done with a single workernode with the flavor `ssc.xlarge.highcpu`and limiting the allowed CPU cores in Spark submit configuration. The core-configuration that was tested can be found in table 3 and result of the data loadtime plus analysing time.

Cores	Load Time	Analyse Time
2	150.00	2538.40
4	131.76	1357.00
6	155.32	1174.25
8	123.62	651.23
10	130.80	540.76
14	103.44	467.45
16	75.10	473.90

Table 3: Sorted Cores vs. Load Time and Analyse Time

The data load time resulted approximately around the same interval as previous tests, i.e in a linear fashion. Time to analyse decreased as more cores were added, at first with a linear relationsship to the amount of cores but around 8 cores the performance boost by adding a core slowed down. To the left of the figure 4 an almost quadratic relation ship could be observed.

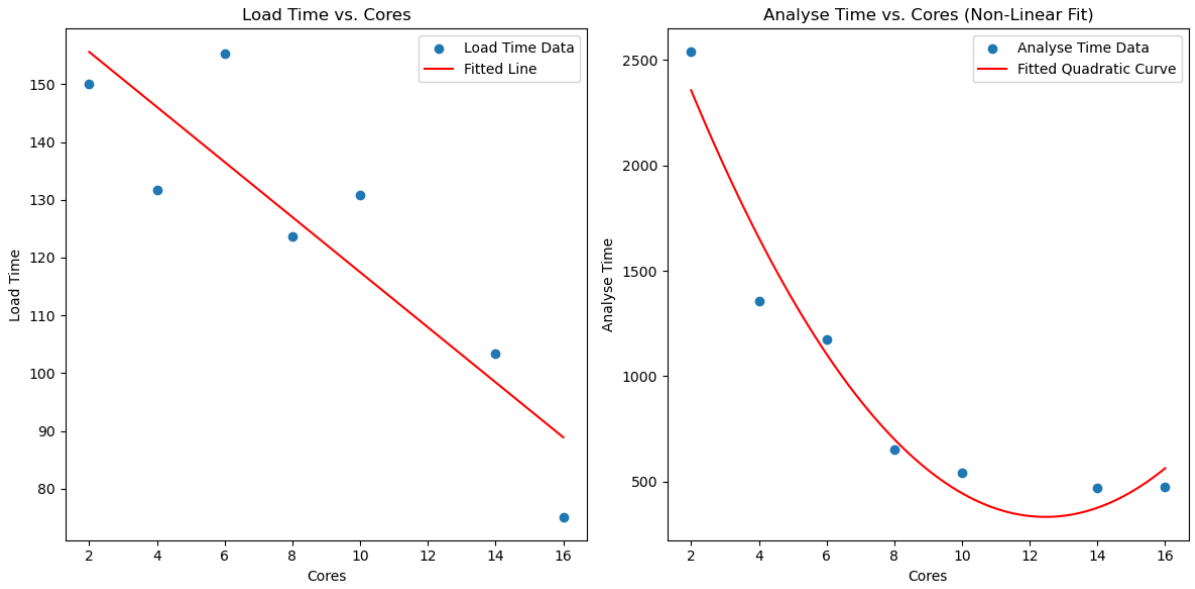

Figure 4: cores vs loadtime & cores vs analyze time

4 Discussion and Conclusion

4.1 The distributed system

After completing the project with the chosen approach, we deem it suitable for the task. It was definitely complicated to set up, and took a lot of configuration, so for a simpler solution it may not be necessary. However, it seems to be quite a good model for a larger distributed system, and is practically infinitely scalable, if one possesses the necessary resources. The key advantage to the setup is that Spark and the data nodes are both hosted on the workers, allowing for fast resource and data utilization.

In future, we would probably implement a very similar solution. Potential improvements could be higher data redundancy, potentially avoiding bottlenecks that have to do with skewed data distribution and so on.

4.2 Data analysis

The data analysis ran quickly and smoothly on the distributed system, using all available workers and cores. There was one strange effect where the spark interface would bug out a little after adding a worker, but when the initiation of the new spark session had gone through, it stabilised. The results themselves were fairly expected, with words generally common in English being common here as well. The 100 most common were filtered out, but the subreddits still shared a few words that did not seem directly related to the subreddit itself. The subreddits also showed an expected identity in their frequently used vocabulary, such as r/gaming having "game" and "games" as the two most common words.

4.3 Scaling

As previously mentioned, we implemented horizontal scaling in the form of launching additional workers automatically using a combination of a python script that accesses the Swedish Science Cloud and creates a new VM with the desired specs. That worker VM is then assigned a name based on our schema, "DE-12-worker[number]", and which number is available. It will for example not always pick the last available number, but check to see if there are any gaps. The workers are booted from an image in order to be configured sufficiently quickly.

When scaling up, the increase in efficiency seems relatively linear up to 3 workers. After that it actually gets a little slower. This may be because the overhead, skewed data distribution or idle workers. This would also explain why two workers is not twice as fast as one, even though they seem linearly dependent.

With the weak scaling, we varied the amount of CPU cores used for the computation. Here the analysis time does not seem to be entirely linearly dependent. This relationship is most likely caused by other bottlenecks in the system, for example network bandwidth to HDFS. A conclusion that can be drawn from this experiment is that for maximum usage of resources, 8 to 10 cores would be beneficial to use on multiple nodes.

5 References

References

1. Zhu Y, Haq EU, Lee LH, Tyson G, and Hui P. A Reddit Dataset for the Russo-Ukrainian Conflict in 2022. The Computer Journal 2022
2. Willaert T, Eecke PV, Soest JV, and Beul K. A tool for tracking the propagation of words on Reddit. Computational Communication Research 2021
3. Zenodo. Webis-TLDR-17 Corpus. 2017. Available from: `https://zenodo.org/records/1043504#.Wzt7PbhXryo` [Accessed on: 2024 Mar 2]
4. GeeksforGeeks. JSON. 2023. Available from: `https://www.geeksforgeeks.org/json/` [Accessed on: 2024 Mar 3]
5. Snowflake. What is JSON? Available from: `https://www.snowflake.com/guides/what-is-json/` [Accessed on: 2024 Mar 3]
6. De S and Panjwani M. A Comparative Study on Distributed File Systems. 2021. DOI: `10.1007/978-3-030-68291-0_5`. Available from: `https://www.researchgate.net/publication/351116448`

A Appendix - Initiate Spark session

```
import time
from pyspark.sql import SparkSession
from operator import add
from pyspark.sql.functions import explode, split, desc, col

#################### Setup spark session ####################

# Stop running SparkSession if it exists
if 'spark_session' in locals():
    spark_session.stop()

spark_session = SparkSession\
        .builder\
        .master("spark://192.168.2.224:7077") \
        .appName("Experiment")\
        .config("spark.dynamicAllocation.enabled", True)\
        .config("spark.dynamicAllocation.shuffleTracking.enabled",True)\
        .config("spark.shuffle.service.enabled", False)\
        .config("spark.dynamicAllocation.executorIdleTimeout","30s")\
        .config("spark.driver.port",9999)\
        .config("spark.blockManager.port",10005)\
        .getOrCreate()

# RDD API
spark_context = spark_session.sparkContext

# Start time
start_time = time.time()

# Load reddit data
reddit_df = spark_session.read.json("hdfs://192.168.2.224:9000/user/ubuntu/reddit/corpus-webis-tldr-1

reddit_df.printSchema()

# End time
end_time = time.time()

# Calculate duration
duration_sec = end_time - start_time
duration_min = duration_sec / 60

# Print duration
print("Time to load the dataset (seconds):", duration_sec, "seconds")
print("Time to load the dataset (minutes):", duration_min, "minutes")
```

B Appendix - Analysis script

```
# Start time
start_time = time.time()

#################### Most popular subreddits ####################

# Group by subreddit and count occurrences
popular_subreddits = reddit_df.groupBy("subreddit").count()

# Sort by count in descending order to find the most popular subreddits
popular_subreddits = popular_subreddits.orderBy(desc("count"))

# Show the top 10 most popular subreddits
print("Top 10 of the most popular subreddits\n")
popular_subreddits.show(10)

#################### Most frequent occurring words ####################

# Tokenize the text data in the body column
words = reddit_df.select(explode(split(reddit_df.body, "\\s+")).alias("word"))

# Count occurrences of each word
word_counts = words.groupBy("word").count()

# Sort by count in descending order to find the most frequent words
most_frequent_words = word_counts.orderBy(desc("count"))

# Show the top 10 most frequent words
print("Top 10 of the most frequent occurring words:\n")
most_frequent_words.show(10)

#################### Authors with the most bad words used ####################

# Tokenize the text in the 'body' column to extract individual words
words_df = reddit_df.select("id", explode(split(reddit_df.body, "\\s+")).alias("word"))

# Load the list of bad words
with open("full-list-of-bad-words_text-file_2022_05_05.txt", "r") as file:
    lines = [line.strip() for line in file.readlines()]
    bad_words = lines

# Filter words to select only bad words
bad_words_df = words_df.filter(words_df.word.isin(bad_words))

# Group by bad words and count occurrences
bad_word_counts_df = bad_words_df.groupBy("word").count()

# Sort by count of bad words in descending order
```

```
48 sorted_bad_word_counts_df = bad_word_counts_df.orderBy(col("count").desc())
49
50 # Show the top 10 most used bad words
51 sorted_bad_word_counts_df.show(10)
52
53 # Group by author and count bad words
54 bad_words_count_df = bad_words_df.join(reddit_df, "id").groupBy("author").count()
55
56 # Sort by count of bad words in descending order
57 sorted_bad_words_count_df = bad_words_count_df.orderBy(col("count").desc())
58
59 # Show the top 10 authors with the most bad words
60 print("Top 10 authors with the most bad words:\n")
61 sorted_bad_words_count_df.show(10)
62
63 ##################### Execution time measurement ####################
64
65 # End time
66 end_time = time.time()
67
68 # Calculate duration
69 duration_sec = end_time - start_time
70 duration_min = duration_sec / 60
71
72 # Print duration
73 print("Time to run the experiment (seconds):", duration_sec, "seconds")
74 print("Time to run the experiment (minutes):", duration_min, "minutes")
75
```