Experiment No.6
Data Visualization using Hive.
Date of Performance:
Date of Submission:

Aim: Data Visualization using Hive.

Theory: Hive has a fascinating history related to the world's largest social networking site: Facebook. Facebook adopted the Hadoop framework to manage their big data. If you have read our previous blogs, you would know that big data is nothing but massive amounts of data that cannot be stored, processed, and analyzed by traditional systems. Architecture of Hive The architecture of the Hive is as shown below. We start with the Hive client, who could be the programmer who is proficient in SQL, to look up the data that is needed.

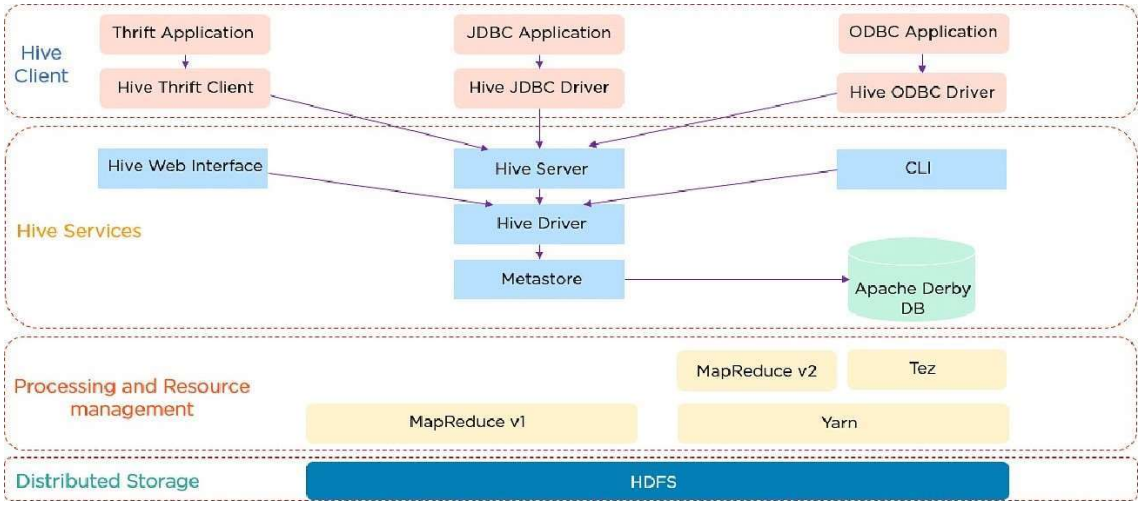

Fig: Architecture of Hive The Hive client supports different types of client applications in different languages to perform queries. Thrift is a software framework. The Hive Server is based on Thrift, so it can serve requests from all of the programming languages that support Thrift.

The data flow in the following sequence:

1. We execute a query, which goes into the driver

2. Then the driver asks for the plan, which refers to the query execution

3. After this, the compiler gets the metadata from the metastore

4. The metastore responds with the metadata

5. The compiler gathers this information and sends the plan back to the driver

6. Now, the driver sends the execution plan to the execution engine

7. The execution engine acts as a bridge between the Hive and Hadoop to process the query

8. In addition to this, the execution engine also communicates bidirectionally with the metastore to perform various operations, such as create and drop tables

9. Finally, we have a bidirectional communication to fetch and send results back to the client

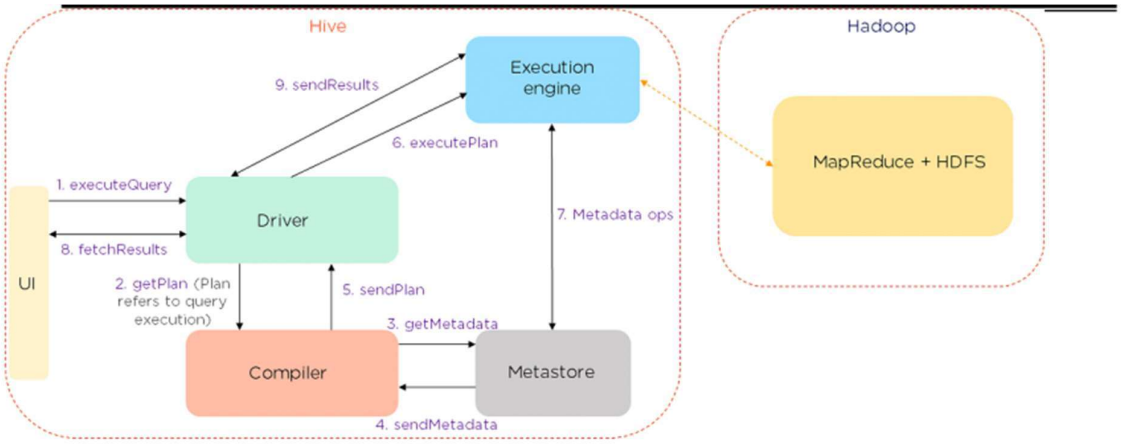

Fig: Data flow in Hive

Pesudocode:

Perform word count analysis on a large dataset using Apache Spark.

1. **Input:**
   - Large text dataset (e.g., collection of articles or documents)
2. **Steps:**
   - Initialize Spark session.
   - Load the text data into an RDD.
   - Split each line of text into words using .flatMap().
   - Map each word to a tuple (word, 1) using .map().
   - Use .reduceByKey() to sum the counts of each word.
   - Sort the words by count in descending order.
3. **Code:**

```
from pyspark import SparkContext


sc = SparkContext("local", "Word Count")
text_file = sc.textFile("data.txt")


# Split each line into words and count them
word_counts = text_file.flatMap(lambda line: line.split()) \
```

```
                .map(lambda word: (word, 1)) \
                .reduceByKey(lambda a, b: a + b)

    # Sort the words by count
    sorted_word_counts = word_counts.sortBy(lambda x: x[1], ascending=False)

    # Collect the results
    results = sorted_word_counts.collect()
    for word, count in results:
        print(f"{word}: {count}")
```

Output:

A list of words with their corresponding count:

Spark: 1000

Big: 850

Data: 800

Processing: 700

...

Conclusion:

This experiment showcases how Apache Spark efficiently processes and analyzes large datasets in parallel. The word count is a common task that highlights Spark's ability to distribute the workload across multiple nodes, drastically reducing processing time for massive text data.