SPRAWOZDANIE Z PROJEKTU SYSTEMU PREDYKCJI OPÓŹNIEŃ LOTNICZYCH

Temat: Flight Delay Prediction System

Autorzy: Jędrzej Małaczyński – 245871, Michał Urbaniak – 245954

1. ZAŁOŻENIA PROJEKTOWE

1.1 Wstęp

Celem projektu było stworzenie funkcjonalnego, rozproszonego systemu przewidywania opóźnień lotniczych, integrującego technologie webowe (Flask) z systemami Big Data (Apache Spark) oraz kolejkowaniem wiadomości (Apache Kafka). W ramach projektu nie skupiano się na trwałości danych, lecz na architekturze asynchronicznego przetwarzania potokowego. System realizuje zadania analityczne w czasie zbliżonym do rzeczywistego, obsługując zapytania użytkownika bez blokowania interfejsu.

1.2 Założenia projektowe

- **Obsługa wielu modeli predykcyjnych:** System implementuje dwa tryby działania:
    - *Model A (Planowanie):* Oparty na algorytmie Gradient Boosted Trees (GBT), wykorzystywany przed podróżą (brak danych o bieżącym opóźnieniu).
    - *Model B (Operacyjny):* Oparty na Regresji Liniowej, uwzględniający dane live (opóźnienie wylotu, czas kołowania).
- Walidacja i kompletność danych: Frontend weryfikuje poprawność logiczną (np. lotnisko wylotu różne od przylotu) oraz automatycznie uzupełnia brakujące parametry (np. dystans na podstawie słownika historycznego).
- **Asynchroniczność i skalowalność:** Wykorzystanie Apache Kafka jako bufora (kolejki) w celu ochrony silnika obliczeniowego przed nagłym skokiem ruchu (traffic spikes).
- **Bezstanowość warstwy obliczeniowej:** Apache Spark przetwarza zadania niezależnie, nie przechowując stanu sesji użytkownika.

1.3 Parametry czasowe i mechanizm Pollingu

W architekturze zrezygnowano z synchronicznego oczekiwania na odpowiedź HTTP na rzecz mechanizmu odpytywania (Polling):

- Frontend wysyła zapytanie i otrzymuje natychmiastowe potwierdzenie z request_id.
- Klient odpytuje serwer co **1s** o status zadania.
- Zastosowano tymczasowy magazyn wyników w pamięci RAM z automatycznym usuwaniem danych po ich odebraniu przez klienta.

2. STRUKTURA SYSTEMU

2.1 Architektura systemu

System Flight Delay Prediction został zaprojektowany jako aplikacja wielowarstwowa składająca się z:

1. **Flask:** Pełni rolę serwera operacyjnego i producenta wiadomości. Odpowiada za walidację, komunikację z klientem i "tłumaczenie" zapytań HTTP na format Kafki.
2. **Apache Kafka:** Pełni rolę bufora między szybkim frontendem a ciężkim silnikiem obliczeniowym.
3. **Apache Spark:** Silnik wnioskujący, odpowiedzialny za ładowanie modeli ML i wykonywanie transformacji danych.
4. **Frontend (HTML/JS):** Warstwa prezentacji realizująca logikę biznesową po stronie klienta (obliczanie czasu przylotu, wizualizacja ryzyka).

3. KOMUNIKACJA I PRZEPŁYW DANYCH (ZAPYTANIA AD HOC)

a. Dostęp Frontend → Flask (HTTP)

Zaimplementowano REST API obsługujące dwa główne kanały komunikacji:

- /predict-kafka (POST): Asynchroniczne zlecenie predykcji.
- /result/<id> (GET): Pobieranie wyniku z tymczasowego magazynu.
- /distance (GET): Ad-hoc zapytanie o parametry statyczne trasy (słownik odległości).

b. Flask → Kafka (Producer)

Utworzono mechanizm serializacji danych JSON do formatu binarnego. Flask działa jako KafkaProducer, wysyłając komunikaty do tematu flight-predictions. Zapewnia to separację warstwy webowej od analitycznej.

c. Kafka → Spark (Consumer)

Wątek tła w aplikacji uruchamia KafkaConsumer, który nasłuchuje na nowe zadania w trybie auto_offset_reset='latest', co zapewnia przetwarzanie tylko bieżących zgłoszeń.

4. INTEGRACJA MODELI

a. Statystyki Historyczne

Zamiast korzystać z zewnętrznej bazy analitycznej w czasie rzeczywistym, system ładuje przy starcie wstępnie przeliczone agregaty (plik aggregated_stats.pkl):

- Średnie opóźnienia dla linii lotniczych (Carrier Stats).
- Statystyki godzinowe i trasowe. Rozwiązanie to symuluje "cache" analityczny, drastycznie przyspieszając czas predykcji.

b. Modele Machine Learning

System integruje dwa potoki (Pipelines) PySpark MLlib:

1. model_a_gbt: Złożony model drzewiasty dla predykcji długoterminowych.
2. model_b_lr: Lekki model regresji dla korekty operacyjnej. Modele są ładowane do pamięci RAM przy inicjalizacji SparkSession, co eliminuje narzut wejścia/wyjścia (I/O) przy każdym zapytaniu.

5. PROCESY PRZETWARZANIA

a. Cykl życia zapytania

Zaimplementowano pełny obieg informacji symulujący transakcję rozproszoną:

1. Nadanie unikalnego identyfikatora UUID.
2. Kolejkowanie w Kafce (status: queued).
3. Przetworzenie przez Spark (status: processing -> ready).
4. Odbiór i usunięcie z pamięci (Atomowość odczytu).

b.

Obsługa błędów

Wdrożono mechanizmy zabezpieczające (Try-Catch) na poziomie Consumera. W przypadku błędu predykcji (np. uszkodzony model), wątek roboczy nie ulega awarii, lecz loguje błąd, a frontend po przekroczeniu czasu oczekiwania (timeout) informuje użytkownika o problemie.

6. WNIOSKI KOŃCOWE

Projekt potwierdził skuteczność architektury opartej na **Apache Kafka** jako medium integrującym lekkie interfejsy webowe z ciężkimi silnikami Big Data (**Spark**). Zastosowanie mechanizmu **polling** oraz buforowania wyników w pamięci RAM pozwoliło

na osiągnięcie wysokiej responsywności systemu przy jednoczesnym zachowaniu możliwości wykonywania złożonych obliczeń matematycznych w tle.