Large Scale Data Management
Athens University of Economics and Business
2^{nd} homework

Theodoros Anagnos, p3352323

April 1, 2024

MSc in Data Science (PT)

Part I

```python
import json
import asyncio
import random
import csv
from datetime import datetime
from aiokafka import AIOKafkaProducer
from faker import Faker

def load_spotify_tracks(file_path):
    """Load Spotify track names from a CSV file."""
    with open(file_path, newline='', encoding='utf-8') as csvfile:
        return [row['name'] for row in csv.DictReader(csvfile)]

# Initialize the Faker library for generating fake names
fake_data_generator = Faker()

# Specify the Kafka topic for song streaming data
kafka_topic = 'song_stream'

async def stream_song_data():
    """Asynchronously produce song streaming data to a Kafka topic.
    """
    producer = AIOKafkaProducer(
        bootstrap_servers='localhost:29092',
        value_serializer=lambda v: json.dumps(v).encode('utf-8')
    )
    await producer.start()

    # Generate a list of random names, including a specific name
    user_names = [fake_data_generator.name() for _ in range(15)] +
    ["Theodoros Anagnos"]

    while True:
        for user_name in user_names:
            # Select a random song from the list
            song = random.choice(tracks)
            # Generate a timestamp in the specified format
```

```
            timestamp = datetime.now().strftime('%Y-%m-%dT%H:%M:%S')
            record = {"user_name": user_name, "timestamp": timestamp, "song": song}

            # Send the data to the Kafka topic
            await producer.send_and_wait(kafka_topic, record)
            print(f"Sent record: {record}")

        # Wait for 60 seconds before sending the next batch of records
        await asyncio.sleep(60)

# Load song data from the CSV file
tracks = load_spotify_tracks('spotify-songs.csv')

# Execute the asynchronous song streaming function
# Run the producer
loop = asyncio.get_event_loop()
loop.run_until_complete(stream_song_data())
```

Part II

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, LongType, IntegerType, FloatType, StringType, TimestampType
from pyspark.sql.functions import split, from_json, col

# Define Kafka message schema
kafkaSchema = StructType([
    StructField("name", StringType(), False),
    StructField("time", StringType(), False),  # Adjusted to StringType to match the produced format
    StructField("song_name", StringType(), False)
])

# Define schema for Spotify songs data
spotifySchema = StructType([
    StructField("name", StringType(), False),
    StructField("artists", StringType(), False),
    StructField("duration_ms", LongType(), False),
    StructField("album_name", StringType(), False),
    StructField("album_release_date", StringType(), False),
    StructField("danceability", FloatType(), False),
    StructField("energy", FloatType(), False),
    StructField("key", IntegerType(), False),
    StructField("loudness", FloatType(), False),
    StructField("mode", IntegerType(), False),
    StructField("speechiness", FloatType(), False),
    StructField("acousticness", FloatType(), False),
    StructField("instrumentalness", FloatType(), False),
    StructField("liveness", FloatType(), False),
    StructField("valence", FloatType(), False),
    StructField("tempo", FloatType(), False)
])

# Initialize SparkSession
spark = SparkSession.builder.appName("SSKafka").config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0").getOrCreate()
```

```python
spark.sparkContext.setLogLevel("ERROR")

# Load Kafka stream
kafka_df = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:29092").option("subscribe", "song_stream").option("startingOffsets", "latest").load()

# Parse JSON message from Kafka
parsed_kafka_df = kafka_df.selectExpr("CAST(value AS STRING)").select(from_json(col("value"), kafkaSchema).alias("data")).select("data.*")

# Load Spotify songs data from CSV
spotify_df = spark.read.csv("spotify-songs.csv", header=True, schema=spotifySchema)

# Join Kafka stream with Spotify songs data
joined_df = parsed_kafka_df.join(spotify_df, parsed_kafka_df.song_name == spotify_df.name, "inner").drop(spotify_df.name)

# Define function to write to Cassandra
def writeToCassandra(write_df, batch_id):
    write_df.write \
        .format("org.apache.spark.sql.cassandra") \
        .options(table="song_records", keyspace="spotify") \
        .mode("append") \
        .save()

# Write joined data to Cassandra
query = joined_df.writeStream \
    .foreachBatch(writeToCassandra) \
    .outputMode("append") \
    .start()

# Await termination
query.awaitTermination()
```

```sql
SELECT * FROM song_records WHERE name = 'Theodoros Anagnos' LIMIT 50;

'''
This query filters the records by my username and the specified time range, then calculates the
average danceability value for these records.
'''
SELECT song_name
FROM song_records
WHERE name = 'Theodoros Anagnos'
  AND time >= '2024-04-01 06:00:00'
  AND time < '2024-04-01 06:20:00';

'''
This query selects the song_name for all records that match my username and fall within the specified hour.
'''
SELECT AVG(danceability) AS avg_danceability
FROM song_records
WHERE name = 'Theodoros Anagnos'
  AND time >= '2024-04-01 06:00:00'
  AND time < '2024-04-01 06:20:00';
```

Figure 1: A sample of persisted lines (50) of the Cassandra table.

Figure 2: CQL queries with otutputs.

1 Introduction

This report outlines the Cassandra data model designed for storing streaming records processed by a Spark application. The Spark application integrates live data streams from Kafka, containing user listening events, with static Spotify song data. The processed records are then stored in Cassandra for real-time analytics and query purposes.

2 Data Sources

2.1 Kafka Streams

The Kafka stream provides real-time user listening events with the following schema:

- **name**: The name of the user (StringType)
- **time**: The timestamp of the event (StringType)
- **song_name**: The name of the song being played (StringType)

2.2 Spotify Song Data

The static Spotify song dataset includes detailed metadata about each song, with the following schema:

- **name**: Song name (StringType)
- **artists**: Artists performing the song (StringType)
- **duration_ms**: Duration of the song in milliseconds (LongType)
- **album_name**: Name of the album (StringType)
- **album_release_date**: Release date of the album (StringType)
- **danceability**, **energy**, **key**, **loudness**, **mode**, **speechiness**, **acousticness**, **instrumentalness**, **liveness**, **valence**, **tempo**: Various musical features of the song (FloatType or IntegerType)

3 Cassandra Data Model

The Cassandra table *song_records* in the keyspace *spotify* is designed to store the joined records from Kafka streams and Spotify song data. The table schema is designed as follows:

```
CREATE TABLE spotify.song_records (
    name TEXT,
    time TIMESTAMP,
    song_name TEXT,
    artist_names TEXT,
    song_duration_ms BIGINT,
    album_name TEXT,
    album_release_date TEXT,
    danceability FLOAT,
    energy FLOAT,
    ...
    PRIMARY KEY ((name), time)
);
```

Primary Key Design: The combination of *name* and *time* is used as the primary key to uniquely identify each listening event. This choice supports queries that retrieve all listening events for a user in a specific time range, optimising the table for user-centric analytics.

4 Conclusion

The Cassandra data model is optimised for efficient storage and retrieval of user listening events, combined with rich song metadata from Spotify. This model supports a wide range of queries necessary for real-time music streaming analytics, such as user listening habits, song popularity trends, and more.