Sprawozdanie z projektu

Przetwarzanie strumieni danych w systemach Big Data

Crimes in Chicago - Spark Structured Streaming

Instrukcja uruchomienia przetwarzania

Uruchomienie klastra

Uruchom `CLOUD SHELL Terminal`, a następnie utwórz klaster poniższym poleceniem.

```
gcloud dataproc clusters create ${CLUSTER_NAME} \
--enable-component-gateway --region ${REGION} --subnet default \
--master-machine-type n1-standard-4 --master-boot-disk-size 50 \
--num-workers 2 --worker-machine-type n1-standard-2 --worker-boot-disk-size 50 \
--image-version 2.1-debian11 --optional-components ZOOKEEPER,DOCKER \
--project ${PROJECT_ID} \
--metadata "run-on-master=true" \
--initialization-actions \
gs://goog-dataproc-initialization-actions-${REGION}/kafka/kafka.sh
```

Przejdź do nowo utworzonego klastra i otwórz pierwszy terminal maszyny `master`.

Przeniesienie skryptów i innych plików projektu

Przenieś wszystkie pliki projektu na maszynę (możesz pominąć `sprawozdanie.md` oraz `sprawozdanie.pdf`) poprzez opcję `UPLOAD FILE` w prawym górnym rogu terminalu.

Uruchomienie skryptu setupu

Nadaj prawo do wykonywania plikowi `setup.sh`.

```
chmod +x setup.sh
```

Uruchom skrypt `setup.sh` poleceniem

```
./setup.sh
```

Przeniesienie plików z danymi

Przenieś pliki z danymi z zasobnika na klaster. Uruchom skrypt `copy_data.sh` z lokalizacją plików jako parametrem. Przykładowo, dla hierarchii katalogów

```
pbd-23-sp
    crimes-data
        crimes-in-chicago_result
            part-00000-10b00d71-feee-417e-b0bc-888b1e6afec7-c000.csv
            part-00001-10b00d71-feee-417e-b0bc-888b1e6afec7-c000.csv
            ...
            part-00099-10b00d71-feee-417e-b0bc-888b1e6afec7-c000.csv
        Chicago_Police_Department_-
_Illinois_Uniform_Crime_Reporting__IUCR__Codes.csv
```

polecenie uruchamiające skrypt będzie wyglądało następująco:

```
./copy_data.sh gs://pbd-23-xyz/crimes-data
```

Operacja może chwilę potrwać.

Program przetwarzający dane

Otwórz nowy terminal korzystając z przycisku `Settings -> New Connection`. Uruchom na nim skrypt `processing.sh` z parametrem `delay`, parametrem `D` oraz parametrem `P`. Przykładowe wywołanie dla trybu `A` w konfiguracji powodującej znalezienie wielu anomalii przedstawione zostało poniżej.

```
./processing.sh A 2 10
```

Dla poniższego wywołania liczba anomalii powinna być bliska zeru.

```
./processing.sh A 14 99
```

W terminalu będą się pojawiały logi Sparka, nie będzie to raczej nic ciekawego.

Skrypt wyświetlający dane z przetwarzania czasu rzeczywistego

Otwórz nowy terminal korzystając z przycisku `Settings -> New Connection`. Uruchom na nim skrypt `read_results.sh`. W terminalu będą pojawiał się bieżący stan tabeli z agregacjami. `CTRL + C` może nie zakończyć działania skryptu. Wtedy polecam `CTRL + Z`.

```
./read_results.sh
```

Skrypt wyświetlający znalezione anomalie

Otwórz nowy terminal korzystając z przycisku `Settings -> New Connection`. Uruchom na nim skrypt `anomalies.sh`. W terminalu będą pojawiały się informacje o wykrywanych anomaliach. `CTRL + C` kończy

działanie skryptu.

```
./anomalies.sh
```

Program wysyłający dane

Otwórz nowy terminal. Uruchom na nim skrypt `producer.sh`. `CTRL + C` kończy działanie skryptu.

```
./producer.sh
```

Resetowanie środowiska

Aby zresetować środowisko i usunąć wszystkie pliki projektowe, wykonaj poniższą komendę uruchamiającą skrypt `cleanup.sh`. Może się to przydać w przypadku wykonania błędu. Nie trzeba będzie wtedy usuwać klastra i uruchamiać go ponownie. Można wtedy rozpocząć uruchamianie projektu od kroku `Przeniesienie skryptów i innych plików projektu`.

```
./cleanup.sh
```

Kryteria projektu

Producent; skrypty inicjujące i zasilający

Skrypty zawarte w projekcie zostały dokładniej omówione w instrukcji uruchomienia przetwarzania znajdującej się powyżej. Poniżej znajduje się krótkie podsumowanie. Obecne skrypty:

- setup.sh - przygotowanie środowiska so przetwarzania, pobranie sterownika jdbc do bazy danych PostgreSQL, nadanie uprawnień wykonywania pozostałym skryptom, utworzenie tematu Kafki, przygotwanie ujścia danych (bazy danych w PostgreSQL),
- copy_data.sh - przekopiowanie danych do przetwarzania z zasobnika, o adresie podanym jako parametr do lokalnego systemu plików,
- read_results.sh - wyświetlanie wyników zapisywanych w tabeli w PostgreSQL,
- anomalies.sh - wyświetlanie anomalii z tematu Kafki `kafka-output`,
- processing.sh - uruchamianie producenta Kafki,
- producer.sh - przeniesienie pliku statycznego do HDFS, uruchominie programu przetwarzającego dane,
- cleanup.sh - usunięcie tematu kafki, zatrzymanie i usunięcie kontenera, usunięcie sterownika jdbc, usunięcie danych, usunięcie checkpointów, usunięcie skryptów, producenta Kafki, programu w pythonie oraz na końcu skryptu `cleanup.sh`.

Skryptami, które mają największe znaczenie w kontekście inicjalizacji to `setup.sh` i `copy_data.sh`. To one powinny zostać wykonane na początku. Skrypt `producer.sh` odpowiedzialny jest za uruchomienie producenta Kafki, który znajduje się w pliku `KafkaProducer.jar`. Jest to producent przyjmujący jako parametry `INPUT_DIR`, czyli katalog z danymi, `SLEEP_TIME`, czyli czas bezczynności po skończeniu wysyłania zawartości pliku, `TOPIC`, czyli nazwa tematu, na który wysyłane mają być dane, `HEADER_LENGTH`, czyli liczba

wierszy zawierających nagłówek oraz `KAFKA_BROKER`, czyli Broker Kafki. Paramtery, z jakimi uruchamiany jest producent, to:

- `INPUT_DIR` - `"./data/crimes-in-chicago_result/"`
- `SLEEP_TIME` - `1`
- `TOPIC` - `"kafka-input"`
- `HEADER_LENGTH` - `1`
- `KAFKA_BROKER` - `"<nazwa_klastra>-w-0:9092"`

Utrzymanie obrazu czasu rzeczywistego – transformacje

Dane statyczne z pliku `Chicago_Police_Department_-_Illinois_Uniform_Crime_Reporting__IUCR__Codes.csv` pobierane są w następujący sposób.

```
# Dane statyczne
iucr_codes = spark.read.option("header", True) \
    .csv("hdfs:///Chicago_Police_Department_-_Illinois_Uniform_Crime_Reporting__IUCR__Codes.csv")
```

Kod danych otrzymywanych strumieniowo przedstawiony został poniżej. Na samym definiowany jest schemat danych. Następnie następuje połączenie z tematem kafki `kafka-input`, z którego przychodzą dane. Po połączeniu z tematem, dane mogą zostać odczytane. Zmieniany jest typ kolumny `Date` tak, żeby można było ustawić na niej znacznik czasowy. Później dane strumieniowe oraz statyczne są łączone po kolumnie `IUCR`. Przetwarzanie polega na utworzeniu kolumny `month` oraz `primary_description`, a następnie pogrupowaniu danych na poziomie miesiąca, głównej kategorii przestępstwa i dzielnicy. Dla każego wiersza wyliczana jest liczba wszystkich przestępstw (do kolumny `total_crimes`), liczba przestępstw zakończonych aresztowaniem (do kolumny `crimes_with_arrest`), liczba przestępstw zakończonych związanych z przemocą domową (do kolumny `domestic_violence_crimes`) oraz liczba przestępstw rejestrowanych przez FBI (do kolumny `fbi_index_crimes`).

```
# Schemat danych
crimes_schema = "id STRING, date STRING, iucr STRING, arrest BOOLEAN, domestic BOOLEAN, " \
    "district DOUBLE, comarea DOUBLE, latitude DOUBLE, longitude DOUBLE"

# Połączenie z tematem Kafki
host_name = socket.gethostname()
ds1 = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", f"{host_name}:9092") \
    .option("subscribe", "kafka-input") \
    .load()

# Dane strumieniowe
valuesDF = ds1.select(expr("CAST(value AS STRING)").alias("value"))

dataDF = valuesDF.select( \
    from_csv(valuesDF.value, crimes_schema) \
```

```
    .alias("data")) \
    .select("data.*")

# Zamienienie Date na timestamp
dataDF = dataDF.withColumn("Date", to_timestamp(dataDF["Date"]))

# Dodanie watermarka
dataDF = dataDF.withWatermark("Date", "1 day")

# Połączenie tabel
joined_df = dataDF.join(iucr_codes, "IUCR")

# Przetwarzanie
resultDF = joined_df \
    .withColumn("month", date_format(col("date"), "yyyy-MM")) \
    .withColumn("primary_description", col("PRIMARY DESCRIPTION")) \
    .groupBy("month", "primary_description", "district") \
    .agg(
        count("*").alias("total_crimes"),
        count(when(col("Arrest") == True, True)).alias("crimes_with_arrest"),
        count(when(col("Domestic") == True,
True)).alias("domestic_violence_crimes"),
        count(when(col("INDEX CODE") == "I", True)).alias("fbi_index_crimes")
    )
```

Utrzymanie obrazu czasu rzeczywistego – obsługa trybu A

Tryb `A` lub `C` obsługiwane są za pomocą `.outputMode(output_mode)`. Jeżeli tryb `output_mode` ustawiony jest na wartość `update`, to mamy do czynienia z najmniejszymi możliwymi opóźnieniami, z aktualizacjami wyników obrazu czasu rzeczywistego.

```
# Ustawienie trybu programu
    if delay == 'A':
        output_mode = "update"
    elif delay == 'C':
        output_mode = "append"
    else:
        raise ValueError("Zła wartość parametru delay. Użyj 'A' lub 'C'.")

# Przesłanie danych do ujścia
query = resultDF \
    .writeStream \
    .outputMode(output_mode) \
    .option("checkpointLocation", "/tmp/checkpoints_etl") \
    .foreachBatch ( \
        lambda batchDF, batchId: \
        batchDF.write \
        .format("jdbc") \
        .mode("append") \
        .option("url", f"jdbc:postgresql://{host_name}:8432/streamoutput") \
        .option("dbtable", "crime_stats") \
```

```
            .option("user", "postgres") \
            .option("password", "mysecretpassword") \
            .save() \
        ).start()
```

Utrzymanie obrazu czasu rzeczywistego – obsługa trybu C

Podobnie jest również w przypadku trybu `C`. `output_mode` ustawiony zostaje na `append` i mamy do czynienia z najmniejszymi możliwymi opóźnieniami przy uwzględnieniu braku aktualizacji wyników obrazu czasu rzeczywistego.

```
# Ustawienie trybu programu
    if delay == 'A':
        output_mode = "update"
    elif delay == 'C':
        output_mode = "append"
    else:
        raise ValueError("Zła wartość parametru delay. Użyj 'A' lub 'C'.")

# Przesłanie danych do ujścia
query = resultDF \
    .writeStream \
    .outputMode(output_mode) \
    .option("checkpointLocation", "/tmp/checkpoints_etl") \
    .foreachBatch ( \
        lambda batchDF, batchId: \
        batchDF.write \
        .format("jdbc") \
        .mode("append") \
        .option("url", f"jdbc:postgresql://{host_name}:8432/streamoutput") \
        .option("dbtable", "crime_stats") \
        .option("user", "postgres") \
        .option("password", "mysecretpassword") \
        .save() \
    ).start()
```

Wykrywanie anomalii

Do wykrywania anomalii służy dataframe `anomalies_result_df`. Wykorzystuje on utworzony wcześniej `joined_df`. Tworzone jest okno o wielkości odpowiadającej parametrowi `D` i szukane są dzielnice, w których w ciągu ostatnich `D` dni procent liczby przestępstw rejestrowanych przez FBI w stosunku do liczby wszystkich przestępstw przekroczył `P%`. Takie dane zamieniane są na zapis, w którym zamiast struktury okna widać `start_date` i `end_date`, a później wysyłane do tematu kafki `kafka-output`.

```
# Anomalie
anomalies_result_df = joined_df \
    .withColumn("date", col("date").cast("timestamp")) \
    .withColumn("primary_description", col("PRIMARY DESCRIPTION")) \
```

```
    .groupBy(window(col("date"), f"{D} days"), "district") \
    .agg(
        count("*").alias("total_crimes"),
        count(when(col("INDEX CODE") == "I", True)).alias("fbi_index_crimes")
    ) \
    .withColumn("fbi_crime_percentage", (col("fbi_index_crimes") /
col("total_crimes")) * 100) \
    .filter(col("fbi_crime_percentage") > P)

anomalies_result_df = anomalies_result_df.select(
    col("window.start").alias("start_date"),
    col("window.end").alias("end_date"),
    col("district"),
    col("fbi_index_crimes"),
    col("total_crimes"),
    col("fbi_crime_percentage")
)

# Przesłanie danych do ujścia
query = anomalies_result_df \
    .selectExpr("to_json(struct(*)) AS value") \
    .writeStream \
    .outputMode(output_mode) \
    .format("kafka") \
    .option("kafka.bootstrap.servers", f"{host_name}:9092") \
    .option("topic", "kafka-output") \
    .option("checkpointLocation", "/tmp/checkpoints_anomalies") \
    .start()
```

Program przetwarzający strumienie danych; skrypt uruchamiający

Program przetwarzający dane znajduje się w pliku `processing.py` i przyjmuje argumenty `delay`, `D` oraz `P`. Aby go uruchomić, wystarczy uruchomić skrypt `processing.sh`, który przenosi statyczny plik z danymi do `HDFS` i uruchamia przetwarzanie. Po chwili terminal zasypywany jest logami Sparka, a program jest gotowy na przyjęcie dawki danych.

Miejsce utrzymywania obrazów czasu rzeczywistego – skrypt tworzący

Miejsce utrzymywania obrazów czasu rzeczywistego przygotowywane jest w skrypcie `setup.sh`. Jest to baza danych w `PostreSQL`.

Miejsce utrzymywania obrazów czasu rzeczywistego – cechy

Jako miejsce utrzymywania obrazu czasu rzeczywistego wybrałem tabelę w bazie danych `PostreSQL` w kontenerze `Docker`. To rozwiązanie ma dużo zalet:

- wykorzystanie kontenera zapewnia izolację bazy danych od innych aplikacji i środowisk na maszynie,
- rozwiązanie to jest bardzo elastyczne, łatwo się je konfiguruje i szybko uruchamia,
- PostreSQL jest bardzo popularnym narzędziem zintegrowanym z wieloma innymi narzędziami. Posiada dużą, ciągle rozwijającą się społeczność,
- obsługa bazy oraz kontenera są łatwe, co minimalizuje ryzyko pomyłki.

Konsument: skrypt odczytujący wyniki przetwarzania

Aby odczytać wyniki przetwarzania ETL, można uruchomić skrypt `read_results.sh`. Będzie on w pętli, co 5 sekund, wykonywał zapytanie `select *` do bazy, w której zapisywane są wyniki.

Aby otrzymywać informacje o anomaliach, można uruchomić skrypt `anomalies.sh`, który łączy się z tematem Kafki `kafka-output` i wyświetla na konsolę każdą wykrytą anomalię.

Skrypty należy uruchamiać w osobnych terminalach, ponieważ po uruchamieniu skryptu, będzie on odczytywał wyniki aż do zakończenia go przez użytkownika.