COEN 242 Big Data

Programming Assignment 3

Project Report on

Log Analytics using Apache Spark

Project Group:

1. Sanskriti Patole
2. Vaibhav Sachdeva

Introduction

In the age of digital transformation, we are experiencing an unprecedented surge in data generation. Big Data, characterized by its massive volume, rapid velocity, and diverse variety, has emerged as a catalyst for innovation and informed decision-making. To navigate this data-driven landscape, organizations are embracing Apache Spark, an advanced and scalable framework that empowers them to process, analyze, and derive valuable insights from vast datasets.

Apache Spark is an open-source cluster computing framework specifically designed for high-speed and scalable data processing. It provides a unified platform that caters to various data processing tasks, ranging from batch processing and real-time streaming to machine learning and graph analysis. Spark's unique capability to distribute data and computations across a cluster of machines enables it to efficiently handle the demanding workloads of Big Data.

Apache Kafka, on the other hand, has established itself as a leading distributed messaging system for handling big data processing and real-time streaming. It offers organizations a scalable solution to effectively manage high data volumes during data ingestion, processing, and delivery across diverse industries. Kafka's distributed architecture guarantees fault tolerance and high availability, allowing organizations to expand their infrastructure by adding more brokers as data loads increase. Its real-time streaming capabilities enable prompt decision-making and immediate action by processing data as it continuously flows in. Moreover, Kafka's ability to retain and replay data streams proves invaluable for purposes such as debugging, auditing, and compliance. Additionally, Kafka seamlessly integrates with other powerful Apache tools, including Spark, Storm, and Flink, enabling the creation of robust end-to-end data processing pipelines.

Problem Statement

This project consists of two primary components, each focusing on a unique aspect of data processing and analytics. The first part involves tackling two subproblems. The initial subproblem revolves around identifying the 100 most commonly occurring words within a large 16GB dataset. The second subproblem is dedicated to finding the 100 most frequent words in the same dataset, but with the additional constraint of considering only words that have more than 6 characters. In both cases, we exclude stop words from the analysis to ensure greater relevance. By leveraging the power of Apache Spark, we efficiently handle the massive dataset and showcase its prowess in extracting meaningful information from large datasets.

In the second phase of the project, we delve into log analytics using the NASA Log Analytics dataset, which consists of web server log files from the NASA Kennedy Space Center spanning a two-month period. Our aim is to analyze this dataset by leveraging a combination of powerful big data technologies, such as Kafka, Spark Streaming, Parquet files, and the HDFS file system. The main objective is to establish a robust and efficient data processing pipeline that starts with the log files. This involves using Kafka both as a producer and consumer, employing Spark Streaming for data transformations, and ultimately storing the processed data in Parquet format on the HDFS. This section not only showcases the seamless integration of these technologies but also emphasizes the smooth flow of data throughout the entire pipeline. The second phase typically has 3 sections in it. The first section requires us to get familiar with log analytics, understand and implement log analytics using Apache Spark from a given article. Further, we are required to generate an output presenting the endpoint that received the highest number of invocations on a specific day of the week, along with corresponding count of invocations. Also, we have to provide the top 10 years in which there are least 404 status codes. The second section requires us to process and analyze log files by establishing a flow that begins with a Kafka producer, followed by a Kafka consumer that performs transformations using Spark. The transformed data should then be converted into Parquet file format and stored in the HDFS. The third section requires to develop a clustering algorithm capable of grouping requests based on several factors, including the host that invoked it, the time at which the endpoint was accessed, the status code received, and the data size of the returned information.

Part 1 - Solution Approach

(Top 100 Most Popular Words using Apache Spark)

For both the subproblems of Part 1, we have used Spark to find the 100 Most Popular Words in the given 16gb dataset. Even though there are multiple approaches that can be followed for processing a 16gb data file, we have chosen to work with the RDD-based approach. The reasons for choosing the RDD-based approach over other approaches lies in its flexibility and control, familiarity (easier for developers to understand), ability to work with complex data types, and enhanced performance in certain scenarios.

RDD-based Approach:

The RDD-based approach in Spark involves creating a Resilient Distributed Dataset (RDD) from the given dataset and performing distributed processing on it. RDDs are immutable collections of objects that can be processed in parallel across a cluster. The approach includes reading the dataset into an RDD, applying transformations to manipulate the data, executing actions to trigger computations, optionally caching intermediate results for performance optimization, and leveraging Spark's distributed execution engine to process the data in parallel. The RDD-based approach offers flexibility, control, and scalability for efficiently processing large datasets.

Results for Part1 Solution Approach

We were successfully able to extract the top 100 most frequent words (having variable length and having length greater than 6 characters) from the given 16gb dataset. The output screenshots can be found in the submitted zip file.

The total execution time for finding the top 100 most frequent words was **862s** and the execution time for finding the top 100 most frequent words of length greater than 6 was **548s**.

Part 2 - Solution Approach

(Exploring NASA Log Analytics)

Part 2 - Section 1 - Analyzing web server logs at scale using Apache Spark

We start by getting familiar with analyzing log analytics using Spark by running and understanding the article given to us - (https://towardsdatascience.com/scalable-log-analytics-with-apache-spark-a-comprehensive-case-study-2be3eb3be977).

In the next step, we generate an output presenting the endpoint that received the highest number of invocations on a specific day of the week, along with corresponding count of invocations. The following were the steps that we followed to achieve the same -

- To determine the endpoint that has the highest number of invocations on each day of the week, the code begins by selecting the necessary columns ('host', 'method', 'endpoint', 'protocol', 'status', 'content_size', 'time', 'Year', and 'Day in a Week') from the logs_df DataFrame.

- The 'Year' column is generated using the F.year function, and the 'Day in a Week' column is derived using the date_format function.

- These columns are then used to create a new DataFrame called new_logs. The data is then grouped by 'Day in a Week' and 'endpoint' columns, and the count of occurrences is calculated to form the unique_Year DataFrame.

- This DataFrame is sorted in descending order based on the count. The next step involves grouping the DataFrame by 'Day in a Week' and aggregating the maximum count and the corresponding endpoint using the agg function.

- The resulting DataFrame is once again sorted in descending order based on the maximum count.

- Finally, the top 7 rows are displayed, representing the days of the week along with the endpoint that received the highest number of invocations and their respective counts.

The next step was to find the top 10 years that had the least number of 404 status codes. The following steps were followed to achieve the same -

- To identify the top 10 years with the fewest occurrences of 404 status codes, the code first creates a new DataFrame called error_404_logs by filtering the new_logs DataFrame to retain only the rows where the 'status' column is equal to 404. This ensures that only the logs with the 404 status codes are included.

- Then, the DataFrame least_error_404_df is generated by grouping the filtered DataFrame by the 'Year' column and counting the occurrences. This provides the count of 404 status codes for each year.

- Finally, the resulting DataFrame is displayed, presenting the top 10 years with the lowest number of 404 status codes and their corresponding counts.

Part 2 - Section 2 - Log data analysis using Kafka, Spark, HDFS and Parquet Files

In this part of section 2, we explore a comprehensive pipeline that efficiently processes and analyzes log files by leveraging the power of Kafka, Spark Streaming, Parquet files, and the HDFS File System. Our main goal was to establish a seamless flow that takes raw log files, applies Spark transformations, and securely stores the transformed data in the highly scalable and fault-tolerant Hadoop Distributed File System (HDFS) in the form of Parquet files.

To kick off the process, we utilize a Kafka producer as the data source and a Kafka consumer that performs transformations using Spark. The producer injects the log data, line by line, into a designated Kafka topic, to which the consumer is subscribed.

Once the log data is available in the Kafka topic, the Kafka consumer regularly checks for new data, fetching it at 10-second intervals. Acting as the entry point for Spark, this consumer enables us to tap into the robust data processing capabilities of Spark.

Using Spark, we unleash a plethora of transformations on the log data, facilitating exploratory data analysis (EDA) and yielding valuable insights. Spark's flexible and efficient framework empowers us to process vast volumes of data in a distributed and parallelized manner.

Upon completing the desired transformations and analyses, the results, represented as dataframes or RDDs (Resilient Distributed Datasets) from the final transformation steps, are seamlessly converted into Parquet files. Parquet, being a highly optimized columnar storage file format, ensures exceptional query performance and efficient data compression.

Ultimately, the Parquet files encapsulating the transformed and analyzed log data are securely stored in the HDFS. The HDFS, purpose-built for distributing and persisting extensive data across a cluster of machines, guarantees fault tolerance and high availability.

The following were the steps that we followed to achieve the same -

- Import the necessary libraries for the code.
- Create a SparkSession, which serves as the entry point for interacting with Spark.
- Set up a Kafka consumer by providing the topic, bootstrap servers, and consumer group ID.
- Initialize variables to store key-value pairs from Kafka and a counter to track iterations.
- Continuously process incoming messages from Kafka. If messages are received, extract the key-value pairs, increment the counter, and store them. If no messages are received, print a "No data" message.
- Create a DataFrame from the collected key-value pairs by defining a schema and applying it to the data.
- Preprocess the DataFrame by using regular expressions to extract specific columns from the value column. Assign appropriate aliases to the extracted columns and create a new DataFrame, logs_df, with the extracted columns.
- Parse the timestamp column using a user-defined function (UDF) to convert it into a datetime object. Assign the parsed timestamp to a new column called "time" and drop the original timestamp column.
- Display the first 10 rows of the logs_df DataFrame and print the count of rows and columns.
- Save the DataFrame as a Parquet file with a unique output path based on the iteration count.
- Copy the Parquet file from the local file system to HDFS using a command-line command executed through os.system. The destination path in HDFS includes the iteration count.
- Calculate the time taken for each iteration of processing and printing messages.
- Break the loop to terminate the program after the first iteration.

- Close the connection to the Kafka consumer.

By following this pipeline, log data can be efficiently processed, transformed, and stored in Parquet format in HDFS for further analysis and storage.

Part 2 - Section 3 - Clustering

In this section we had to apply clustering on a log dataset for which we used ‘NASA Website Data-Server logs from August 1995’. The dataset contains information about requests, including the requesting host with 75060 unique values, datetime from 31 July 1995 to 31st August 1995, request like ‘GET /images/NASA-logosmall.gif HTTP/1.0’, status code from 200 to 501, and response size.

To develop an algorithm for clustering, we had to clean the dataset 'nasa_aug95_c.csv'. After cleaning the dataset, it was saved as 'newdata.csv'.

The developed algorithm begins by loading the dataset and selecting the relevant features for clustering. The datetime feature is preprocessed by converting it to a Unix timestamp. The numerical features (status code and response size) are scaled using Min-Max normalization. Next, K-means clustering is applied with a specified number of clusters (k=3 in this case). The 'datetime', 'status', and 'response_size' features are used for clustering. The resulting cluster assignments are stored in the 'cluster' column of the DataFrame. Finally, the cluster assignments and cluster statistics (number of instances in each cluster) are printed.

The output file can be found in the submitted zip file.

Conclusion

This project focused on two components: identifying the most common and frequent words in a large dataset using Apache Spark, and analyzing the NASA Log Analytics dataset using Kafka, Spark Streaming, Parquet files, and HDFS. We successfully extracted meaningful information and developed a robust data processing pipeline. The project highlighted the seamless integration of technologies and their effectiveness in handling big data. Overall, we gained valuable insights into data processing, analytics, and the implementation of these technologies in real-world scenarios.