
BIG DATA W PRZEMYŚLE 4.0
PROJEKT
ANALIZA BAZY DANYCH HASEŁ

Autorzy:
ADRIANNA SZARUGA 247294
RAFAŁ RZEWUCKI 248926

Prowadzący: dr inż. Andrzej Gnatowski

17 stycznia 2023

1 Wstęp

Współcześnie człowiek jest w stanie wygenerować lub zebrać bardzo dużą ilość danych. Dochodzi więc często do sytuacji, gdzie nie jesteśmy w stanie przetworzyć zebranych danych na jednym procesorze w akceptowalnym czasie. Mamy wtedy doczynienia ze znanym już obecnie problemem Big Data. Polega on na znalezieniu sposobu przetworzenia lub analizy bardzo dużych ilości informacji w akceptowalnym czasie.

2 Cel projektu

Celem projektu jest zapoznanie się z narzędziami wspomagającymi analizę bardzo dużych ilości informacji, takich jak rozproszone macierze dyskowe i rozproszone przetwarzanie danych. Dodatkowym celem jest zapoznanie się ze sposobami wizualizacji danych po przeprowadzonej analizie w sposób interaktywny dla odbiorcy.

3 Problemy wynikające z przetwarzania dużych ilości danych

3.1 Pojemność dysku

Pierwszym problemem jaki napotykamy jest ograniczona pamięć dyskowa. Jako że objętość Big Data może być liczona w petabajtach, trudno jest znaleźć jeden komputer, który będzie zarządzał taką ilością pamięci. O ile fizycznie jest to możliwe do zrealizowania, jednak może pociągać za sobą spore nakłady finansowe. Dlatego w takich sytuacjach stosuje się narzędzia umożliwiające rozdzielenie danych na wiele komputerów, z których każdy przechowuje tylko część danych. Zarządzanie tym, który komputer będzie przechowywał jaką część informacji przejmują najczęściej jeden lub więcej komputerów nadrzędnych, tak aby można było zgromadzone dane ułożyć w logiczną całość.

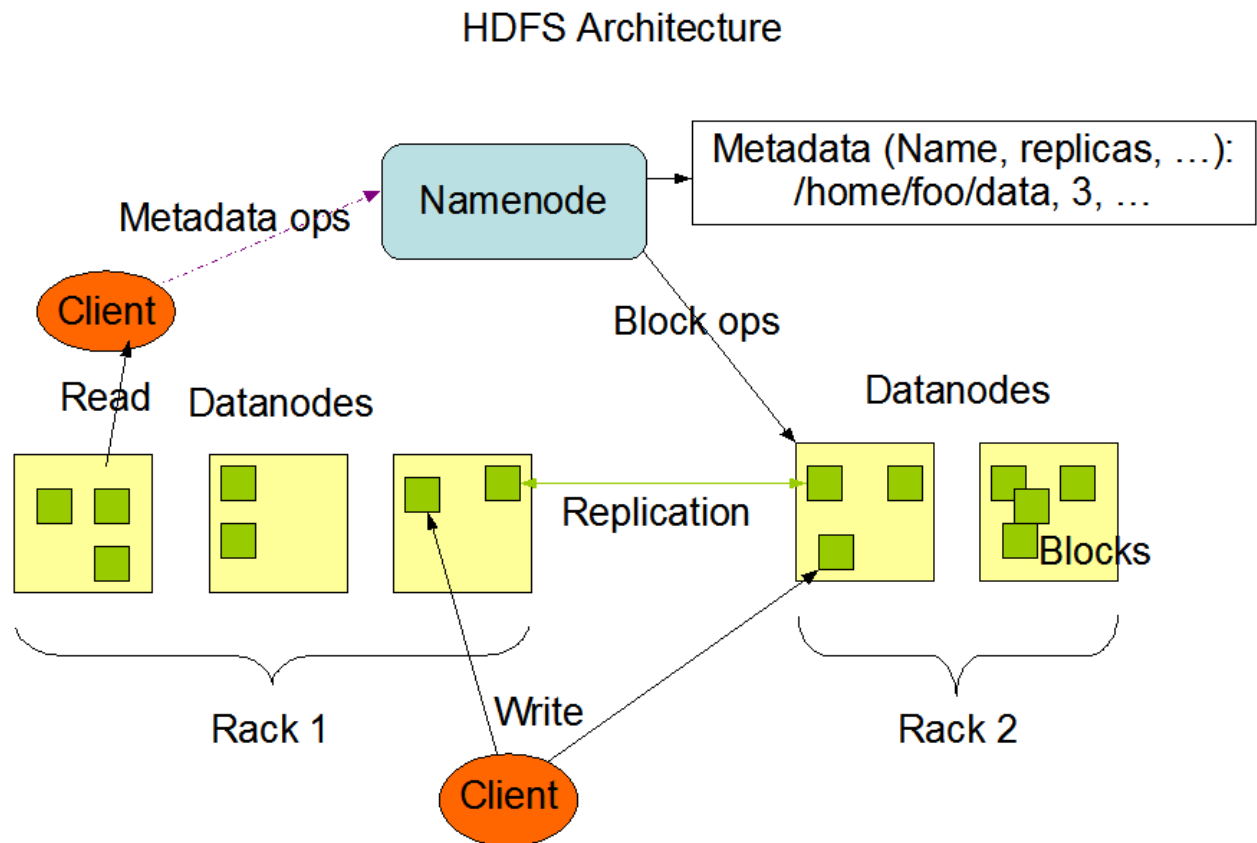
3.2 Szybkość przetwarzania

Kolejnym problemem jest sam proces przetwarzania. W obecnych czasach proces rozwoju procesorów nie nadąża za rosnącą ilością danych do przetworzenia, a co za tym idzie, firmy technologiczne na dzień dzisiejszy nie są w stanie wyprodukować procesora, który będzie w stanie przetwarzać Big Data w akceptowalnym czasie. W takich sytuacjach stosuje się rozproszone przetwarzanie, które jest możliwe dzięki zastosowaniu wcześniej wspomnianego rozproszonego przechowywania danych. Końcowo sprowadza się do tego, że dane są analizowane na tej samej maszynie, na której są przechowywane, co niweluje konieczność przesyłania danych poprzez sieć w celu przetworzenia.

4 Środowisko programistyczne

4.1 HDFS - Hadoop Distributed File System

System plików opracowany przez fundację Apache do projektu wyszukiwarki Apache Nutch web search engine jednak zauważono jego inne zastosowania, dlatego obecnie jest to osobny projekt. System ten wyróżnia się na tle innych rozproszonych systemów plików tym, że można go uruchamiać na maszynach, które nie posiadają infrastruktury przemysłowej, a co za tym idzie nie mają zbyt dużej możliwości przetwarzania danych. Dodatkowo zapewnia bardzo dużą odporność na awarię, poprzez zastosowanie mechanizmów replikacji danych pomiędzy maszynami oraz odpowiednich algorytmów ich balansowania. Schemat budowy rozproszonego systemu plików został przedstawiony na rysunku 1.



Rysunek 1: Schemat budowy rozproszonego systemu plików (źródło: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html)

4.2 Apache Hadoop

Jest to framework programistyczny umożliwiający przetwarzanie danych w rozproszonym systemie hd fs pomiędzy wieloma maszynami jednocześnie używając do tego prostych modeli programistycznych. Apeache Hadoop sam zapewnia ochronę przed awariami, dlatego może być uruchamiany również na maszynach, które mają skłonności do awarii. Potrafi on wykryć awarię i przekazać pracę do innych maszyn, tak aby uzyskać końcowy wynik.

5 Dane wejściowe

Danymi wejściowymi używanymi w tym projekcie do zapisu w rozproszonym systemie plików były bazy danych haseł wygenerowane za pomocą narzędzia pydictor <https://github.com/LandGrey/pydictor>, które generuje listę haseł o określonej strukturze. Wygenerowano dwa pliki, jeden o wielkości 6 GB, który służył jako plik testowy na którym można było testować programy analizujące dane i mieć pole do porównań oraz sprawdzić działanie rozproszonego systemu plików. Drugi plik był docelowym plikiem o objętości 100 GB, na którym zostały uruchomione docelowe analizy danych. Oba pliki zawierały wygenerowaną listę haseł składających się z małych, dużych liter oraz cyfr. Długość haseł była zmienna w zależności od pliku, jednak nie przekraczała ona 6 znaków na hasło.

6 Instalacja środowiska programistycznego

6.1 Przygotowanie maszyn

W projekcie wykorzystano dwie maszyny z zainstalowanymi systemami Windows 10. W celu uporządkowania ról maszyn w systemie rozproszonym nadano im statyczne adresy IP, które zapisano w pliku host każdej z maszyn.

```
192.168.1.4 hadoop-datanode2
192.168.1.8 hadoop-namenode localhost
```

Rysunek 2: Zawartość pliku hosts na obu maszynach

6.2 Instalacja środowiska Apache Hadoop

Pliki źródłowe można pobrać ze strony <https://hadoop.apache.org/> a sama instalacja polega na rozpakowaniu pobranego archiwum na partycji, gdzie zainstalowany jest system operacyjny, w tym przypadku był to katalog C:/hadoop.

Apache Hadoop jest programem napisanym w języku Java, dlatego należy również zainstalować środowisko programistyczne tego języka. W projekcie zostało użyte JDK w wersji 11.0, ponieważ według twórców frameworka z tą wersją nie występują problemy.

Domyślnie Apache Hadoop jest przystosowany do uruchamiania w środowisku Linux, aby pomyślnie uruchomić go w środowisku Windows należy pobrać wcześniej przygotowane pliki wykonywalne ze strony <https://github.com/steveloughran/winutils> i dodać je do katalogu **bin** w katalogu instalacyjnym Hadoop'a. Aby rozpocząć pracę z rozproszonym systemem plików, należy dokonać wstępnej konfiguracji, gdzie ustawiane są podstawowe parametry, takie jak to która maszyna będzie zarządzać systemem plików oraz to, na których maszynach będą przechowywane dane. Definiujemy tu również fizyczne położenia danych na dyskach poszczególnych maszynach oraz to jak dane będą pomiędzy nimi replikowane. Całość konfiguracji to edytowanie plików konfiguracyjnych znajdujących się w katalogu instalacyjnym Hadoop'a w podkatalogu **etc**. Całość konfiguracji wraz z objaśnieniami prezentuje artykuł <https://towardsdatascience.com/installing-hadoop-3-1-0-multi-node-cluster-on-ubuntu-16-04-step-by-step-8d1954b31505> autora *Hadi Fadlallah*. Mimo, że odnosi się on do środowiska Linux, to kroki są analogiczne w środowisku Windows.

Wartym odnotowania jest, że Apache Hadoop wymaga systemu operacyjnego Windows w wersji językowej angielskiej, gdyż niektóre z funkcji systemu różnią się pomiędzy wersjami polską i angielską.

6.3 Przygotowywanie systemu plików

Aby rozpocząć pracę z rozproszonym systemem plików należy go wstępnie przygotować. Proces ten nazywa się formatowaniem, a sprowadza się do wykonania kilku komend w terminalu systemu. Ważnym odnotowania jest fakt, że wszystkie komendy działają poprawnie tylko jeśli zostaną uruchomione z prawami administratora. W przeciwnym wypadku możliwe jest otrzymanie błędu braku uprawnień do wykonania poszczególnych poleceń. Aby przygotować maszynę zarządzającą systemem plików do pracy należy przygotować wymagane struktury katalogów tej maszyny. Do wykonania tego zadania służy polecenie:

```
hdfs namenode -format
```

, którego wykonanie trwa zwykle od kilku do kilkunastu sekund.

Kolejnym krokiem jest przygotowanie maszyn przechowujących dane, a więc na obu maszynach należy wykonać polecenie

```
hdfs datanode -format
```

, które przygotowuje wszystkie wymagane struktury plików na dysku i połączy maszynę przechowującą dane z maszyną zarządzającą.

6.4 Uruchomienie Apache Hadoop

Po skonfigurowaniu i sformatowaniu systemu plików, oraz rozwiązaniu wstępnych problemów można było przystąpić do uruchomienia programu. Twórcy Apache Hadoop w katalogu **sbin** przygotowali skrypty powłoki pozwalające na uruchomienie wszystkich usług za pomocą tylko jednej komendy.

Na maszynie zarządzającej należy uruchomić wszystkie wymagane usługi a więc

- namenode - zarządza przechowywaniem plików na wielu maszynach
- datanode - zarządza plikami przechowywanymi na maszynach
- node manager - zarządza dostępnymi maszynami
- resource manager - zarządza dostępnymi zasobami obliczeniowymi na maszynach

Do uruchomienia wszystkich wymienionych powyżej usług służy skrypt:

```
start-all
```

Po uruchomieniu wszystkich usług otrzymano 4 nowe okna terminali wyświetlające statusy poszczególnych usług, zrzut ekranu został przedstawiony na rysunku 3.

The screenshot displays four terminal windows from the 'Apache Hadoop Distribution' directory. The first window shows the 'namenode' startup logs, including the 'fsck' command and the 'start-all' command. The second window shows the 'datanode' startup logs, including the 'start-all' command. The third window shows the 'resource manager' startup logs, including the 'start-all' command. The fourth window shows the 'namenode' startup logs, including the 'start-all' command. The logs show the progress of the services starting up, including the 'fsck' command and the 'start-all' command.

Rysunek 3: Widok ekranu po uruchomieniu wszystkich usług na maszynie zarządzającej

Na maszynach, które odpowiadają za przechowywanie danych wystarczy uruchomić usługi pozwalające na zarządzanie danymi, a więc tylko usługę datanode. Służy do tego skrypt

```
start-dfs
```

, którego efektem jest uruchomienie tylko jednego nowego terminala wyświetlającego status działającej usługi. Po wykonaniu powyższych kroków można było już w pełni korzystać z możliwości jakie udostępnia Apache Hadoop.

7 Analiza danych

7.1 Przetwarzanie równoległe

Ważnym konceptem, do którego został przygotowany Apache Hadoop jest możliwość przetwarzania równoległego na wszystkich dostępnych maszynach. Każda maszyna ma możliwość analizy danych, które przechowuje na swoim dysku, a więc wystarczyło tylko zarządzić tak tą pracę, aby uzyskać oczekiwane efekty.

Cała koncepcja oparta jest na platformie MapReduce stworzonej przez firmę Google. Podczas analizy danych są one przetwarzane w dwóch krokach:

1. Krok **'map'** - mapuje dane, tak aby zredukować ich rozmiar i wyciągnąć określone dane. Np. wyszukuje ilość wielkich liter w ciągu znaków i zwraca ich liczbę do głównego programu.
2. Krok **'reduce'** - działa w głównym programie, otrzymuje informacje z kroku map i łączy je w jeden wynik

Jedną z głównych zalet podejścia MapReduce jest to, że zakładamy niezależność pojedynczej operacji map od pozostałych, a więc można rozdzielić je na osobne serwery

7.2 Przesłanie danych do rozproszonego systemu plików

Przed przystąpieniem do analizy danych, należy je najpierw umieścić w rozproszonym systemie plików. Jako, że dfs działa analogicznie jak standardowy, można w nim również tworzyć struktury katalogów. Dla utrzymania porządku utworzono katalog *scripts* i w nim podkatalogi *input-data* i *output-data*. Przygotowane wcześniej pliki o objętościach 6 GB i 100 GB umieszczono na jednej z maszyn i komendą

```
hadoop fs -put {źródło} {ścieżka docelowa na dfs}
```

wysłano do rozproszonego systemu plików. Sam proces przesyłania trwał kilkadziesiąt minut, ponieważ program wysyłał jednocześnie dane na obie maszyny. Poprawność wysłania plików można sprawdzić korzystając z komendy

```
hadoop fs -ls {ścieżka docelowa na dfs}
```

umożliwiającej przeglądanie katalogów i ich zawartości w systemie. Wynik został przedstawiony na rysunku 4.

```
C:\hadoop\sbin>hadoop fs -ls /
Found 3 items
drwxr-xr-x - rrzewucki supergroup      0 2022-12-07 07:34 /mr-history
drwxr-xr-x - rrzewucki supergroup      0 2022-12-07 07:33 /scripts
drwx----- - rrzewucki supergroup      0 2022-12-07 07:34 /tmp

C:\hadoop\sbin>hadoop fs -ls /scripts
Found 2 items
drwxr-xr-x - rrzewucki supergroup      0 2022-12-06 23:25 /scripts/input-data
drwxr-xr-x - rrzewucki supergroup      0 2022-12-15 10:37 /scripts/output-data

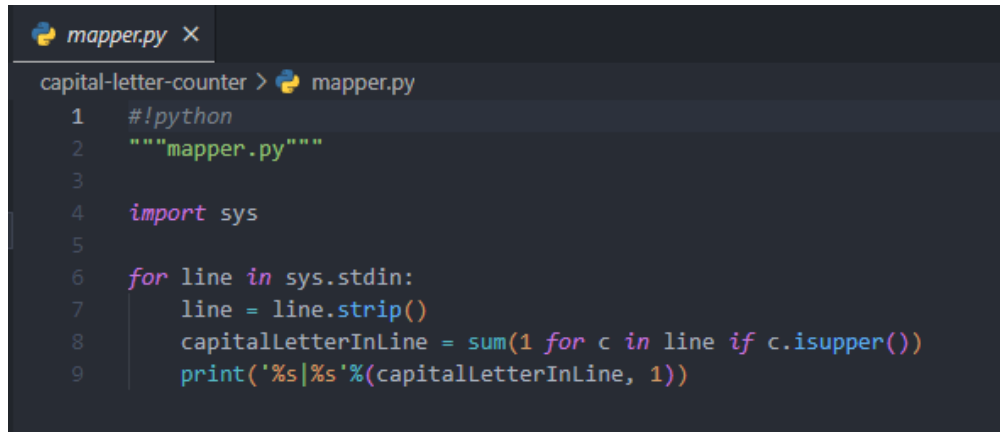
C:\hadoop\sbin>hadoop fs -ls /scripts/input-data
Found 2 items
-rw-r--r--  2 rrzewucki supergroup 107374182400 2022-12-06 23:25 /scripts/input-data/input-100G.txt
-rw-r--r--  2 rrzewucki supergroup  6502779478 2022-12-06 20:15 /scripts/input-data/input-6G.txt

C:\hadoop\sbin>
```

Rysunek 4: Podgląd zawartości katalogów w systemie

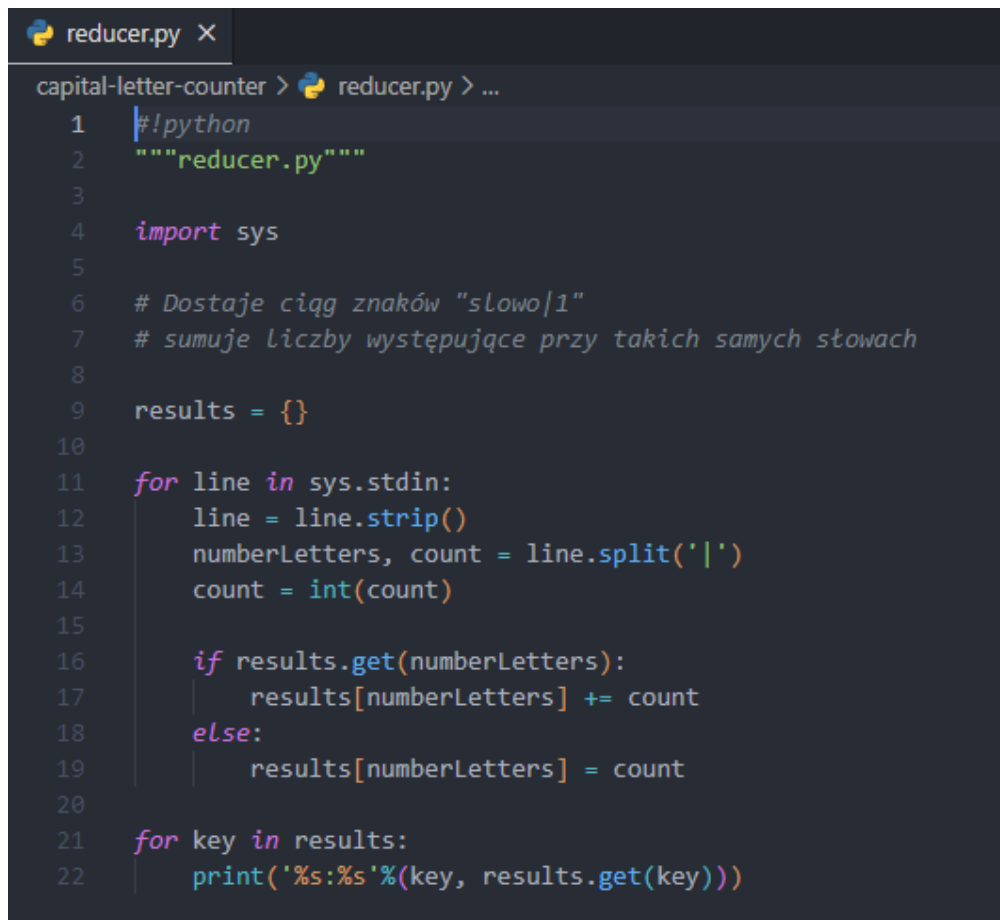
7.3 Przygotowanie programów mapper.py reducer.py do analizy danych

W celu analizy danych z wykorzystaniem modelu MapReduce opisanego w rozdziale 7.1 przygotowano zestaw skryptów w języku python analizujących dane przesyłane strumieniowo. Takie podejście pozwalało również na testowanie programów lokalnie. Jako przykład została przedstawiona analiza ilości wielkich liter w hasłach. Wszystkie analizy dostępne są pod adresem <https://github.com/rafal166/PWr2sem2BigData>.



```
capital-letter-counter > mapper.py
1  #!python
2  """mapper.py"""
3
4  import sys
5
6  for line in sys.stdin:
7      line = line.strip()
8      capitalLetterInLine = sum(1 for c in line if c.isupper())
9      print('%s|%s'%(capitalLetterInLine, 1))
```

Rysunek 5: Skrypt mapujący dane w pierwszym kroku przetwarzania



```
capital-letter-counter > reducer.py > ...
1  #!python
2  """reducer.py"""
3
4  import sys
5
6  # Dostaje ciąg znaków "słowo|1"
7  # sumuje liczby występujące przy takich samych słowach
8
9  results = {}
10
11 for line in sys.stdin:
12     line = line.strip()
13     numberLetters, count = line.split('|')
14     count = int(count)
15
16     if results.get(numberLetters):
17         results[numberLetters] += count
18     else:
19         results[numberLetters] = count
20
21 for key in results:
22     print('%s:%s'%(key, results.get(key)))
```

Rysunek 6: Skrypt zliczający dane z pierwszego kroku przetwarzania

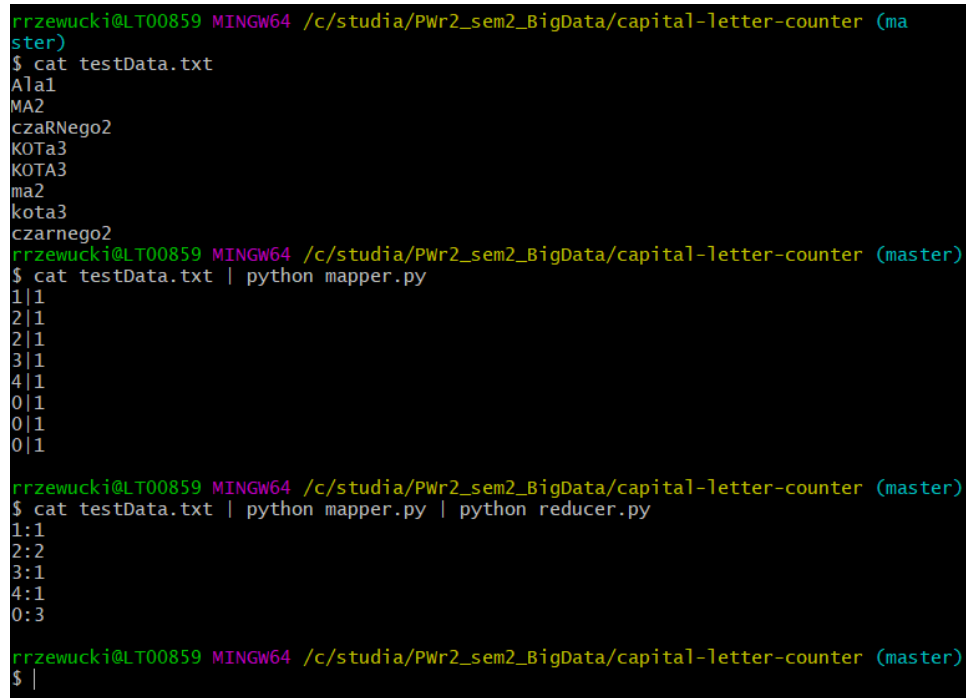
Plik mapper.py został przedstawiony na rysunku 15. Jego wyjściem po przekazaniu do niego danych w formie strumienia jest ciąg

```
2|1
4|1
5|1
```

gdzie

```
{liczba_duzych_liter_w_haśle}|{stała_liczbowa}
```

stała_liczbowa jest konieczna aby móc zastosować standardowy plik reducer przedstawiony na rysunku 6. Aby przetestować działanie skryptu można uruchomić go lokalnie przekazując odpowiedni strumień wejściowy jak na rysunku 7.



```
rrzewucki@LT00859 MINGW64 /c/studia/Pwr2_sem2_BigData/capital-letter-counter (master)
$ cat testData.txt
Ala1
MA2
czaRNego2
K0Ta3
K0TA3
ma2
kota3
czarnego2
rrzewucki@LT00859 MINGW64 /c/studia/Pwr2_sem2_BigData/capital-letter-counter (master)
$ cat testData.txt | python mapper.py
1|1
2|1
2|1
3|1
4|1
0|1
0|1
0|1

rrzewucki@LT00859 MINGW64 /c/studia/Pwr2_sem2_BigData/capital-letter-counter (master)
$ cat testData.txt | python mapper.py | python reducer.py
1:1
2:2
3:1
4:1
0:3

rrzewucki@LT00859 MINGW64 /c/studia/Pwr2_sem2_BigData/capital-letter-counter (master)
$
```

Rysunek 7: Lokalna symulacja przetwarzania strumieniowego

7.4 Uruchomienie analiz

Uruchomienie przetwarzania danych za pomocą wcześniej przygotowanych skryptów możliwe jest dzięki przygotowanego przez twórców Apache Hadoop programowi, którego jedynym zadaniem jest odczytanie wskazanego pliku znajdującego się w systemie i przesłanie go na standardowe wyjście gdzie został on przechwycony przez program MapReduce. Znajduje się on w katalogu *share\hadoop\tools\lib*

Przykładowa komenda uruchamiająca analizę ilości wielkich liter w haśle dla pliku 6 GB:

```
hadoop jar C:\hadoop\share\hadoop\tools\lib\hadoop-streaming-3.3.4.jar
-mapper "python C:\big_data\python\capital-letter-counter\mapper.py"
-reducer "python C:\big_data\python\capital-letter-counter\reducer.py"
-input /scripts/input-data/input-6G.txt
-output /scripts/output-data/capital-letter-counter
```

Przełącznik **jar** w poleceniu hadoop uruchamia plik jar dla wskazanych argumentów:

- mapper - określa skrypt odpowiedzialny za mapowanie danych

- reducer - określa skrypt, który będzie przetwarzał dane w drugim kroku
- input - ścieżka na dfs do pliku, który ma być analizowany
- output - ścieżka do katalogu (nieistniejącego), w którym zostaną zapisane wyniki przetwarzania

7.5 Proces przetwarzania danych

W czasie procesu przetwarzania w terminalu prezentowany jest aktualny status działania poszczególnych kroków wraz z postępem procentowym. Zrzut części statusu działania dla analizy wielkich liter w hasle został zamieszczony poniżej:

```

2022-12-07 20:53:05,115 INFO mapreduce.Job:  map 100% reduce 44%
2022-12-07 20:53:30,203 INFO mapreduce.Job:  map 100% reduce 45%
2022-12-07 20:53:55,311 INFO mapreduce.Job:  map 100% reduce 46%
2022-12-07 20:54:13,391 INFO mapreduce.Job:  map 100% reduce 47%
2022-12-07 20:54:38,499 INFO mapreduce.Job:  map 100% reduce 48%
2022-12-07 20:55:02,606 INFO mapreduce.Job:  map 100% reduce 49%
2022-12-07 20:55:27,729 INFO mapreduce.Job:  map 100% reduce 50%
2022-12-07 20:55:45,806 INFO mapreduce.Job:  map 100% reduce 51%
2022-12-07 20:56:10,908 INFO mapreduce.Job:  map 100% reduce 52%
2022-12-07 20:56:36,019 INFO mapreduce.Job:  map 100% reduce 53%
2022-12-07 20:56:54,133 INFO mapreduce.Job:  map 100% reduce 54%
2022-12-07 20:57:18,249 INFO mapreduce.Job:  map 100% reduce 55%
2022-12-07 20:57:43,387 INFO mapreduce.Job:  map 100% reduce 56%
2022-12-07 20:58:07,566 INFO mapreduce.Job:  map 100% reduce 57%
2022-12-07 20:58:26,687 INFO mapreduce.Job:  map 100% reduce 58%
2022-12-07 20:58:50,826 INFO mapreduce.Job:  map 100% reduce 59%
2022-12-07 20:59:15,994 INFO mapreduce.Job:  map 100% reduce 60%
2022-12-07 20:59:40,150 INFO mapreduce.Job:  map 100% reduce 61%
2022-12-07 21:00:05,315 INFO mapreduce.Job:  map 100% reduce 62%
2022-12-07 21:00:23,405 INFO mapreduce.Job:  map 100% reduce 63%
2022-12-07 21:00:47,493 INFO mapreduce.Job:  map 100% reduce 64%
2022-12-07 21:01:12,649 INFO mapreduce.Job:  map 100% reduce 65%
2022-12-07 21:01:36,761 INFO mapreduce.Job:  map 100% reduce 66%
2022-12-07 21:01:55,864 INFO mapreduce.Job:  map 100% reduce 67%
2022-12-07 21:12:24,031 INFO mapreduce.Job:  map 100% reduce 69%
2022-12-07 21:49:49,666 INFO mapreduce.Job:  map 100% reduce 77%
2022-12-07 22:43:11,169 INFO mapreduce.Job:  map 100% reduce 88%
2022-12-07 23:21:07,405 INFO mapreduce.Job:  map 100% reduce 97%
2022-12-07 23:34:39,999 INFO mapreduce.Job:  map 100% reduce 100%
2022-12-07 23:36:32,504 INFO mapreduce.Job: Job job_1670417653956_0002 completed successfully
2022-12-07 23:36:32,584 INFO mapreduce.Job: Counters: 55
    File System Counters
        FILE: Number of bytes read=334470603501
        FILE: Number of bytes written=428646828348
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=107478999328
        HDFS: Number of bytes written=82
        HDFS: Number of read operations=2405
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
        HDFS: Number of bytes read erasure-coded=0

```

Job Counters

Killed map tasks=1
Launched map tasks=801
Launched reduce tasks=1
Data-local map tasks=801
Total time spent by all maps in occupied slots (ms)=67818797
Total time spent by all reduces in occupied slots (ms)=21772179
Total time spent by all map tasks (ms)=67818797
Total time spent by all reduce tasks (ms)=21772179
Total vcore-milliseconds taken by all map tasks=67818797
Total vcore-milliseconds taken by all reduce tasks=21772179
Total megabyte-milliseconds taken by all map tasks=69446448128
Total megabyte-milliseconds taken by all reduce tasks=22294711296

Map-Reduce Framework

Map input records=13421772800
Map output records=13421772800
Map output bytes=67108864000
Map output materialized bytes=93952414400
Input split bytes=90400
Combine input records=0
Combine output records=0
Reduce input groups=6
Reduce shuffle bytes=93952414400
Reduce input records=13421772800
Reduce output records=6
Spilled Records=61203283967
Shuffled Maps =800
Failed Shuffles=0
Merged Map outputs=800
GC time elapsed (ms)=32158
CPU time spent (ms)=101994579
Physical memory (bytes) snapshot=479041138688
Virtual memory (bytes) snapshot=576751988736
Total committed heap usage (bytes)=430356561920
Peak Map Physical memory (bytes)=655618048
Peak Map Virtual memory (bytes)=771788800
Peak Reduce Physical memory (bytes)=1078685696
Peak Reduce Virtual memory (bytes)=1100038144

Shuffle Errors

BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0

File Input Format Counters

Bytes Read=107478908928

File Output Format Counters

Bytes Written=82

2022-12-07 23:36:32,585 INFO streaming.StreamJob: Output directory: /scripts/output-data/capital-letter-

Można z niego otrzymać informacje na przykład o tym ile sumarycznie linii zostało przetworzonych, informację o napotkanych błędach oraz ścieżkę, pod którą został zapisany wynik przetwarzania.

8 Wyniki analizy danych

Wyniki analiz zostały zebrane w zewnętrznym programie dołączonym do tego sprawozdania. W celu jego stworzenia należało przemyśleć następujące zagadnienia.

8.1 Przechowywanie zbiorów danych

Wyniki przeprowadzonych analiz docelowo zostały zapisane w plikach .txt, które następnie należało odpowiednio przetworzyć, wczytać do programu i w odpowiedni sposób przechowywać. W tym celu zastosowano możliwości biblioteki Pandas i struktury DataFrame.

DataFrame jest dwuwymiarową, potencjalnie niejednorodną tabelaryczną strukturą danych z oznaczonymi kolumnami i wierszami. Jest to jedna z najczęściej używanych struktur danych w nowoczesnej analityce danych.

Wyniki powyżej opisanych analiz, dostarczone w formacie .txt, wczytano do macierzy (numpy array). Następnie na ich podstawie utworzono cztery zbiory danych typu DataFrame. Pliki te były w następującym formacie:

1. Liczba dużych liter w hasle wraz z wartościami z wartościami znormalizowanymi do całkowitej ilości haseł:

	6G	100G	6G_NORM	100G_NORM	Amount
0	62192448	906992640	0.066791	0.067576	0
1	223303392	3239264404	0.239815	0.241344	1
2	320724144	4626236524	0.344440	0.344681	2
3	230333480	3302606112	0.247365	0.246063	3
4	82712656	1178505952	0.088829	0.087806	4
5	11881376	168167168	0.012760	0.012529	5

Rysunek 8: DataFrame - liczba dużych liter w hasle

2. Liczba cyfr znajdujących się na końcu hasła wraz z wartościami znormalizowanymi do całkowitej ilości haseł:

	6G	100G	6G_NORM	100G_NORM	Amount
0	780962416	1815640370	0.838710	0.838710	1
1	125961680	292844700	0.135276	0.135276	2
2	20317400	47227000	0.021820	0.021816	3
3	3286000	7590000	0.003529	0.003506	4
4	620000	1500000	0.000666	0.000693	5

Rysunek 9: DataFrame - liczba cyfr znajdujących się na końcu hasła

3. Liczba cyfr znajdujących się na początku hasła wraz z wartościami znormalizowanymi do całkowitej ilości haseł:

	6G	100G	6G_NORM	100G_NORM	Amount
0	780962416	7683694720	0.838710	0.838801	1
1	125961680	1239305600	0.135276	0.135291	2
2	20317400	199888000	0.021820	0.021821	3
3	3286000	32240000	0.003529	0.003520	4
4	620000	5200000	0.000666	0.000568	5
5	0	1000000	0.000000	0.000109	6

Rysunek 10: DataFrame - liczba cyfr znajdujących się na początku hasła

4. Liczba zduplikowanych lub potrojonych takich samych znaków w hasle:

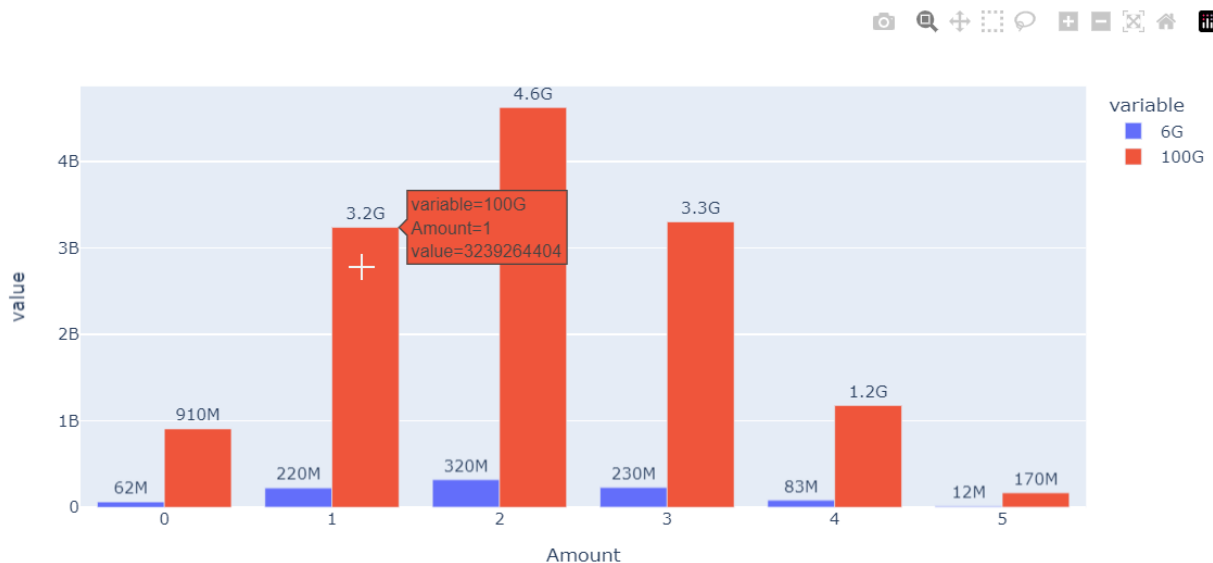
	Record_2	6G_2	100G_2	Record_3	6G_3	100G_3
0	00	11532	402981	000	953435	28425667
1	11	11532	402981	111	953435	28425667
2	22	11532	402981	222	953435	28425667
3	33	11532	402981	333	953435	28425667
4	44	11532	402981	444	953435	28425667
5	55	11532	402981	555	953435	28425667
6	66	11532	402981	666	953435	28425667
7	77	11532	402981	777	953435	28425667

Rysunek 11: DataFrame - liczba zduplikowanych lub potrojonych takich samych znaków w hasle

Ze względu na czasochłonność procesu wczytywania powyższych danych, zrobiono to w oddzielnym programie, a DataFrame'y zapisano w postaci .pickle, w celu łatwiejszego ich odczytu w docelowym programie do wizualizacji danych.

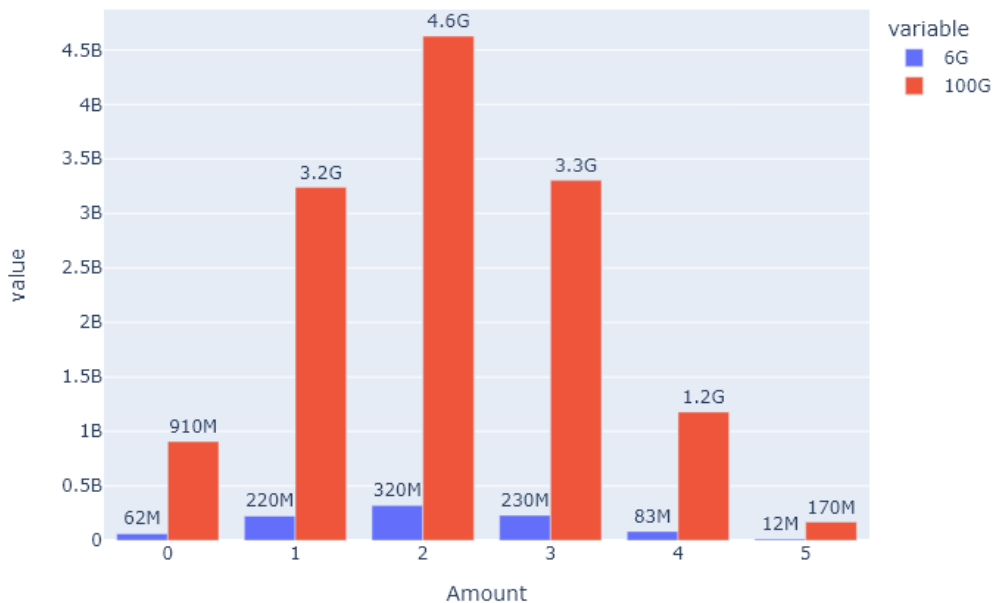
8.2 Wizualizacja wyników analizy

W celu wizualizacji wyników analiz zdecydowano się na wykorzystanie możliwości biblioteki Plotly. Plotly jest interaktywną biblioteką do prezentacji danych, typu open source, która obsługuje ponad 40 różnych typów wykresów i sposobów wizualizacji analiz. Biblioteka ta w przystępny sposób pozwala na tworzenie interaktywnych wykresów. Jej interaktywność szczególnie łatwo dostrzec wykorzystując ją w Jupiterze.



Rysunek 12: Przykład ineraktywnego wykresu wygenerowanego w Jupiterze

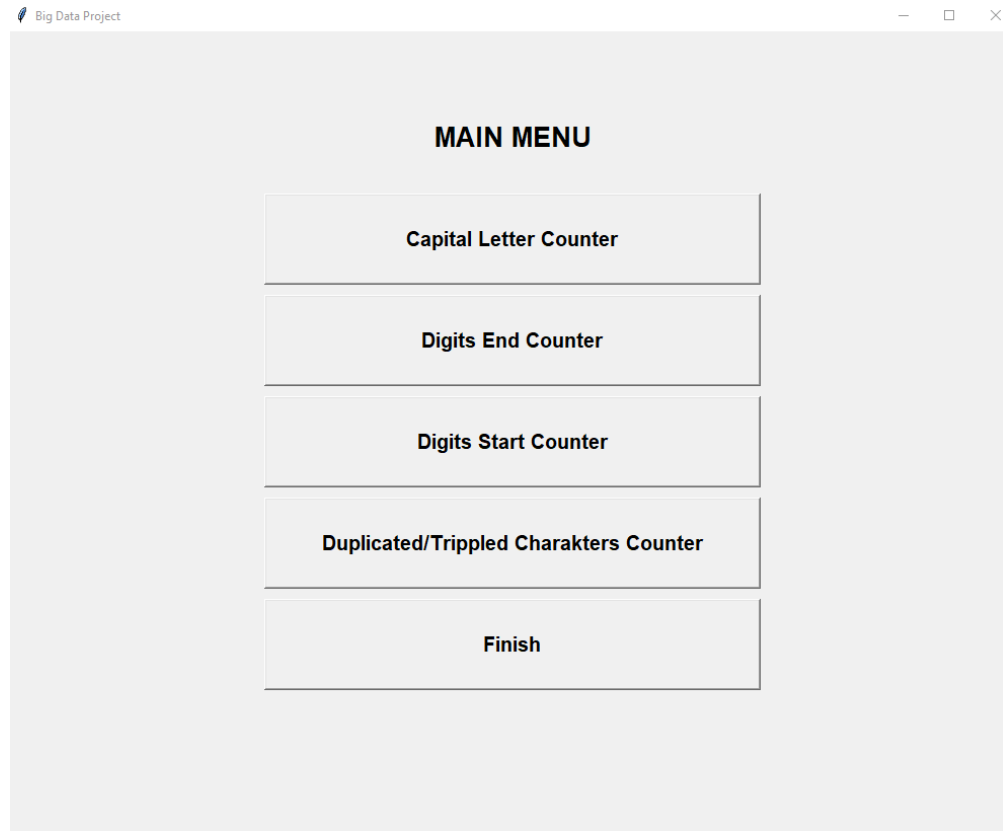
Jak widać na grafice powyżej wykres wygenerowany za pomocą biblioteki plotly w Jupiterze jest całkowicie interaktywny. Pozwala na zapisanie wykresu, przybliżanie i manipulację. Ponadto po najechaniu na jeden z wygenerowanych słupków pokazuje się monit informujący o danych jego dotyczących. W celu ułatwienia analizy, w dalszej części programu do wizualizacji, jej wyniki zaprezentowano w postaci wykresów słupkowych.



Rysunek 13: Przykład wykresu słupkowego użytego do wizualizacji wyników analiz

8.3 Interaktywny program do wizualizacji danych

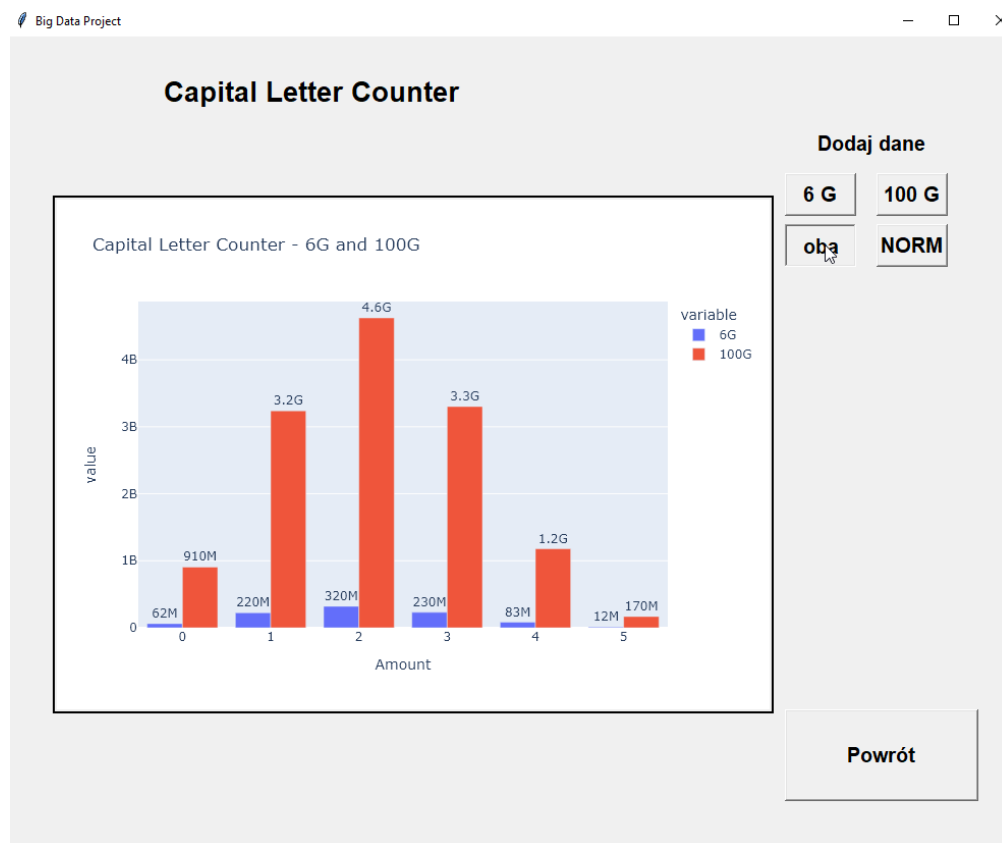
Wszystkie wygenerowane analizy zebrano w jednym programie do wizualizacji wyników analiz:



Rysunek 14: Program do wizualizacji wyników analiz

Program stworzono wykorzystując bibliotekę Tkinter. Jest to biblioteka składająca się z kilku modułów, które razem umożliwiają stworzenie interfejsu graficznego(GUI).

W stworzonym interfejsie pokazuje się możliwość wyboru przeprowadzonej analizy - Capital Letter Counter, Digits End Counter, Digits Start Counter, Duplicated/Tripped Characters Counter. Przechodząc do wybranego badania pokazuje się możliwość wyboru wyświetlenia kilku wykresów, w zależności od przeprowadzonej analizy. Następnie po wyborze danego wykresu wyświetli się wybrana grafika.



Rysunek 15: Program do wizualizacji wyników analiz - wybrana analiza

Podczas tworzenia powyższej aplikacji natrafiono na problem połączenia wykresów generowanych za pomocą Plotly a interfejsem graficznym stworzonym za pomocą Tkinter'a. Ze względu na to, że obie biblioteki nie są kompatybilne ze sobą, a więc nie da się wygenerować wykresu w samym interfejsie. Problem ten został rozwiązany poprzez zapisywanie wygenerowanego wykresu w formie obrazka z rozszerzeniem .png, a następnie wczytywanego do interfejsu. W wyniku takiego działania traci się jednak bardzo dużo na interaktywności, którą oferuje biblioteka Plotly. Ponadto prowadzi to do wydłużenia działania programu. Z tego względu do sprawozdania dołączono plik Jupiter, wyświetlający wszystkie wykresy wygenerowane w programie do wizualizacji.

9 Wnioski

1. Apache Hadoop to program, który można uruchamiać na każdym systemów operacyjnych typowo przeznaczonych pod serwery takie jak Windows czy Linux.
2. Konfiguracja środowiska sprowadza się do edytowania plików tekstowych zapisanych w formacie XML, co pozwala na szybką edycję parametrów.
3. Rozproszone przetwarzanie pozwala znacząco przyspieszyć działanie na dużych zbiorach danych szczególnie, gdy dysponujemy większą liczbą maszyn, na których dane będą przechowywane.
4. Ważne jest odpowiednie zbalansowanie danych pomiędzy poszczególnymi maszynami, tak aby na każdej była przechowywana odpowiednia ich ilość. Zapewni to możliwość równoległego ich przetwarzania na tych maszynach.
5. Zastosowanie replikacji danych pozwala na stabilne działanie systemu nawet w przypadku awarii. W projekcie po odłączeniu jednej maszyny analiza danych mogła z powodzeniem działać na pozostałej działającej maszynie.
6. Zarządzanie zasobami w Apache Hadoop pozwala przekazać więcej pracy na maszynę, która wykonuje zadania szybciej, dzięki temu automatycznie równoważy się obciążenie poszczególnych maszyn ze względu na moc obliczeniową.

10 Literatura

Podczas pracy nad projektem korzystano z następujących źródeł literaturowych:

1. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
2. <https://github.com/LandGrey/pydictor>
3. <https://github.com/steveloughran/winutils>
4. <https://hadoop.apache.org/releases.html>
5. <https://towardsdatascience.com/installing-hadoop-3-1-0-multi-node-cluster-on-ubuntu-16-04-step-by-step-8d1954b31505>