UPPSALA
UNIVERSITET

Profanity in Reddit posts and comments
Spring 2024

Group 15

Erik Larsson
Syed Muhammad Hasan
Georgios Tsouderos
Chamal Mihiranga Devasingha
Zezheng Zhang

March 2023

Contents

1 Abstract

This project sits at the intersection of computational linguistics, social media analytics, and data engineering, reflecting the data-driven approach required to analyze user-generated content on a large scale. Moreover, the present project offers insights into how profanity is woven into the fabric of Reddit conversations, contributing to a deeper understanding of online communities.

To face the scalability and efficiency challenges when processing large datasets, the project employs a suitable architecture setting utilizing Docker Swarm for container management and MongoDB for data storage, with Apache Spark handling the distributed data processing and Jupyter Notebooks implementing the computing experiments. This design ensures the system's competence to dynamically allocate computing resources, manage workload distribution, and maintain high availability and scalability.

2 Access to Code Repository

The Code for these experiments can be found here, git clone using this URL:

`https://hasansyed101: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@github.com/hasansyed101/DE_Project_15.git`

3 Background

Profanity is something that is seen as wrong in day-to-day life and in a professional setting. However, during our daily lives, there is a tendency to hear profanity much more in certain settings, such as at gaming communities. This is just from our own experience and it would be interesting to see if this is actually correct. In order to see how profanity is used in different communities, Reddit is a great place to research. As of February 2024 Reddit is the 8th most visited website in the world and the third most visited in the United States [6]. With so much traffic and considering that Reddit consists of subreddits too, which is basically its own community, analyzing these can give us a great insight into the amount of profanity that is used in the different communities.

Scraping Reddit has been used by companies and researchers to gain insights into products or trends. With the rise of cloud computing, it has been possible to analyze more and more data which led to more research based on posts on Reddit. For example during Covid-19, researchers used natural language processing to examine mental health on Reddit. The researchers noticed that there weren't enough mental health support groups and that there was an increase in

health anxiety [1]. Sentiment analysis has also been used to see if the Covid vaccine was seen as positive or negative [3]. The above are just two of the examples but of course, it can be used for much more, such as seeing company trends or how people perceive a certain product, which can be invaluable for companies [7].

To avoid scraping Reddit and pre-process all the data, the WebWebis-TLDR-17 Corpus is used. This dataset contains over 3.8 million posts already preprocessed and stored in JSON *(JavaScript Object Notation)* format. The WebWebis-TLDR-17 Corpus was made to fill a gap in existing English corpus, available for training and evaluating single document summarizing networks [10]. The existing corpus, before the WeBWis-TLDR-17, was mainly based on news articles. The makers of the dataset argued that the lack of genre diversity could be a hindrance to getting good deep learning-based summarizing [10]. The reason the dataset contains Reddit posts is because of the way it is structured. Users make posts in different subreddits, that are related to a certain topic, which greatly helps with diversifying the corpus. Another reason scraping Reddit was used is due to the trend of too long did not read (TLDR) which basically is a summarising of the author's text, made from the author. Another difference with this corpus is that it is not made by journalists and probably not proofread, which means the summarizing models have another challenge since the text is informal.

With the release of the Corpus the author also made a TLDR challenge to encourage research. The task in short was to provide software that given a text, generates a summary of it [9]. With this competition in mind, different research groups utilized the corpus. One example is the paper Task Proposal - The TL;DR challenge where the authors conclude that models less abstract, perform better when examining the summarizing performance of the models[2].

3.1 System Architecture

For BigData pipelines, Data Management is crucial and normally MongoDB or Hadoop's HDFS (Hadoop's Distributed File System) are used due to their robustness and scalability. MongoDb is a NoSQL database meanwhile HDFS is a file management system in the Hadoop ecosystem. Both allow for horizontal scaling over multiple machines which is necessary for analyzing the data in the corpus [8]. Both have some pros and cons. MongoDB provides fault tolerance, is simple to use and because of its flexible schema, it is easy to evolve the database as a system grows [8][4]. The cons for MongoDB is that it can not use joins, which means it needs to be manually coded which can reduce performance. Pros for HDFS are that it protects against hardware failure by redirecting jobs to different nodes and it's also easy to add nodes [8]. Deciding between those two came to how easy they were to set up. MongoDB seemed easier to set up and the documentation was easier to read.

To process the data Spark is used as the engine. Spark was an obvious choice given the magnitude of our dataset, given its in-memory processing and ability

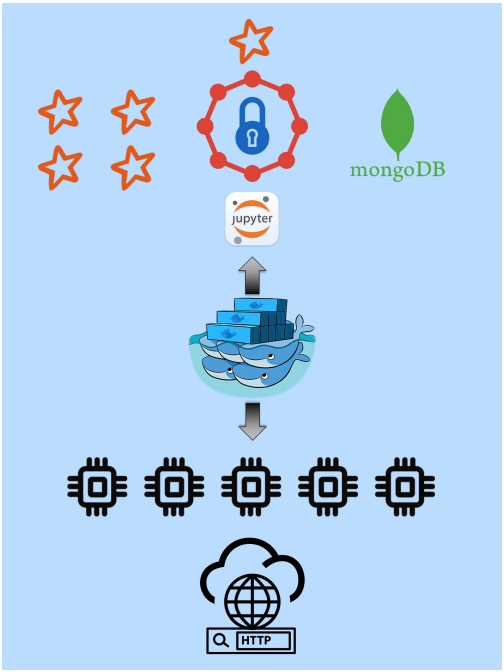

Figure 1: MultiNode Cluster Architecture

to distribute data over a cluster of devices. We also played around with the fault-tolerant RDD (Resilient Distributed Dataset), which is Spark's primary data structure. Moreover, Spark provides an extensive array of tools that facilitate the essential analysis activities we intended to carry out on the Reddit comments dataset. An additional tool used is PySpark, which is a python API for spark that allows us to utilize the python environment [5].

Finally, our interactive tool for experimenting is Jupyter Notebooks. With notebooks, we have an easy-to-use web-based interface to quickly build Spark code, examine data, and iteratively explore outcomes.

To make it easier to set up every component, Docker was employed in our project to manage the distributed architecture. Docker's containerization makes deployment easier and guarantees consistent environments for all workers. Every service such as MongoDb, Spark, Jupyter Notebook is launched as a container. These containers are fast to launch and set up which makes it easy to attach more machines to the cluster. We also made images of Spark Master and Worker which are hosted on Dockerhub for seamless access on all nodes. We defined separate services in a Docker Compose for the Jupyter Notebook environment, the MongoDB database and client, the Spark Master, and the Spark Workers, favoring in that way maintainability and organization. Furthermore, we created

Spark images that were specifically suited to the dependencies of our project using customized Dockerfiles, setting specific CPU and Memory allocations, in order to optimize our cluster performance. The Volumes mapping lets our notebooks access project code and provide data persistence for MongoDB. In that way, the Docker configuration offers a strong basis for our data engineering project's portability, reproducibility, and orchestration.

We also chose Docker Swarm since it makes managing and deploying our distributed Spark cluster easier. It manages the difficulties of load balancing data processing jobs and distributing containers among worker nodes. More crucially, it enables the smooth addition and/or removal of worker nodes as well as rescheduling tasks automatically in case of node failures.

Orchestrating the aforementioned tools was handled by a docker-compose file, which is a YAML-based configuration file, enabling us to define what images to launch as containers, in what quantity, their sequence of starting (in case of dependency), what network to use for the limited access to the applications' ports and what resource (mainly CPU and Memory) limits to set for each container.

We create a local network using the docker-compose in which all the containers can communicate with each other but only exposed the Jupyter Notebook, Spark Master UI and the MongoDB Client. The Spark Master UI allowed us to track the applications that were running, the resources they were consuming, and so on. The MongoDB Client allowed us to verify what data was being loaded and how our operations were interacting with the Database, in terms of Reads/second, etc. Finally, the Jupyter Notebook would be used to write code and communicate with Spark Master to run our operations.

The docker-compose file was being managed by Docker Swarm to easily scale the instances of any application, in this case, the spark worker nodes. In order for us to scale the applications it was important to have a multi-node setup (node cluster). So we decided to have 5 nodes each having **32 vCPUs** and **16GB Memory**. All of these nodes had a volume mounted with them that had the Jupyter Notebook files and the MongoDB data, which allowed for the seamless deployment of applications regardless of the node on which they existed. If any file in the Jupyter Notebook would change, we would just push our latest commit to our upstream source control and then pull it on all the other nodes so that we always had the latest code in all nodes.

Since our choice of Database was motivated by two main factors, management of the data storage tool and the format of the data that we had, it was quite fast and simple to load all the data to the MongoDB using a one-time container with a script to dump all the data from the JSON file into the Database. Since the Reddit comments data are semi-structured, JSON seems to be the feasible option as it would allow adding data of, almost, any format.

To process this data in PySpark, we chose its default, fault-tolerant, and robust data structure known as RDD (Resilient Distributed Dataset). This allowed us to not worry too much regarding how and where our data would be during processing. We also do not have to worry about any data losses especially when working with such a large dataset.

4 Evaluating performance

To evaluate the scalability of the system we evaluated its performance, testing the strong scalability by running the program in different configurations to see if it is scalability. We have decided to both test strong scalability and weak scalability.

4.1 Strong Scalability Experiment

To test the strong scalability, we ran the experiment on the whole dataset and fixed the number of cores per worker. We evaluated the computation time as we increased the number of workers. At Table 1 below the different tests and configurations are shown.

Table 1: Strong Scalability Tests Result

NumCores/Worker	**NumWorkers**
5	16
5	8
5	4
5	2
5	1

4.2 Weak Scalability Experiment

To evaluate the performance of the system, we also evaluated the weak scalability by running the program in different configurations to see if it is possible to scale. We have decided to restrict the test to the number of cores, number of workers and size of data. For all the trials the size of the data will be increased with the increase in the number of workers. The different tests are shown in table 2

Table 2: Weak Scalability Tests Result

NumCores/Worker	NumWorkers	Size of Data
1	16	160K
2	8	80K
4	4	40K
8	2	20K
16	1	10K

5 Results

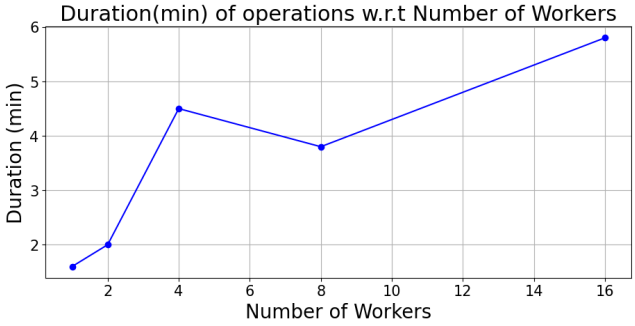

Figure 2: Strong Scalability Analysis

Table 3: Strong Scalability Tests Result

NumCores/Worker	NumWorkers	Durations(min)
5	16	5.8
5	8	3.8
5	4	4.5
5	2	2
5	1	1.6

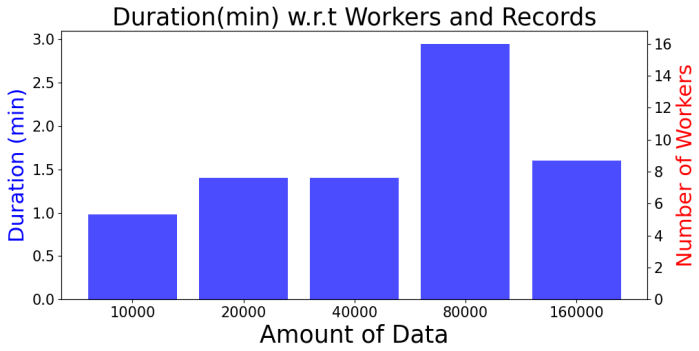

Figure 3: Weak Scalability Analysis

Table 4: Weak Scalability Tests Result

NumCores/Worker	NumWorkers	Size of Data	Durations(min)
1	16	160K	1.6
2	8	80K	2.95
4	4	40K	1.4
8	2	20K	1.4
16	1	10K	0.983

The picture below shows a sample of the output data and shows that censored words are replaced by 4 asterisks. The scalability results are without summarizing because the time complexity was shown to be too high.

```
censored_tupple.take(5)

[15]: [('funny',
   'Art is about the hardest thing to categorize in terms of good and bad. To consider one work or artist as d
ominate over another comes down to personal opinion. Sure some things maybe blatantly better than other work
s, but it ultimately lies with the individual. I personally enjoy the work of "street artists" (using quotati
ons not to be sarcastic, but mainly because this is in a different category than graffiti and since my backgr
ound is not in art I don\'t know what the "proper" term is , if there is one), but I do see where you are com
ing from. CLET tends to use the same images continuously (to a point where one could say "Is this it?") as do
most street artists (I do think this term is thrown around a lot more than it should be, I agree with you the
re) and it can be annoying. \n tl;dr: Personal opinions \'n **** \n'),
```

Figure 4: One sample of output after running the program

Summarizing result on a small sample of the dataset

```
[8]:
censored_tupple = subset_tuple.map(lambda x: (x[0],profanity.censor(x[1])))
count_of_stars = censored_tupple.map(lambda x: (x[0], x[1].count("****")))

[6]:
# count_of_stars_tuple = count_of_stars.map(lambda x: (x[0], x[2]))

# reduced_count = count_of_stars.reduceByKey(add).sortBy(lambda x: x[1], ascending=False)

[9]:
count_of_stars.take(5)
# censored_tupple.take(5)

[9]:
[('funny', 1),
 ('gamingpc', 0),
 ('RedditLaqueristas', 0),
 ('Diablo', 0),
 ('apple', 0)]
```

Figure 5: Count of censored posts in specific subreddits

6 Discussion

6.1 Strong Scalability

The Strong Scalability Experiment clearly shows that increasing the number of spark workers is not always the solution to gain speedup. This could be due to various factors one of which could be that the operation(s) we are using in our experiments may not be so easy to parallelize. The spark's internal working to distribute and manage the workers could also add extra delay in processing even before the actual operation is initiated. Also, important to keep in mind is the resource availability for the executors, where the operations might be limited due to network bandwidth or CPU cycles.

Another important factor that arises based on operations such as Reduction or Sorting, where the data needs to be sent between the workers, can add extra overhead as the number of workers grows since moving data back and forth could prove to be more resource-intensive. However, in our experiments, we could not make use of operations involving shuffling of the data due to reasons discussed in 6.3.

6.2 Weak Scalability

We can not claim that our experiments hold for weak scalability but the figure 3 shows that there is a slight tendency for weak scalability if one takes into account how to efficiently run Spark. But, ignoring the outlier at the *80000* records, we can safely project, that the weak scalability will also not hold as the number of workers increases regardless of the size of the data because then the overhead of Spark managing the Workers would come into play a lot more.

6.3 Caveats

Although we have shared the results of our experiments, it would be worth noting the issues we ran into while conducting our experiments.

In the beginning, we decided to both filter profanity from the subreddit posts/-comments and show the top five, or so, subreddits that had the highest count of bad words. The count operation required the use of `reduceByKey` function and showing the top subreddits needed to `sortBy` them after the reduction. It may seem from the architecture that our infrastructure was well-equipped, with so many cores and lots of memory, however, our cluster was not able to handle the reduction and sorting. Some of the things we tried to implement to resolve the above issues were:

- Reduce the number of records that we were processing
- Reduce the number of executors per worker so that fewer divisions and shuffling of data were made
- Limit the number of cores per executor, in case the executor was trying to consume too much CPU
- Reduce the workers to 5 (the total nodes of our cluster) with each worker having 16 vCPUs,
- Read directly from the JSON file in case MongoDB was the bottleneck
- Use `DataFrame` for operations rather than RDD,
- We also tried a small experiment on the last day before our submission to try Reduction and Sorting on just 800 records with one worker node with 5 cores and 5GB of Memory, but our experiment never returned

Figure 6: failed experiment

It seems like no matter our efforts, `reduceByKey` and `sortBy` would always cause a crash. We even tried leaving the operations to run overnight, but the operation crashed sometime around *5.5 hours*.

7 Conclusion

The system shows no sign of scalability when trying to summarize total profanity by subreddit. Even with very small sets of data, the time complexity is very high which shows signs of issues in the code. The function that seems to make the system slow down is the function `reduceByKey` and as of now we have no solution to it. However, when making the calculations simpler and just censoring profanity, we do get some interesting results. There are signs of having weak scalability, however the system does not improve performance as we add cores with a fixed data size. The reason for this is probably that the overhead increases and the processing is simple enough for our program to be able to work efficiently on fewer nodes.

References

[1] Dr Nick Adams. "'Scraping' Reddit posts for academic research? Addressing some blurred lines of consent in growing internet-based research trend during the time of COVID-19." In: Accessed: 2024-03-13. International journal of social research methodology, Feb. 2024, pp. 59–63. DOI: doi.org/10.1080/13645579.2022.2111816. URL: https://doi.org/10.1080/13645579.2022.2111816.

[2] Sebastian Gehrmann, Zachary Ziegler, and Alexander Rush. "Generating Abstractive Summaries with Finetuned Language Models". In: *Proceedings of the 12th International Conference on Natural Language Generation*. Ed. by Kees van Deemter, Chenghua Lin, and Hiroya Takamura. Accessed: 2024-03-13. Tokyo, Japan: Association for Computational Linguistics, Oct. 2019, pp. 516–522. DOI: 10.18653/v1/W19-8665. URL: https://aclanthology.org/W19-8665.

[3] Chad A. Melton et al. "Public sentiment analysis and topic modeling regarding COVID-19 vaccines on the Reddit social media platform: A call to action for strengthening vaccine confidence". In: *Journal of Infection and Public Health* 14.10 (2021). Accesses: 2024-03-13, pp. 1505–1512. ISSN: 1876-0341. DOI: https://doi.org/10.1016/j.jiph.2021.08.010. URL: https://www.sciencedirect.com/science/article/pii/S1876034121002288.

[4] MongoDB. *Hadoop or MongoDB: What is the Difference?* https://www.mongodb.com/compare/hadoop-vs-mongodb. Accessed: 2024-03-13.

[5] PySpark. *PySpark Overview¶*. https://spark.apache.org/docs/latest/api/python/index.html. Accessed: 2024-03-13.

[6] SEMRUSH. *Reddit.com*. https://www.semrush.com/website/reddit.com/overview/. Accessed: 2024-03-13.

[7] SEMRUSH. *Reddit.com*. https://scrapingrobot.com/blog/web-scraping-reddit/. Accessed: 2024-03-13.

[8] Simlilearn. *Hadoop Vs. MongoDB: What Should You Use for Big Data?* https://www.simplilearn.com/hadoop-vs-mongodb-article. Accessed: 2024-03-13. 2023.

[9] Shahbaz Syed et al. "Task Proposal: The TL;DR Challenge". In: *Proceedings of the 11th International Conference on Natural Language Generation*. Ed. by Emiel Krahmer, Albert Gatt, and Martijn Goudbeek. Accessed: 2024-03-13. Tilburg University, The Netherlands: Association for Computational Linguistics, Nov. 2018, pp. 318–321. DOI: 10.18653/v1/W18-6538. URL: https://aclanthology.org/W18-6538.

[10] Michael Völske et al. “TL;DR: Mining Reddit to Learn Automatic Summarization”. In: *Proceedings of the Workshop on New Frontiers in Summarization*. Ed. by Lu Wang et al. Accessed: 2024-03-13. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 59–63. DOI: 10.18653/v1/W17-4508. URL: https://aclanthology.org/W17-4508.