1 Introduction

Uber is a transportation firm with a mobile app that enables customers to request rides and drivers to collect payments for their services. Uber is a ridesharing company that specifically employs independent contractors as drivers. It is one of many services available today that support the sharing economy by offering a way to connect available resources rather than providing the actual resources themselves. The business, which has its headquarters in San Francisco, was established in 2009 by Travis Kalanick and Garrett Camp. Worldwide, the corporation is thought to have 110 million subscribers.

Problem statements(s): In many of the nations it operates, Uber dominates the ride-hailing market. Other regional goliaths have, however, fiercely competed against the corporation. In Asia, Grab, Ola and Didi-Chuxing have given it a run for its money, while Bolt and Yango are strong rivals in Europe and Africa.

Since the ecosystem of the ride-hailing sector is primarily dependent on “Drivers”, who are independent contractors who provide vehicles and services to the consumer market, In the trucking industry, there is a big problem called "driver churn," which happens when drivers resign and start working for a competitor company or in another sector. By analysing the dataset, we would be able to identify these drivers and investigate their income to determine how many of these resignations have happened because of income matters.

2 Methodology

The original dataset for this study was obtained from Kaggle (2022). This dataset contains 200,000 records of uber trips made in October 2016. According to the location information, we assume the data is extracted for New York City. The size of the csv file is 16.022MB.

Feature	**Data Type**	**Description**
id	int	index
key	string	a unique identifier for each trip
driver_id	int	The id of each driver
fare_amount	float	The fare amount of each trip
pickup_datetime	datetime	The date and time when pick up the customer
pickup_longitude	float	The longitude of the pickup location
pickup_latitude	float	The latitude of the pickup location
dropoff_longitude	float	The longitude of the dropoff longitude
dropoff_latitude	float	The latitude of the dropoff location
passenger	int	The number of passengers in the trip
status	string	The status of the driver whether he/she has resigned

Table 1: Dataset Information

A few selected Hadoop big data tools will be used on top of Hadoop File System (HDFS). Apart from that, we will also be using MongoDB as a comparison to Hadoop technologies. Here are some introductions to the tools used in this project.

2.1 Hadoop, MapReduce, and HDFS

Hadoop is a suite of open-source software tools that solve big data issues by using a computer network. It uses the MapReduce programming style and provides a software framework for distributed computing, processing, and storing large amounts of data. The Hadoop Distributed File System (HDFS) is the storage component of Apache Hadoop and divides files into large blocks for distribution to nodes for parallel processing. MapReduce is a fundamental component of Hadoop; it converts data into key-value pairs and then joins the data into smaller sets using the output of the map task as its input.

HDFS is a subproject of Apache Hadoop, designed to work on standard hardware with excellent fault tolerance and enables streaming access to file system data. HDFS has limitations such as only allowing append actions and not supporting file modification or simultaneous writes.

2.2 HBase

An open-source project called HBase is based on the HDFS and has horizontal scalability. In order to give quick access to massive and organised data, the HBase database is created similarly to Google's big data table and distributed column-wise. It also supports random real-time read, write, and access in the HDFS system as part of the Hadoop ecosystem (Tutorialspoint, 2023). HBase can aggregate databases with billions of rows and analyse them; the database can also be shared. For online analytical procedures like real-time data analysis of financial data and stock markets, reading and processing take very little time and are frequently employed. Incompatibility with some characteristics of conventional database models is a drawback. HBase lacks a specific query language, and MapReduce requires a lot of CPU and memory resources. Additionally, integrated Hive and Pig jobs on HBase have latency and time memory concerns with MapReduce processes (Taylor, 2022).

2.3 Hive

A data warehouse programme called Hive makes it easier to read, write, and manage big datasets kept in HDFS and other data storage systems. For data processing and querying, it offers the Hive Query Language, a programming language that is similar to SQL. It also has the added benefit of making MapReduce programming querying even more straightforward by mapping queries with SQL-like instructions and then reducing functions (IBM, 2021). A big collection of unstructured data can be given a table structure thanks to Hive's out-of-the-box installation of the Hive metastore that comes with Hive installation (IBM, 2021).

2.4 Pig

Using the Apache Hadoop framework, Apache Pig is helpful for analysing huge datasets. By using MapReduce to speed up the query, it provides data analysis inquiries. Thus, the query's time complexity is reduced, which is advantageous (IBM, 2021). However, the queries are written in a high-level language called Pig, which is a disadvantage as it would necessitate learning a new language (IBM, 2021).

2.5 MongoDB

Document-oriented and schema-less MongoDB offers enormous data storage capabilities. It makes use of documents that resemble JSON for storing purposes. These documents are nested documents and key/value arrays. It offers a wide range of data types and supports several application-side programming languages to execute the queries. Additionally, MongoDB supports ad-hoc querying using the MongoDB Query Language. Additionally, each transaction's ACID attributes are included (GeeksforGeeks, 2022).

2.6 PySpark

For massive processing over one or more clusters, Spark offers a unified engine. By limiting shared memory across distributions, Spark serves as a working set for distributed querying. As a result, it gets around MapReduce's restriction on linear data flow. Spark can run independently or in conjunction with the Hadoop framework. Furthermore, it offers native bindings for a variety of programming languages. Spark's processing is iterative by nature, which is a severe drawback. (Pointer, 2020)

In this study, the data storage tools HDFS, HBase, and MongoDB are compared to the data processing tools Hive, Pig, MongoDB, and Spark.

3 Environment Setup

The analysis in this study is conducted using Cloudera Virtual Machine on top of MacBook M1 Pro with the following configurations. Table 2 lists the hardware configuration.

Configuration	Description
Processor	M1 Pro
RAM	16 GB
Operating System	MacOS 13.1

Table 2: Hardware Configurations for This Study

The details of the virtual machine configurations are listed in Table 3 as follows:

Configuration	Description
Operating System	Cloudera
RAM	12 GB
Disk	80 GB
vCPUs	10

Table 3: Cloudera Virtual Machine Configurations

4 Data Analytics Pipeline

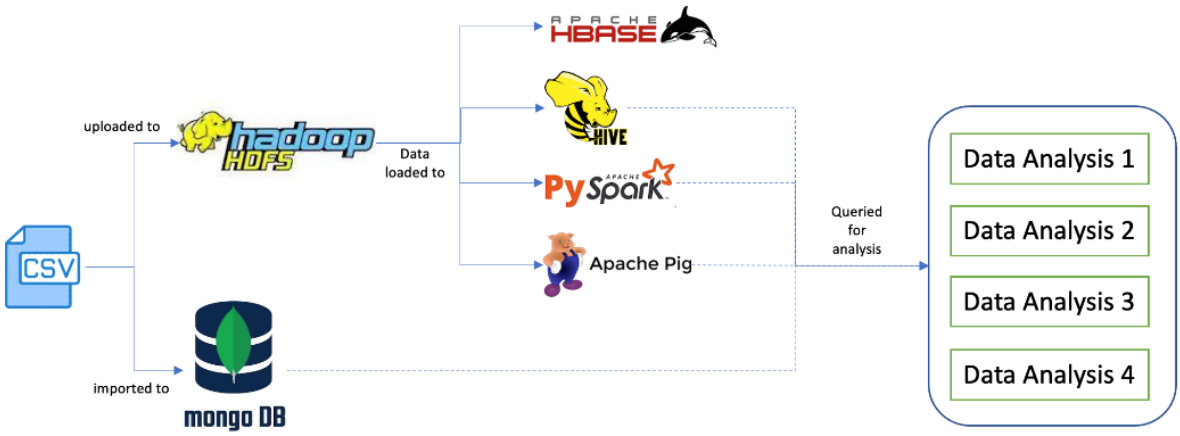

Figure 1: Data Analytics Pipeline for this Project

Figure 1 is the data analytics pipeline of our project. First, we uploaded the dataset which is in csv format to HDFS directory for the other big data tools to access. Secondly, we imported it to MongoDB as well to test the NoSQL tool against Hadoop technologies. The file from HDFS will then be accessed by four different tools from Hadoop ecosystem: HBase, Hive, Spark and Pig. data into all the 5 tools, but we found it quite challenging to write Java programs to perform analysis in HBase. Hence, we use Hive, Spark and Pig together with MongoDB to perform 4 different data analysis to get the driver churn.

5 Big Data Management

5.1 Big Data Storage

One of the main challenges Uber faces is data management; this includes the storage and processing of data. In this section, the Uber dataset was loaded in different big data storage, specifically HDFS, MongoDB and HBase. The time taken to load the dataset forms the comparison metric between these data storage tools. The following sections show the commands executed in each data storage tool.

5.1.1 HDFS

```
sudo -u hdfs hadoop fs -copyFromLocal /home/cloudera/Documents/Uber_Driver_Churn.csv /hdfs/uber/
```

5.1.2 HBase

```
Create Table
hbase > create 'Uber_DB', 'id', 'key', 'driver_id', 'fare_amount', 'pickup_datetime', 'pickup_longitude',
'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'status'
Import Dataset
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator="," -
Dimporttsv.columns="HBASE_ROW_KEY,id, key, driver_id, fare_amount, pickup_datetime,
pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, passenger_count, status"
Uber_DB /hdfs/uber/ Uber_Driver_Churn.csv
```

5.1.3 MongoDB

```
Create Database
Use Uber_DB
Create Collection
Db.createCollection("uberdriverchurn")
Import Dataset
mongoimport --db Uber_DB --collection uberdriverchurn --type=csv --
file=[/hdfs/uber/Uber_Driver_Churn.csv] --headerline
```

5.1.4 Big Data Storage Results

Big Data Storage	Time to Load Dataset (s)
HDFS	4.51
HBase	7.05
MongoDB	2.157

Table 4: Time to Load Uber Driver Churn Data by Storage Tools

Table 4 shows the total time taken to load and export Uber data to different storage tools. It is observed that MongoDB had the shortest loading time since it is better at handling memory as compared to HDFS and HBase. However, this can only be applied in this dataset as the dataset used only contains 200,000 records of data.

5.2 Big Data Processing

The numerous operations on the data are what the data processing tools are concerned with. Hive, Pig, and Spark are the three data processing platforms we chose for this investigation. The measurement metric employed is the amount of time it takes for data to load in data processing/querying tools. The commands used in each tool to load the Uber Driver Churn Data are displayed in the sections below.

5.2.1 Hive

Hive needs a table to be created before it can process data. An external table is made in Hive to query data and process the dataset, enabling data processing of the dataset stored in HDFS.

```
Create Database & Table:
hive> create database Uber_DB;
hive> Use Uber_DB;
hive> set hive.cli.print.current.db=true

hive (Uber_DB) > CREATE TABLE IF NOT EXISTS uberdriverchurn(id INT, key String,driver_id INT,
fare_amount Double,pickup_datetime String, pickup_longitude Double, pickup_latitude Double,
dropoff_longitude Double, dropoff_latitude Double, passenger_count INT, status String)
        > row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile
        > tblproperties("skip.header.line.count"="1");
Load Data:
hive (Uber_DB)> LOAD DATA LOCAL INPATH '/home/cloudera/Documents/Uber_Driver_Churn.csv'
OVERWRITE INTO TABLE uberdriverchurn;
```

5.2.2 Pig

```
uberdriverchurn = load '/user/hdfs/uberproject/Uber_Driver_Churn.csv' using PigStorage(',') as (id: int,
key: chararray, driver_id: int, fare_amount: float, pickup_datetime: chararray, pickup_longitude:
chararray, pickup_latitude: chararray, dropoff_longitude: chararray, dropoff_latitude: chararray,
passenger_count: int, status: chararray);
```

5.2.3 PySpark

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Uber Driver Churn").getOrCreate()
uberdriverchurn = spark.read.format("csv").option("header", "true").option("inferSchema",
"true").load("/user/hdfs/Uber_Driver_Churn.csv")
```

5.2.4 Big Data Processing Results

Table 5 shows the time to load the Uber dataset in the different data processing tools. Loading data in Hive was almost instantaneous. According to the Hive documentation (Apache Hive, 2022), when creating a table in Hive to load data, it points to external storage for the tables, so Hive is not copying the data over. Moreover, PySpark and Pig have a relatively close time to load the dataset, with Pig being slightly faster due to Pig's architecture, where the data loading can be parallelised and enable the load of large datasets. In contrast, PySpark provides batch and streaming data loading operations (EDUCBA, 2022). To clarify, the load statement in Pig will simply load the data into the specified relation from HDFS. Hence, we need to use a diagnostic operator, DUMP, to verify the data was loaded correctly in Pig. Based on the results, Hive took the shortest time to load the dataset. However, it is still insufficient to conclude it is the best tool for data processing. Hence, we will perform a series of data analyses to assess which data processing tool has the shortest querying time.

Data Processing Tool	Time to Load Dataset (s)
Hive	7.1
Pig	9.9
PySpark	10.7

Table 5: Time to Load Data in Data Processing Tools

5.3 Data Analysis

5.3.1 Analysis 1: List Top 10 Drivers based on Monthly Income

Our first analysis will calculate the monthly income of the drivers and extract top 10 records from the results.

5.3.1.1 Hive

```
hive(Uber_DB) >  SELECT driver_id, sum(fare_amount) as Total_Income From uberdriverchurn
   > Group By driver_id Order By Total_Income DESC  limit 10;
```

5.3.1.2 MongoDB

```
db.uberdriverchurn.aggregate([
   { $group: { _id: "$driver_id", Total_Income: { $sum: "$fare_amount" } } },
   { $sort: { Total_Income: -1 } },
   { $limit: 10 } ])
```

5.3.1.3 PySpark

```
from pyspark.sql.functions import col,sum
uberdriverchurn_grouped = uberdriverchurn.groupBy("driver_id")
uberdriverchurn_total_income =
uberdriverchurn_grouped.agg(sum("fare_amount").alias("Total_Income"))
uberdriverchurn_total_income_sorted =
uberdriverchurn_total_income.orderBy(col("Total_Income").desc())
uberdriverchurn_top_10 = uberdriverchurn_total_income_sorted.limit(10)
uberdriverchurn_top_10.show()
```

5.3.1.4 Pig

```
uberdriverchurn_grouped = GROUP uberdriverchurn BY driver_id;
uberdriverchurn_total_income = FOREACH uberdriverchurn_grouped GENERATE group as driver_id,
SUM(uberdriverchurn.fare_amount) as Total_Income;
uberdriverchurn_total_income_sorted = ORDER uberdriverchurn_total_income BY Total_Income DESC;
uberdriverchurn_top_10 = LIMIT uberdriverchurn_total_income_sorted 10;
DUMP uberdriverchurn  top  10;
```

5.3.2 Analysis 2: Count the Total Number of Resigned Drivers

Next, we used different tools to count the total number of resigned drivers based on the dataset. The result of this analysis is **108** drivers.

5.3.2.1 Hive

```
hive (uber_db)> SELECT COUNT(*) FROM uberdriverchurn > WHERE status = 'resigned';
```

5.3.2.2 MongoDB

```
db.uberdriverchurn.countDocuments({ status: "resigned" })
```

5.3.2.3 PySpark

```
from pyspark.sql.functions import count
resigned_drivers = uberdriverchurn.filter(uberdriverchurn["status"] == 'resigned')
resigned_drivers_count = resigned_drivers.agg(count("*"))
resigned_drivers_count.show()
```

5.3.2.4 Pig

```
resigned_drivers = FILTER uberdriverchurn BY status == 'resigned';
resigned_drivers_count = FOREACH (GROUP resigned_drivers ALL) GENERATE COUNT(resigned_drivers);
DUMP resigned_drivers_count;
```

5.3.3 Analysis 3: The Total Number of Resigned Drivers with Income of less than $2350

We cut off the minimum salary to $2350, this is based on the minimum wage of United States being $7.25 per hour although it varis from state to state. To find the number of drivers resigned with an income of less than $2350. The result of this analysis is **66** drivers resigned due to low salary.

5.3.3.1 Hive

```
hive(Uber_DB)> WITH tbl_Total_Income As(
   > SELECT driver_id, sum(fare_amount) as Total_Income From uberdriverchurn
   > Group By driver_id  > Order By Total_Income DESC)
   > SELECT Distinct(U.driver_id),U.status, T.Total_Income From uberdriverchurn
   > U JOIN tbl_Total_Income T ON U.driver_id = T.driver_id
   > WHERE status ='resigned' and T.Total_Income < 2350 Order By T.Total_Income DESC;
```

5.3.3.2 MongoDB

```
db.uberdriverchurn.aggregate([
   { $group: { _id: "$driver_id", Total_Income: { $sum: "$fare_amount" } } },
   { $match: { Total_Income: { $lt: 2350 } } },
   { $sort: { Total_Income: -1 } },
   { $lookup: { from: "uberdriverchurn", localField: "_id", foreignField: "driver_id", as: "U" } },
   { $match: { "U.status": "resigned" } },
   { $sort: { Total_Income: -1 } },
   { $project: { _id: 0, driver_id: "$_id", status: "$U.status", Total_Income: "$Total_Income" }}])
```

5.3.3.3 PySpark

```
from pyspark.sql.functions import sum
uberdriverchurn_grouped = uberdriverchurn.groupBy("driver_id")
tbl_Total_Income = uberdriverchurn_grouped.agg(sum("fare_amount").alias("Total_Income"))
tbl_Total_Income_sorted = tbl_Total_Income.orderBy(col("Total_Income").desc())
resigned_drivers = uberdriverchurn.filter(uberdriverchurn["status"] == 'resigned')
resigned_drivers_with_income = resigned_drivers.join(tbl_Total_Income_sorted, "driver_id")
resigned_drivers_with_income_filtered = resigned_drivers_with_income.filter(col("Total_Income") <
2350)
resigned_drivers_with_income_filtered_sorted =
resigned_drivers_with_income_filtered.orderBy(col("Total_Income").desc())
resigned_drivers_with_income_filtered_sorted_distinct =
resigned_drivers_with_income_filtered_sorted.dropDuplicates()
resigned_drivers_with_income_filtered_sorted_distinct.show()
```

5.3.3.4 Pig

```
uberdriverchurn_grouped = GROUP uberdriverchurn BY driver_id;
tbl_Total_Income = FOREACH uberdriverchurn_grouped GENERATE group as driver_id,
SUM(uberdriverchurn.fare_amount) as Total_Income;
tbl_Total_Income_sorted = ORDER tbl_Total_Income BY Total_Income DESC;
resigned_drivers = FILTER uberdriverchurn BY status == 'resigned';
resigned_drivers_with_income = JOIN resigned_drivers BY driver_id, tbl_Total_Income_sorted BY
driver_id;
resigned_drivers_with_income_filtered = FILTER resigned_drivers_with_income BY
(tbl_Total_Income_sorted::Total_Income < 2350);
resigned_drivers_with_income_filtered_sorted = ORDER resigned_drivers_with_income_filtered BY
tbl_Total_Income_sorted::Total_Income DESC;
resigned_drivers_with_income_filtered_sorted_distinct = DISTINCT
resigned_drivers_with_income_filtered_sorted;
DUMP resigned_drivers_with_income_filtered_sorted_distinct;
```

5.3.4 Analysis 4: Total Passengers Each Driver Transported in a Month

The next analysis we want to do is to calculate the total number of passengers each driver could transport in a month.

5.3.4.1 Hive

```
hive(Uber_DB)> SELECT driver_id, SUM(passenger_count) AS Total_Passenger
            > From uberdriverchurn Group By driver_id Order By Total_Passenger DESC limit 10;
```

5.3.4.2 MongoDB

```
db.uberdriverchurn.aggregate([
   { $group: { _id: "$driver_id", Total_Passenger: { $sum: "$passenger_count" } } },
   { $sort: { Total_Passenger: -1 } },
   { $limit: 10 } ])
```

5.3.4.3 PySpark

```
from pyspark.sql.functions import sum, desc
grouped_df = uberdriverchurn.groupBy("driver_id")
total_passenger_df = grouped_df.agg(sum("passenger_count").alias("Total_Passenger"))
sorted_df = total_passenger_df.sort(desc("Total_Passenger"))
top_10_df = sorted_df.limit(10)
top_10_df.show()
```

5.3.4.4 Pig

```
uberdriverchurn_grouped = GROUP uberdriverchurn BY driver_id;
uberdriverchurn_total_passenger = FOREACH uberdriverchurn_grouped GENERATE group as driver_id,
SUM(uberdriverchurn.passenger_count) as Total_Passenger;
uberdriverchurn_total_passenger_sorted = ORDER uberdriverchurn_total_passenger BY Total_Passenger
DESC;
uberdriverchurn_top_10 = LIMIT uberdriverchurn_total_passenger_sorted 10;
DUMP uberdriverchurn_top_10;
```

5.3.5 Data Analysis Results

Analysis	Hive	Pig	MongoDB	PySpark
1	101.7	240.2	4.1	2.5
2	54.6	121.5	1.9	1.1
3	203.3	481.7	8.9	3.8
4	113.9	256.3	4.6	2.7
Average (s)	118.3	274.9	4.8	2.5

Table 6: Time to Complete Queries in Each Data Processing Tool

PySpark performs best in terms of data analysis as compared to Hive, Pig and MongoDB. This is due to Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read or write operations to disk. It stores the intermediate processing data in memory. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application in which it is designed for fast computation (Tutorialspoint, 2023). MongoDB followed closely after PySpark. MongoDB has deep query-ability which supports dynamic queries on documents using a document-based query language (Tutorialspoint, 2023) that is nearly as powerful as PySpark. Both Hive and Pig took relatively longer time to return the query as compared to MongoDB and PySpark. Hive is slightly faster than Pig it is specifically designed for data summarization, query and analysis. Pig took the longest time to execute the query because it is a procedural data flow language. It requires more time for the data flows which involved multiple inputs and transform them to outputs.

6 Conclusion

Overall, we have chosen a medium size data set that contains 200,000 uber trips and use different big data tools to solve the driver churn problem.

As for data storage, Mongo DB takes the shortest time for data storage since it is better at memory handling, while Hadoop technologies are better at space handling compared to MongoDB. Among the tools Hive performs the best in terms of data loading as for scripting, Spark performance's is the most satisfactory.

This project helps us to better understand the use of each software and understand their practical application scenarios. HBase is suitable for processing semi-structured and unstructured data whose structure may change. HDFS applies to the streaming read mode and is suitable for the scenario where data is written once, and data is read multiple times. Due to very high performance, MongoDB is very suitable for real-time insertion, update and lookup, and has a high degree of scalability. Hive can help data analysts tap into the Hadoop platform for data analysis. Pig is a data flow language for handling large data sets. Both Pig and Spark are data-streaming.

One limitation was that we did not have enough types of data, and we only used structured data for this project. Another limitation was that due to the limitation of time, equipment, and resources, we did not choose the data type applicable to the tool for analysis and multi-aspect comparison.

In the future, for data analysis, we will use Impala, a high-performance SQL engine for data analysis. For data storage, we will explore working with more types of data such as semi-structured and unstructured data. Apart from that, we will explore the application of NoSQL database such as MongoDB, HBase and Redis.