UPPSALA
UNIVERSITET

LDSA Project

Group 17

Pei-Hsuan Lin
Ricardo Danza Madera
Reuben Sajit Rajkumar
Clara Bernedal Nordström
Uvais Karni

Background

The data was collected in interest of studying social media. With this data it is possible to see what topics were hot at that moment and with newer datasets you can also see what is trending right now which can aid in marketing, news, media etc. It can also be used to get an image of what opinions are out there and their popularity together with user interactions with others or topics. Three examples on when social media data was used for different studies are firstly, a study on the attitude of the consumers of two competing brands on twitter and facebook comments (F. Neri, C. Aliprandi, F. Capeci, M. Cuadros, T. By, 919-926) which could be used on our dataset too. Another is a text mining study over the three biggest pizza chain's image online (Wu He, Shenghua Zha, Ling Li, 464-472), which also could be used on our dataset. Lastly a study actually made on Reddit comments that studied if it would be possible to see anxiety disorder through personal narratives on Reddit (Hollingshead, Kristy & Ireland, Molly & Loveys, Kate, 69).

Data format

The data is in a JSON format which means it is semi-structured in nature. This is needed since the data collected for a comment aren't always the same but it still follows a type of order. Other alternatives to JSON are YAML, XML, CSV and more. YAML *(Yaml Ain't Markup Language)* has about the same features as JSON but instead of curly brackets as in JSON it uses indentation for nesting levels. It is therefore readable, at least that's the intent. XML *(eXtensible Markup Language)* uses angle-brackets similarly to HTML. This was more widely used before but is decreasing in popularity to JSON. It can handle more complex data than simply string-value as in JSON but this also means JavaScript can be harder to parse with. It is also less readable. CSV *(Comma Separated Values)* This is good for importing spreadsheets for example in excel. It consists of a header row and string-values pairs.

Pros of using JSON:

Speed: JSON syntax is easy to use, parse and execute.
Browser compatible: JSON is widely supported on multiple browsers which results in less effort needed to make it all browser compatible.
Server parsing: Since JSON is parsed fast in browsers the whole user experience will become faster which is important to developers.
Sharing: When data is shared between a server and a browser the shared data can only be text, which is what the JSON format is. JSON is also great for sharing all kinds of data, even things like audio and video. This comes from how JSON stores data which is in the form of an array. Using arrays makes transferring the data much easier. Another thing is that JSON can easily convert into JavaScript and back again. This makes JSON a superior format for web development and APIs.

Cons of using JSON:

Error handling: JSON unfortunately doesn't have any error handling for it's calls so when it fails, it fails silently.
Safety: The JSON response will be wrapped in a function call which the browser will execute. This makes the web application that hosts vulnerable to attacks if an untrusted browser is used. So this is something very important to keep in mind when using JSON.

Computational experiments

Our program uses Apache Spark with HDFS for the file management system. We thought that given the type of data we are dealing with, as discussed previously, these would be the most suitable tools for the reasons that will now be outlined. We will be using the term "the Job" to refer to the entire pipeline of operations performed on our data.

Tools and distributed system

The reason for using Apache Spark is because it is a platform that is designed for processing large amounts of data in a distributed manner. Spark focuses on fast stream-processing, since it mostly stores the data in-memory. However, one can also carry out batch-processing in Spark by integrating it with HDFS for file management as well as access to other HDFS functionalities.. HDFS allows Spark to not be completely dependent on RAM when processing data, instead also writing data to file and reading from files. This ability is useful given the fact that we have large files containing JSON objects stored on disk.

HDFS also provides security in the form of data replication, in case one node fails, then the information in that node will not be lost and can be recreated from the copies stored in other nodes. HDFS provides the added benefit of, with the help of YARN, scheduling tasks across available nodes in the cluster.

Given that we are using these frameworks, there is not much that can be done regarding the architecture of the solution. Spark already provides us with a master-worker framework for processing the data in the desired manner. Our job is more akin to choosing the parameters to use, for example; we have a master node (driver node) where the program is written using Jupyter, but the number of workers is determined by us.
For the scalability tests, we choose the number of worker-nodes that are going to be used, this is done external to the code that we write, that is, by linking the IP addresses

of the worker nodes to that of our master so that they are automatically detected and used. After these parameters have been decided, the amount of data to use is chosen. To make things simpler, we decided to load for example, 20GB of comments into HDFS. To choose 20% of workload, we decided to take a random sample of 20% of the rows in the data set.

Scalability and results

Pre-processing

Firstly we had to read in our data from a JSON format to a dataframe. We then dropped all the unnecessary columns and only kept “Subreddit”, “Body” and “Count”. To reduce the amount of files even more we decided to only look at the comments with at least a score of 100 or greater since we only wanted to look at the most popular topics. We then removed everything that is not letters (like periods, commas etc) and made everything into lowercase to get a better matching later on. We broke up the comments from one whole string into separate words and then used a function called StopWordsRemover which removes words like “a”, “is”, “it” etc. Finally we grouped all the rows by its corresponding subreddit, counted each word and sorted it by count in descending order.

Pre-processing results

subreddit	word	count
AskReddit	one	1675
AskReddit	like	1642
AskReddit	im	1292
AskReddit	get	1280
AskReddit	people	1134
AskReddit	time	1094
AskReddit	dont	1076
AskReddit	got	872
AskReddit	back	862
AskReddit	know	825
AskReddit	go	774
AskReddit	really	763

AskReddit	day	721
AskReddit	didnt	699
AskReddit	think	682
AskReddit	never	667
AskReddit	guy	634

Table 1: The most used words in the subreddit "AskReddit" together with the amounts of time they have been used in the most liked comments.

Scalability

For the tests, we will be testing Strong and Weak scalability using Horizontal Scalability. What this means is that processing power will be measured and implemented horizontally and not vertically. "Vertical" would imply using different instances with increasing numbers of vCPU and RAM. But what we are doing is to use more nodes of equal size. The time measurement shows how long it takes to carry out the preprocessing that was mentioned earlier.

Weak scaling tests will be carried out by maintaining a constant load per node. To achieve this, it is necessary to carry out several tests, and measure the time taken to execute the Job. A first baseline test can be carried out with one worker node, and X load. As the number of workers increases, the ratio of workers to load needs to be maintained. Implying that a bigger amount of data needs to be fed to our program. We will know that our architecture scales well if the execution time remains roughly equal, ie. constant, throughout the different trials. If the execution times decrease it would be suggestive of increased effectiveness, otherwise it would imply a bad scaling program.

For the strong scalability testing we executed the Job on a fixed data size but on an increasing cluster. For example, the baseline can be executed on only one worker node. Then increasing to two worker nodes, then four, etc. A good implementation should show that the time taken has decreased by a factor equal to the number of nodes

Scalability Test Results

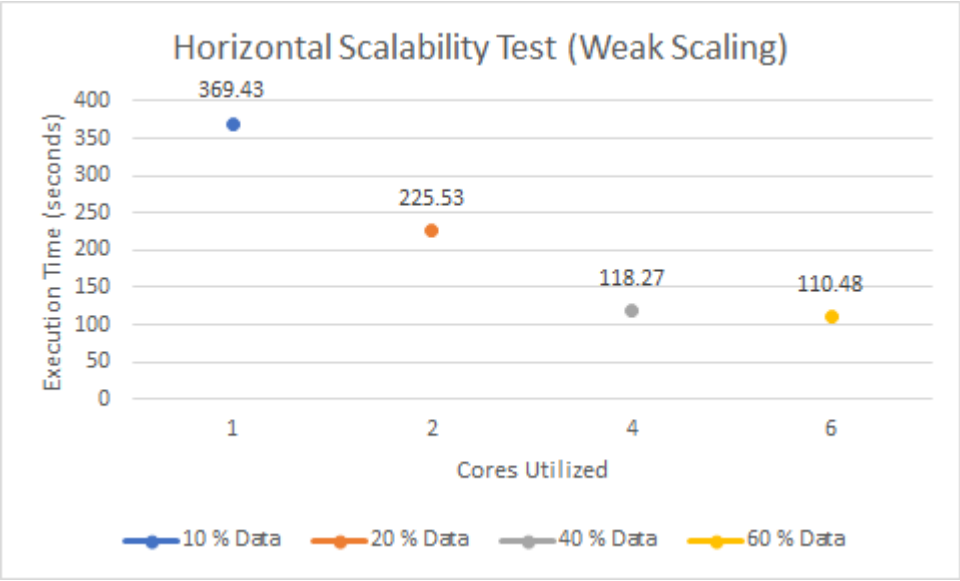

Figure 1: A plot of the Weak Scalability test on a different number of cores and amount of data.

1 core, 10% of data	2 cores, 20% of data	4 cores, 40% of data	6 cores, 60% of data
369.43	225.53	118.27	110.48

Table 2: The results of the weak scaling test.

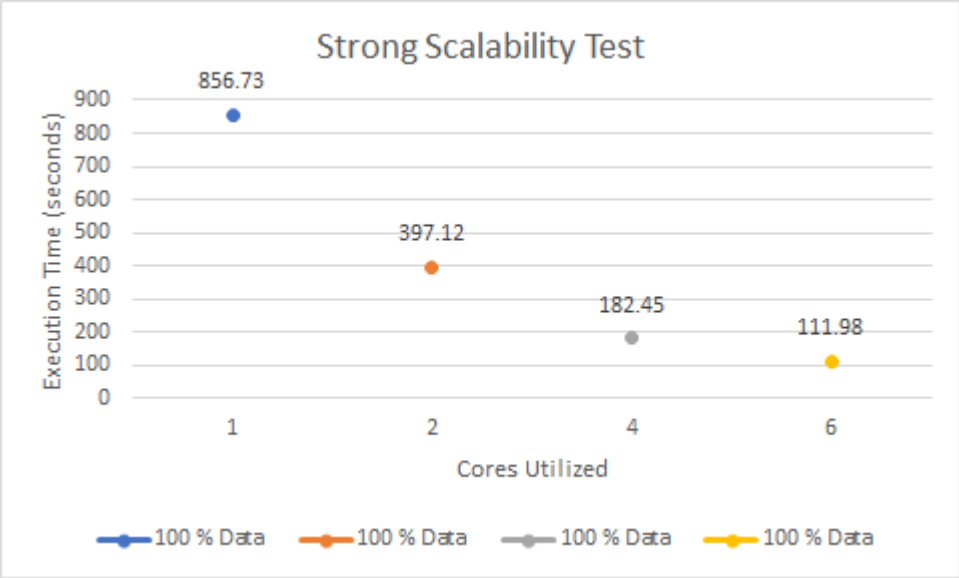

Figure 2: A plot of the strong scalability test on a different number of cores and the same amount of data.

1 cores, 100% of data	2 cores, 100% of data	4 cores, 100% of data	6 cores, 100% of data
856.73	397.12	182.45	111.98

Table 3: The result of the strong scaling test.

Discussion and conclusion

First we wanted to only look at the comments that had a controversial score but we soon noticed that extremely few comments were classified as such. One of the subsets didn't even have any at all. Because of this we changed the pre-processing to look at the comments with the highest score instead. Looking at the words from Table 1, if we had more time we could also have tried to filter away things like spelled numbers and words like "like", "im" and "get" to maybe be able to get a glimpse of what the comments are talking about. But all in all, we were able to do what we had set out to accomplish.

Weak Scalability test

The scaling part of this project worked really well and as seen in Figure 1: 4 and 6 cores, together with corresponding 40% and 60% of the data, takes around the same amount of time which is exactly what we want to see when doing this type of testing. 1 and 2 cores take more time than the others which we do not exactly know why. We assume that the data sizes in the first two experiments were relatively small and that the overhead of starting a Spark job might be a factor. Nonetheless, the plot looks good since it technically got better with more cores until it flattened out for the 3rd and 4th runs.

Strong Scalability test

The results from Figure 2 show; if looking at the rounded numbers; 1 core = 850s, 2 cores = 400s, 4 cores = 200s and 6 cores = 100s, looks like it should, since the time decreases as cores get added. It would be even better if we had tested with 8 cores too since that would have been double the amount of cores from 4 and the time then should have been halved too. So, that the slope of the curve looks like it is starting to get less steep is because it is for 6 cores and not 8. Taking this into account, we believe this test also shows that our code is scalable.

Setting up the framework; which includes setting up Spark in distributed mode, setting up HDFS to be run with Spark, as well as other technicalities like being able to run Jupyter and see it from our local browsers, all of these took considerable effort and time. It might have been possible to create more interesting experiments with other paradigms or to have created a more interesting preprocessing Job, but we felt that we were constrained by the time available to us and could therefore not do so. With the results that were obtained so far we are satisfied, as it shows that the theory matches with practical results. The only thing that seems a bit odd is the decreasing run-times for the Weak-Scaling experiments.

References

https://github.com/reddit-archive/reddit/wiki/JSON

Neri, F. Aliprandi, C. Capeci, F. Cuadros M. By, T. "Sentiment Analysis on Social Media," 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Istanbul, 2012, pp. 919-926, doi: 10.1109/ASONAM.2012.164.

He, W. Zha S. Li, L. “Social media competitive analysis and text mining: A case study in the pizza industry”, *International Journal of Information Management*, vol 33, no. 3, 2013, pp. 464-472, ISSN 0268-4012

Hollingshead, K. Ireland, M. Loveys, K. Shen JH. Rudzicz F. “CLPsych 2017 The Fourth Workshop on Computational Linguistics and Clinical Psychology-From Linguistic Signal to Clinical Reality Proceedings of the Workshop”, page 69