Sprawozdanie z projektu dyplomowego

Wykorzystanie ekosystemu Hadoop do analizy danych strumieniowych

Mikołaj Burdzy 319 023

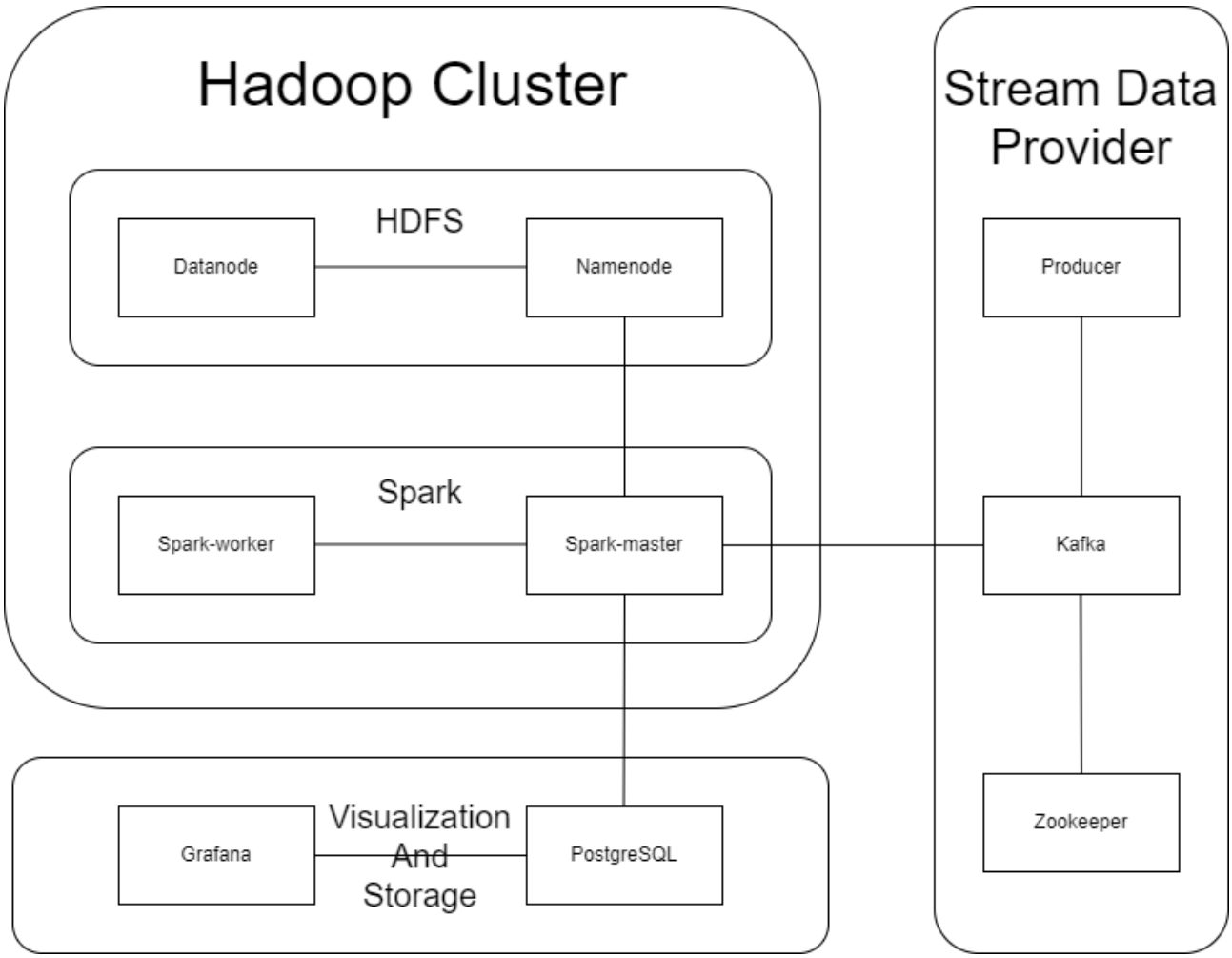
- ## Opis problemu
  W projekcie skupiam się na analizie sentymentu postów internautów na platformie X na temat różnych polityków sceny amerykańskiej. Sentyment jest określany za pomocą modelu
- ## Dane
  W projekcie głównymi danymi są wypowiedzi internautów z platformy X. W celu przyśpieszenia działania samej analizy oraz strumieniowania danych za pomocą kafki, dane te przechodzą preprocesing co znacznie zmniejsza ich rozmiar i normalizuje je do formatu zrozumiałego przez resztę komponentów.
  Surowe są w formacie csv z kolumnami:

  ```
  user_name,user_location,user_description,user_created,user_followers,user_friends,user_favourites,user_verified,date,text,hashtags,source,is_retweet
  ```

  Większość z tych kolumn na rzecz tej analizy jest zbędna, a część wierszy jest w złym formacie więc trzeba zadbać o ich pominięcie podczas wczytywania. Dodawana jest także nowa kolumna do której wpisywany jest polityk którego dotyczy dana wypowiedź. Polityk z posta jest wychwytywany za pomocą zwykłych wyrażeń regularnych. Po przeróbce dane mają format:

  ```
  timestamp,tweet,politician
  ```

- ## Architektura



Architektura projektu jest podzielona na 3 zasadnicze części, klaster Hadoop, wirtualnego dostawcę danych strumieniowych oraz część odpowiedzialną za przechowywanie i wizualizacje wyników. Sam klaster Hadoop jest podzielony na 2 części, system przechowywania plików HDFS oraz głównego wykonawce kodu Spark'a.

- Spark odpowiada za główne obliczenia wykonywane w projekcie.
- HDFS jest używamy do przechowywania danych źródłowych i danych wynikowych działania niektórych spark jobów. Nie jest on głównym źródłem danych dla sparka ani głównym miejscem zapisu wyników jego pracy, ale został skonfigurowany tak aby połączenie ze sparkiem było możliwe, co prezentują skrypty testowe.
- Stram Data Provider tworzy dane strumieniowe i przy użyciu kafki przesyła je do subskrybentów.
- Grafana jest używana do tworzenia wizualizacji na podstawie danych wyprodukowanych przez spark'a. Które zostały zapisane w bazie danych PostgreSQL.

• Opis komponentów

W projekcie znajduje się 9 kontenerów, oto ich krótki opis:

- Kafka
  - Po uruchomieniu tworzy automatycznie temat, następnie zaczyna przyjmować dane i rozsyłać je do subskrybentów.
- Data Producer

        - Po włączeniu czeka na włączenie kafki oraz inicjalizacje tematu, następnie zaczyna produkować dane.
    - Zookeeper
        - Koordynuje prace kafki.
    - Spark-master
        - Uruchamia skrypty obliczeniowe jako job'y które następnie dzieli na taski i dystrybuuje je spark-worker'om.
    - Spark-worker
        - Wykonuje taski otrzymane od spark-master'a.
    - Namenode
        - Węzeł nazw, jego główną rolą jest komunikacja między systemem plików HDFS a kontererami Spark'a.
    - Datanode
        - Węzeł danych, jego główną rolą jest przechowywanie danych w obrębie systemu pików HDFS.
    - PostgreSQL
        - Baza danych do której spark zapisuje wyniki obliczeń.
    - Grafana
        - Narzędzie do wizualizacji skonfigurowane pod bazę danych PostgreSQL z której pobiera dane.
-

Użycie

Projekt składa się w całości z kontenerów co gwarantuje jego przenośność i możliwość uruchomienia na nowej maszynie w relatywnie prosty sposób. Na maszynie musi być zainstalowany program Docker Desktop a w przypadku systemów unix Docker Engine. W tym celu należy skopiować repozytorium git, a następnie użyć komendy:

```
docker compose up -d –build
```

Jeśli kontenery się utworzyły poprawnie, można przejść do uruchamiania konkretnych skryptów.

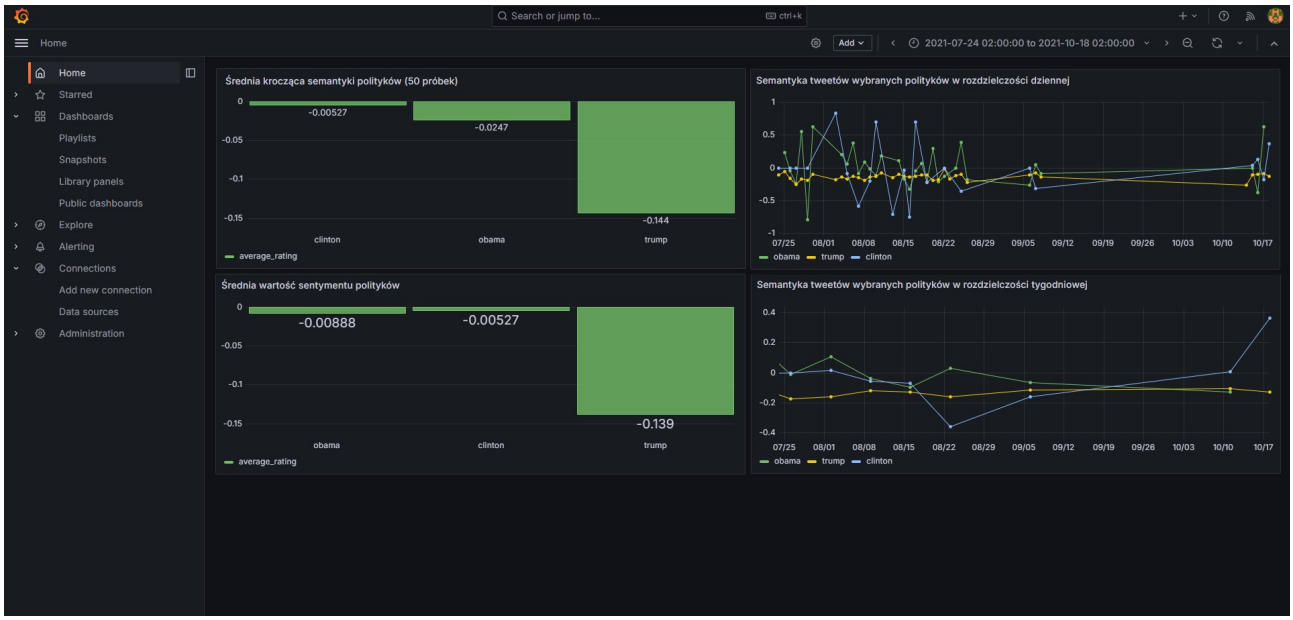
W projekcie znajduje się ich kilka, każdy powstał w konkretnym celu.

- wordcount.py
    - To skrypt testowy sprawdzający głównie komunikacje z namenode'm oraz działanie samego skryptu w środowisku rozproszonym jako spark jobs.
    - Dane wejściowe na których skrypt będzie wykonywał operacje zostaną wczytane z HDFS. Przed jego wywołaniem należy wgrać dane do systemu plików:
      ```
      docker exec -it namenode bash /scripts/load_data.sh
      ```
    - Następnie można go uruchomić używając komendy:
      ```
      docker exec -it spark-master
      /opt/bitnami/spark/bin/spark-submit --master
      spark://spark-master:7077 /spark_jobs/wordcount.py
      ```
    - Skrypt zapisze wynik wywołania również w HDFS. Aby je wyświetlić:
      ```
      docker exec -it namenode bash /scripts/show_output.sh
      ```
- csv_to_kafka.py
    - To skrypt (jedyny wywoływany nie na sparku jako job a na osobnym, do tego przeznaczonym kontenerze) symulujący producenta danych. Po włączeniu wrzuca pierwsze 500 linii pliku naraz (w celu zbudowania pewnej historii danych aby nie trzeba było czekać z trenowaniem modelu na wystarczającą ilość próbek uczących) a następnie w sposób strumieniowy z częstotliwością 1Hz.
    - Włącza się automatycznie po tym jak wykryje, że kafka się uruchomiła i został stworzony temat.

- kafka_stream_output.py
  - To skrypt testujący komunikacje z kafką i pobieranie z niej danych. Po włączeniu łączy się do tematu kafki i wyświetla w konsoli dane które konsumuje.
  - Uruchamiany jest:
    ```
    docker exec -it spark-master
    /opt/bitnami/spark/bin/spark-submit --master
    spark://spark-master:7077
    /spark_jobs/kafka_stream_output.py
    ```
- main.py
  - To główny skrypt obliczeniowy projektu, przeprowadza on analizę sentymentu wiadomości oraz zapisuje wyniki w bazie danych PostgreSQL.
  - Uruchamiany jest:
    ```
    docker exec -it spark-master
    /opt/bitnami/spark/bin/spark-submit --master
    spark://spark-master:7077 --jars /jars/postgresql-
    42.2.29.jre7.jar --driver-class-path /jars/postgresql-
    42.2.29.jre7.jar /spark_jobs/main.py
    ```

Wizualizacja

Wizualizacje w projekcie przeprowadziłem przy użyciu narzędzia Grafana. Narzędzie to jest postawione na osobnym kontenerze do którego mamy dostęp przez przeglądarkę. Dużą wagę poświęciłem temu aby użytkownik mógł zobaczyć wyniki obliczeń w jak najprostszy sposób, bez konieczności konfiguracji widoków czy wykresów od nowa. Było to problematyczne z powodu założenia konteneryzacji i przenośności. Po uruchomieniu kontenerów oraz skryptu main.py w bazie danych są zapisywane dane. W celu obserwacji wyników należy otworzyć w przeglądarce adres localhost:3000. Następnie zalogować się jako użytkownik admin z hasłem admin. Naszym oczom powinna ukazać się strona główna grafany wyglądająca mniej więcej tak:



Widok który spotkamy po uruchomieniu narzędzia będzie się różnił wyświetlanymi danymi, powyższy przykład jest efektem kilku godzin obliczeń.

W panelu widzimy 4 wykresy.

W pierwszej kolumnie są wykresy średnich, odpowiednio średniej kroczącej z ostatnich 50 postów dla każdego z polityków oraz średniej z całych danych.

W drugiej kolumnie są wykresy sentymentu tweetów agregowane odpowiednio dziennie i tygodniowo. Agregacja dzienna jest dokładniejsza ale w przypadku mniej popularnych polityków (Clinton, Obama) w ciągu jednego dnia jest o nich na tyle mało tweetów, że trudno jest zobaczyć ogólny trend, a widać raczej szum. Agregacja tygodniowa może ukryć niektóre subtelne wzrosty lub spadki danych polityków, ale gwarantuje dobre zwizualizowanie ogólnego długoterminowego trendu.

• Napotkane problemy i rozwiązania

Pierwszym fundamentalnych problemem był brak dostępu do wpisów z platformy X w czasie rzeczywistym. Od 09.02.2023, Twitter API nie oferuje już pobierania tweetów w ramach darmowego pakietu. Teraz zapewnia on tylko postowanie wpisów a nie ich pobieranie. Podjąłem również próbę web-scrapingu tweetów przy użyciu takich narzędzi jak python selenium. Niestety to mi się nie udało ponieważ bardzo mi zależało na pełnej konteneryzacji rozwiązania a pakiet selenium wymagał kilku konfiguracji systemowych takich jak kompatybilna przeglądarka oraz odpowiednie sterowniki między przeglądarką i pakietem selenium. Sprawiało to problemy z konteneryzacją. W zamian zdecydowałem się na podejście w którym pobieram cały zbiór danych tweetów (historycznych) a następnie je strumieniuje symulując ich produkcję w czasie rzeczywistym.
Następnym problemem który napotkałem był sposób wizualizacji wyników. Pierwotnym pomysłem było użycie jupyter notebook’a ale nie było to przemyślane rozwiązanie. Strumieniowy charakter danych wymaga aby wizualizacje się odświeżały w czasie rzeczywistym na co notebook nie pozwala. Narzędziem które na to pozwala oraz które łatwo pozwala na prostą konteneryzacje jest biblioteka python Plotly Dash. Niestety rozwiązania tego nie udało mi się wprowadzić ponieważ aplikacja Dash wymaga osobnego wątku, tak samo jak spark. Sprawiało to problemy z wątkami które w pythonie nie są najprostsze do rozwiązania. Znalazłem gotowe ‘out of the box’ rozwiązanie w postaci grafan’y. Jedynym problemem z tym narzędziem była jej inicjalizacja, aby po zbudowaniu i uruchomieniu kontenera, tak zwany ‘provisioning’. Było to trudne zadanie, ale po poprawnej konfiguracji narzędzie działa perfekcyjnie.

• Wnioski

Fundamentalny wniosek (zgodny ze zdrowym rozsądkiem) jaki można wyciągnąć z przeprowadzonej przeze mnie analizy jest taki, że o politykach zawsze mówi się źle. Niezależnie od polityka, średnia semantyczna jest zawsze ujemna w wystarczająco odległym horyzoncie czasowym. Jednakże dzięki możliwości obserwacji zmiany tej średniej semantycznej w czasie, można obserwować spadki i wzrosty danych polityków. Na podstawie tych obserwacji można obiektywnie oceniać ich działania, np. czy konkretna wypowiedź na konferencji prasowej miała pozytywny czy negatywny wpływ na reputacje danej osoby.

• Kierunki rozwoju

Projekt ten jest prosty w swoich założeniach i implementacji. Na rozwój projektu mam następujące pomysły:

- Trafniejsze wyznaczanie polityków których dotyczy dany post.
  Obecnie tą informacje otrzymujemy za pomocą wyrażeń regularnych, ale można by do tego zadania użyć klasyfikatora języka naturalnego. Np takiego już wytrenowanego z

hugging face'a. Nie zdecydowałem się na to rozwiązanie ponieważ wymaga to dużo więcej zasobów, a sama analiza przebiegała by wielokrotnie wolniej.

- Zwiększenie zasobów obliczeniowych poprzez stworzenie klastra obliczeniowego spark nie w środowisku wirtualnym a w rzeczywistości.
  Połączenie mocy obliczeniowej wielu maszyn przyśpieszy analizę albo pozwoli użyć bardziej wyrafinowanych narzędzi jak chociażby wyżej wymieniony model do klasyfikacji języka naturalnego.

- Rozszerzenie analizy o dodatkowe parametry z dostępnych danych.
  Jak widać było wyżej w sekcji Dane, mam do dyspozycji dużo więcej (meta)danych o wpisach. Można by np. nanieść na mapę wyniki z analizy w celu zrobienia geo-analizy.

- Dobrym ruchem było by przeanalizowanie danych z okresu bardziej intensywnego politycznie, np. z okresu wyborów prezydenckich w Stanach z roku 2016.
  Obecne dane są z okresu od 2021-07-24 do 2022-08-20, nie jest to żaden specjalnie ciekawy okres politycznie. Próbę pozyskania takich danych podjąłem ale nie udało mi się znaleźć darmowych danych z tego okresu (płatne oczywiście są).

Bibliografia

- Model do analizy sentymentu:
  https://github.com/cjhutto/vaderSentiment
  https://medium.com/@sharma.tanish096/sentiment-analysis-using-pre-trained-models-and-transformer-28e9b9486641

- Zbiory danych
  https://data.world/alexfilatov/2016-usa-presidential-election-tweets
  https://mega.nz/folder/coA3DAwS#l44jOgGdWivTq973e8qPiw
  https://www.kaggle.com/datasets/kaushiksuresh147/political-tweets

- Grafana:
  https://grafana.com/docs/grafana/latest/panels-visualizations/visualizations/
  https://grafana.com/docs/grafana/latest/datasources/postgres/
  https://grafana.com/docs/grafana/latest/administration/provisioning/?utm_source=grafana_ds_list#data-sources
  https://grafana.com/docs/grafana/latest/dashboards/use-dashboards/#set-dashboard-time-range
  https://grafana.com/docs/grafana/latest/panels-visualizations/visualizations/time-series/
  https://grafana.com/docs/grafana/latest/setup-grafana/installation/docker/#use-environment-variables-to-configure-grafana
  https://community.grafana.com/t/refresh-live-dashboards/84959/3
  https://grafana.com/docs/grafana/latest/panels-visualizations/query-transform-data/#:~:text=To%20add%20a%20query%3A&text=Click%20the%20Query%20tab.,you%20add%20the%20first%20panel.
  https://grafana.com/docs/grafana/latest/getting-started/build-first-dashboard/
  https://community.grafana.com/t/where-is-the-server-log-file-when-using-grafana-web/34262
  https://volkovlabs.io/blog/provisioning-grafana-20230509/

- Postgres:
  https://jdbc.postgresql.org/download/

- Stack overflow
  https://stackoverflow.com/
  https://stackoverflow.com/questions/76170810/tweepy-twitter-api-v2-retrieve-tweets-on-free-access
  https://stackoverflow.com/questions/69206745/unable-to-load-web-page-with-seleniumwire
  https://stackoverflow.com/questions/44290548/how-to-write-pyspark-dataframe-to-hdfs-and-then-how-to-read-it-back-into-datafra
  https://stackoverflow.com/questions/53087752/unable-to-load-libhdfs-when-using-pyarrow
  https://stackoverflow.com/questions/58237848/how-to-serve-data-from-hdfs-fast-in-realtime-in-my-services
  https://stackoverflow.com/questions/34948296/using-pyspark-to-connect-to-postgresql
  https://stackoverflow.com/questions/38825836/write-spark-dataframe-to-postgres-database

- HDFS python
  https://pypi.org/project/hdfs3/
  https://hdfs3.readthedocs.io/en/latest/
  https://github.com/erikmuttersbach/libhdfs3
  https://medium.com/@arush.sharma1/connecting-hadoop-hdfs-with-python-267234bb68a2
  https://medium.com/@arush.sharma1/connecting-hadoop-hdfs-with-python-267234bb68a2
  https://community.cloudera.com/t5/Support-Questions/libhdfs-missing/td-p/192982
  https://community.cloudera.com/t5/Support-Questions/storage-dataframe-as-textfile-in-hdfs/m-p/108706

- Spark Python
  https://spark.apache.org/docs/latest/sql-data-sources-load-save-functions.html
  https://hub.docker.com/r/bitnami/spark

- Kafka
  https://github.com/Eneco/kafka-connect-twitter
  https://www.youtube.com/watch?v=cGFjd7ox4h4
  https://www.conduktor.io/kafka/kafka-producers/
  https://www.javatpoint.com/kafka-topics
  https://www.conduktor.io/kafka/kafka-topics-cli-tutorial/

- Tweets scraping
  https://stackoverflow.com/questions/69206745/unable-to-load-web-page-with-seleniumwire
  https://github.com/wkeeling/selenium-wire/issues/49
  https://www.marekrost.cz/selenium-wire-how-to-resolve-blinker-saferef
  https://medium.com/@nimk/scraping-tweets-without-twitter-api-and-free-d576ace29f4c
  https://docs.tweepy.org/en/stable/examples.html
  https://developer.x.com/en
  https://developer.x.com/en/docs/tutorials/stream-tweets-in-real-time
  https://www.confluent.io/hub/jcustenborder/kafka-connect-twitter