UNIVERSITY OF MALAYA

Faculty of Computer Science & Information Technology

Semester 1 Session 2023/2024

WQD7007 Big Data Management

NAME	MATRIX NUMBER
YING MING TANG	S2180377
LEE JIH SHIAN	17126198
TAN ZE YING	22058059
ELAINE LI	S2164604
XING ZHAO CHUA	S2188563
HUN YEE CHONG	S2197999
KAR HONG SAM	S2191926
WAI HONG WOO	22065374

TABLE OF CONTENT

1 INTRODUCTION

During the initial phases of the Covid-19 pandemic, Netflix experienced a surge in subscriber numbers as people around the world sought entertainment options while adhering to lockdowns and social distancing measures. The company's content library and original productions played a crucial role in retaining and attracting new subscribers during this period. However, Netflix's performance during the later stages of the pandemic had changed after the easing of lockdowns, the reopening of economies and evolving consumer behaviors. The initial months of 2023 marked a slow start for Netflix, culminating in the lowest revenue growth since 2013.

Additionally, Netflix, a pioneer in the streaming industry, has encountered formidable challenges that have significantly impacted its market standing and revenue. One of the foremost challenges is the escalating competition within the streaming domain, marked by the entry of new competitors such as Amazon Prime Video, Apple TV, HBO and Disney+. The influx of these famous and content-rich platforms has intensified the battle for viewer attention and subscription, leading to a decline in Netflix's revenue (Sadq, 2013). Moreover, Netflix is facing the challenge of tailoring its recommendation system to cater to the diverse and evolving needs of various global markets. It is actively finding approaches to fit in different cultural nuances, content preferences and viewer behaviors across regions. To overcome this challenge, Netflix requires a thorough understanding of local markets and the deployment of adaptive algorithms that deliver customized recommendations that resonate with diverse audiences (Sadq, 2013).

Hence, the study's objectives are to seek a solution to boost sales and subscriber expansion to enhance overall revenue and subscriber growth of Netflix. We also aim to utilize various tools to analyze and compare their effectiveness in addressing the first objective.

2 METHODOLOGIES

2.1 HIVE

Apache Hive is a Hadoop-based data warehouse infrastructure that allows users to manage and query large datasets in a distributed storage environment. It was created by the Apache Software Foundation and is an open-source project. Hive is intended to simplify data querying and analysis for users who are unfamiliar with complex MapReduce programming.

2.2 PIG

Apache Pig is a platform for analyzing large data sets that consists of a high language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. Pig is an application that creates MapReduce jobs based on a language called Pig Latin. Pig is best for semi structured data, for programming used as procedural languages and does not have dedicated metadata of database (Pol, U. R., 2016). It runs on Hadoop by making use of both HDFS and MapReduce.

2.3 MONGODB

Based on GeeksforGeeks (2023), MongoDB is a schema-less NoSQL database. It provides large-scale data storage where it makes use of JSON-like documents. These documents are nested documents which have key/value arrays. Besides, it allows various types of data and supports a few application-side programming languages to run the queries. To add on, MongoDB Query Language allows users to do ad-hoc querying. Moreover, it provides ACID (Atomicity, Consistency, Isolation, Durability) properties per transaction as well.

2.4 APACHE SPARK

According to the official website of Apache Spark (2023), Apache Spark is a versatile, multi-language engine designed for executing tasks in data engineering, data science, and machine learning, whether on single-node machines or distributed clusters. This tool boasts several key features, including the seamless integration of batch and real-time streaming data processing, supporting programming languages such as Python, SQL, Scala, Java, or R. It enables rapid execution of ANSI queries on a variety of data warehouses, facilitating efficient dashboard and reporting functionalities. Additionally, Apache Spark empowers users to conduct Exploratory Data Analysis (EDA) on petabyte-scale data sets without the need for down sampling. Notably, it allows the training of machine learning algorithms on a local machine and provides the flexibility to scale the same code effortlessly to fault-tolerant clusters comprising thousands of machines.

3 DATA ANALYTICS PIPELINE

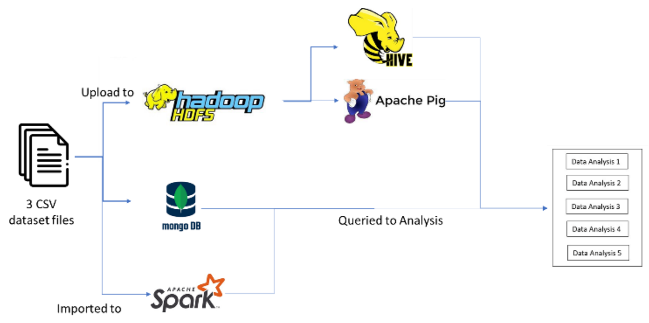


Figure 1: Data Analytics Pipeline in this project

Figure 1 shows the steps we use to analyze data in this project. First, we upload three datasets in CSV format to the HDFS directory. This is important because it lets different big data tools use the data. Then, we put the data into MongoDB, a NoSQL database, to see how it works with big data tools like Spark and Hadoop.

Next, two tools from the Hadoop ecosystem, Hive and Pig, are used to access and work with the data in HDFS. These tools are great for handling big amounts of data. The main part of our project uses Hive, Spark, and Pig, along with MongoDB, to do five different analyses. These analyses help us understand how Netflix's money and number of subscribers are changing.

This process shows us how to combine old and new ways of processing data. We use both SQL-like queries (with Hive) and scripts (with Pig), which shows that we can handle different types of data analysis. The mix of HDFS and MongoDB also shows how we can work with both organized and unorganized data. Below show six main parts which have covered in this pipeline, from Data Generation to making smart decisions (Decision Making).

Data Generation: The initial phase involves the creation of data. In our case, this pertains to the compilation of three distinct datasets relevant to our project. These datasets are prepared in CSV format, a common structure for data representation.

Data Acquisition: This step involves uploading the generated datasets to the Hadoop Distributed File System (HDFS) directory. Uploading to HDFS is a important step as it facilitates the accessibility of data by various big data tools.

Data Storage: Once in HDFS, we transfer the data into MongoDB, a NoSQL database and Sparks. MongoDB is instrumental in handling and storing large volumes of data efficiently.

Data Analysis: For the analytical part, Hive, Spark, and Pig from the Hadoop ecosystem are employed. Each tool has its strengths – Hive is used for its SQL-like querying capabilities, while Pig is utilized for its scripting process. Together with MongoDB and Spark, these tools execute five different types of analyses to deduce trends in Netflix's revenue and subscriber growth.

Data Visualization: Although not detailed in the initial description, this stage would typically involve converting our analysis results into visual formats like charts or graphs. This helps in better understanding and interpreting the data, making it easier to communicate findings.

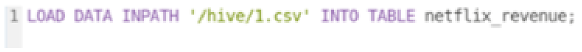
Decision Making: The final phase is where insights drawn from data visualization guide strategic decisions. Based on our analysis, decisions could be about content strategy, marketing, or customer engagement for Netflix.

4 BIG DATA MANAGEMENT

4.1 BIG DATA STORAGE

4.1.1 HIVE

Align to the design principle of the Hadoop ecosystem, the CSV file is first stored into the HDFS. After that, a database and a table were created on Hive to store the dataset for further analysis. The queries used include 'CREATE DATABASE 'database_name'' and 'CREATE TABLE 'table_name''. After that, the dataset in HDFS is loaded into the table created using the query 'LOAD DATA INPATH 'file_location' INTO TABLE 'table_name''.



```
LOAD DATA INPATH '/hive/1.csv' INTO TABLE netflix_revenue;
```

Figure 2: Query loading dataset into Hive table.

In this project, three tables had been created and loaded with 3 datasets. The three tables are 'netflix_revenue', 'netflix_title' and 'netflix_usebase'. After the table is created, the data storage is stored into the Hive and the storage time is 0s for all three tables.

4.1.2 PIG

Hadoop Pig is a high-level scripting language built on top of Hadoop and it provides a data flow language and execution environment for processing large datasets. Hadoop Distributed File System (HDFS) is the underlying storage system for Hadoop and Pig interacts with HDFS for data storage as Pig is primarily used for data processing. The huge amount of data processed by Pig can be stored back in HDFS.

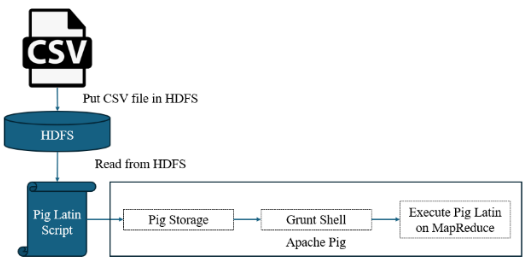


Figure 3: Apache Pig Architecture

4.1.3 MONGODB

Create MongoDB cloud server in MongoDB Atlas and configure network access by adding IP address for the cloud server. Then, connect the created database in the server through MongoDB compass, which is the graphical user interface (GUI) tool for MongoDB. It provides a user-friendly way to explore and manage your MongoDB data. Compass allows users to visualize, query, and analyze data stored in MongoDB. The dataset was uploaded to the MongoDB database by using the GUI. The process of uploading took approximately a few seconds only.

4.1.4 APACHE SPARK

There are two approaches to load the data into Apache Spark, Spark SQL and Spark Shell. For Spark SQL, start the session on terminal using spark-sql, then create database, 'netflix' then create table 'netflix_revenue' to load the dataset 'netflix_revenue'. It took about 3.36 seconds to complete. For Spark Shell, start the session with spark-shell. Assign path variable to the dataset file and then create a temporary view as table. It took about 3 seconds to complete for dataset 'netflix_title' and 1 second for the dataset 'netflix_userbase'.

4.2 BIG DATA PROCESSING

4.2.1 HIVE

The language in Hive is called Hive Query Language (HQL), which is quite similar to SQL, a popular language for dealing with databases. HQL is made for asking questions and handling data in Hive. Since HQL looks a lot like SQL, people who already know how to use SQL for relational databases will find it easy to understand. In HQL, you can use common SQL commands like SELECT, FROM, WHERE, GROUP BY, and JOIN. These commands help you search, sort, and change data. This makes it convenient for users to work with large amounts of data in Hive.

4.2.2 PIG

Pig Latin is the parallel dataflow language which is designed to fit between SQL and MapReduce. It enables the use to define the reading, processing and storing data in parallel. Pig Latin script explicates a directed acyclic graph, where data flows are represented as edges and operators are represented as nodes. 'LOAD' statement is used to load data into Pig from HDFS or local file systems. Data transformation can be applied to the loaded data using Pig Latin operations. This can include filtering, grouping, joining and other operations to process the data. The results of the processing can be saved back to HDFS using the 'STORE' statement. Pig also provides tools for debugging and testing scripts. The 'Describe', 'Dump', and 'Explain' statements help in understanding the structure of the data.

4.2.3 MONGODB

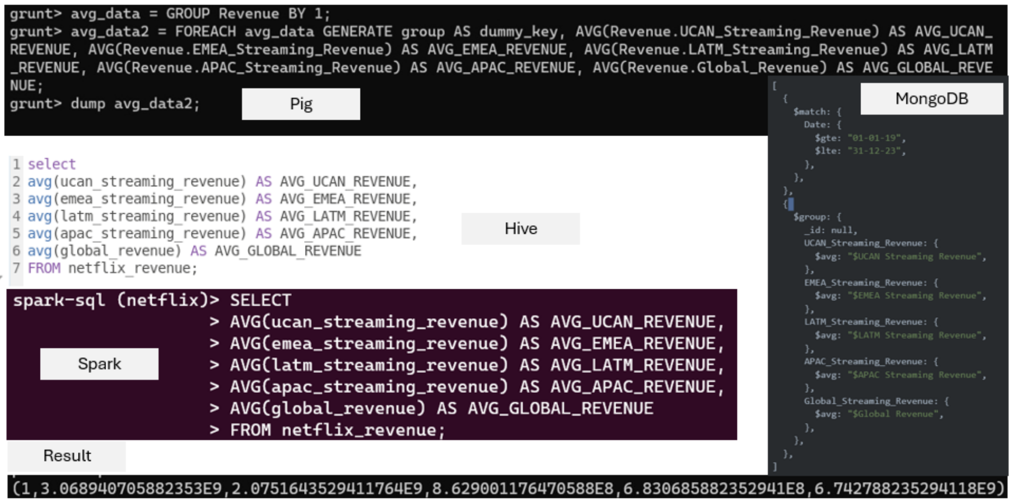
MongoDB performs data analysis primarily through an aggregation framework which operates on the concept of pipeline, where data passes through multiple stages and transforms in certain way. Each stage is defined in a JSON-like format, which allows for clear specification of the operations to be performed. The most common stages in the aggregation pipeline include $match, $group, $sort, $project, $limit, $out and more. This pipeline method in MongoDB is very flexible, allowing users to combine different stages to handle complex data queries and manipulations. For example, data can be filtered, grouped by certain criteria, sorted in a specific order, and then the results can be limited or directed to a specific output. This makes MongoDB a powerful tool for working with large datasets and extracting meaningful insights from them.

4.2.4 APACHE SPARK

Based on Databricks (2023), Spark SQL is an integral component of Apache Spark, streamlines the processing of structured and semi-structured data through SQL queries. This empowers developers to seamlessly import relational data from sources such as Parquet files and Hive tables, enabling the execution of SQL queries on both imported data and existing RDDs. Spark SQL offers added convenience by easily writing RDDs back to Hive tables or Parquet files. Noteworthy features include a cost-based optimizer, columnar storage, and code generation, ensuring swift query performance. The scalability of Spark SQL allows efficient handling of thousands of nodes and supports multi-hour queries using the Spark engine. With full mid-query fault tolerance, users need not worry about switching engines for historical data processing. Additionally, Spark Shell available in versions like spark-shell for Scala, pyspark for Python, and sparkR for R, provides an indispensable interactive command-line interface (Laskowski, 2023). This tool is essential for users engaged in interactive exploration, prototyping, and debugging of Spark applications, enhancing the efficiency of development processes.

4.3 DATA ANALYSIS

4.3.1 ANALYSIS 1: Average revenue for different regions from 2019-2023



```
grunt> avg_data = GROUP Revenue BY 1;
grunt> avg_data2 = FOREACH avg_data GENERATE group AS dummy_key, AVG(Revenue.UCAN_Streaming_Revenue) AS AVG_UCAN_
REVENUE, AVG(Revenue.EMEA_Streaming_Revenue) AS AVG_EMEA_REVENUE, AVG(Revenue.LATM_Streaming_Revenue) AS AVG_LATM
_REVENUE, AVG(Revenue.APAC_Streaming_Revenue) AS AVG_APAC_REVENUE, AVG(Revenue.Global_Revenue) AS AVG_GLOBAL_REVE
NUE;
grunt> dump avg_data2;
```

```
select
avg(ucan_streaming_revenue) AS AVG_UCAN_REVENUE,
avg(emea_streaming_revenue) AS AVG_EMEA_REVENUE,
avg(latm_streaming_revenue) AS AVG_LATM_REVENUE,
avg(apac_streaming_revenue) AS AVG_APAC_REVENUE,
avg(global_revenue) AS AVG_GLOBAL_REVENUE
FROM netflix_revenue;
```

```
spark-sql (netflix)> SELECT
                   > AVG(ucan_streaming_revenue) AS AVG_UCAN_REVENUE,
                   > AVG(emea_streaming_revenue) AS AVG_EMEA_REVENUE,
                   > AVG(latm_streaming_revenue) AS AVG_LATM_REVENUE,
                   > AVG(apac_streaming_revenue) AS AVG_APAC_REVENUE,
                   > AVG(global_revenue) AS AVG_GLOBAL_REVENUE
                   > FROM netflix_revenue;
```

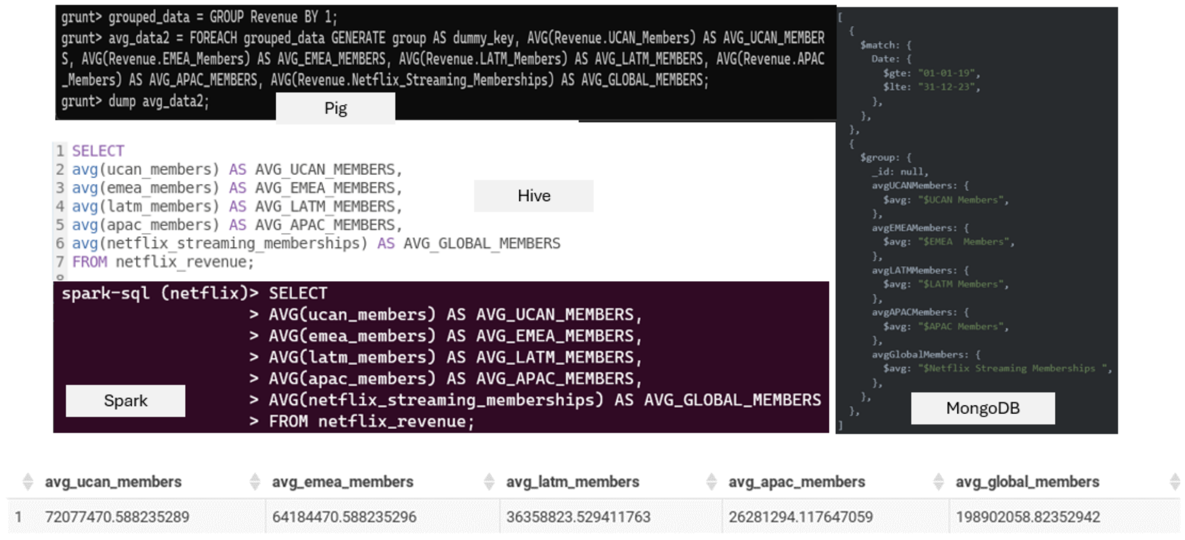
```
[
  {
    $match: {
      Date: {
        $gte: "01-01-19",
        $lte: "31-12-23",
      },
    },
  },
  {
    $group: {
      _id: null,
      UCAN_Streaming_Revenue: {
        $avg: "$UCAN Streaming Revenue",
      },
      EMEA_Streaming_Revenue: {
        $avg: "$EMEA Streaming Revenue",
      },
      LATM_Streaming_Revenue: {
        $avg: "$LATM Streaming Revenue",
      },
      APAC_Streaming_Revenue: {
        $avg: "$APAC Streaming Revenue",
      },
      Global_Streaming_Revenue: {
        $avg: "$Global Revenue",
      },
    },
  },
]
```

```
(1,3.068940705882353E9,2.0751643529411764E9,8.629001176470588E8,6.830685882352941E8,6.742788235294118E9)
```

Figure 4: Analysis 1 query and result from Pig

From the result analyze using Pig, we could observe that UCAN region has the most revenue from 2019-2023.

4.3.2 ANALYSIS 2: Average members for different regions from 2019-2023



```
grunt> grouped_data = GROUP Revenue BY 1;
grunt> avg_data2 = FOREACH grouped_data GENERATE group AS dummy_key, AVG(Revenue.UCAN_Members) AS AVG_UCAN_MEMBER
S, AVG(Revenue.EMEA_Members) AS AVG_EMEA_MEMBERS, AVG(Revenue.LATM_Members) AS AVG_LATM_MEMBERS, AVG(Revenue.APAC
_Members) AS AVG_APAC_MEMBERS, AVG(Revenue.Netflix_Streaming_Memberships) AS AVG_GLOBAL_MEMBERS;
grunt> dump avg_data2;
```

```
SELECT
avg(ucan_members) AS AVG_UCAN_MEMBERS,
avg(emea_members) AS AVG_EMEA_MEMBERS,
avg(latm_members) AS AVG_LATM_MEMBERS,
avg(apac_members) AS AVG_APAC_MEMBERS,
avg(netflix_streaming_memberships) AS AVG_GLOBAL_MEMBERS
FROM netflix_revenue;
```

```
spark-sql (netflix)> SELECT
                   > AVG(ucan_members) AS AVG_UCAN_MEMBERS,
                   > AVG(emea_members) AS AVG_EMEA_MEMBERS,
                   > AVG(latm_members) AS AVG_LATM_MEMBERS,
                   > AVG(apac_members) AS AVG_APAC_MEMBERS,
                   > AVG(netflix_streaming_memberships) AS AVG_GLOBAL_MEMBERS
                   > FROM netflix_revenue;
```

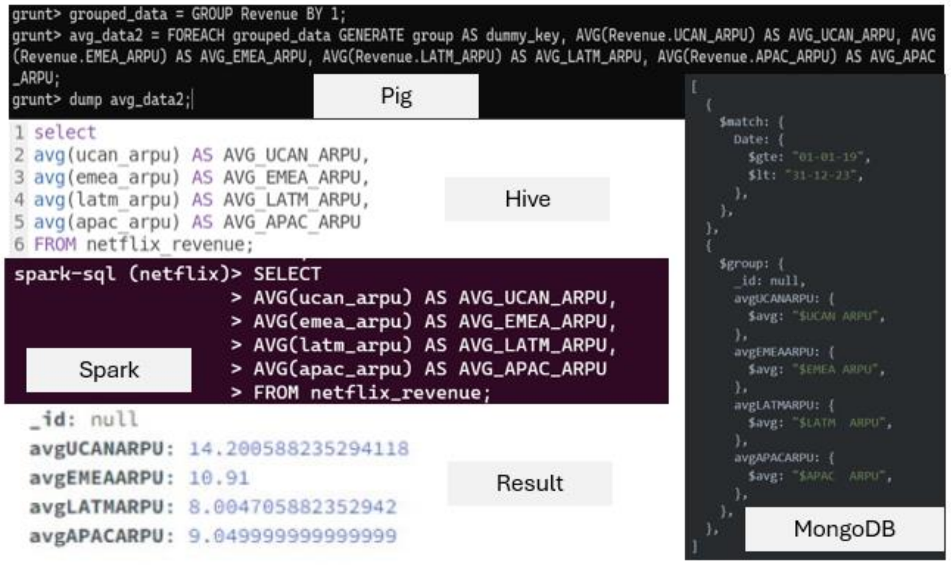
```
[
  {
    $match: {
      Date: {
        $gte: "01-01-19",
        $lte: "31-12-23",
      },
    },
  },
  {
    $group: {
      _id: null,
      avgUCANMembers: {
        $avg: "$UCAN Members",
      },
      avgEMEAMembers: {
        $avg: "$EMEA  Members",
      },
      avgLATMMembers: {
        $avg: "$LATM Members",
      },
      avgAPACMembers: {
        $avg: "$APAC Members",
      },
      avgGlobalMembers: {
        $avg: "$Netflix Streaming Memberships ",
      },
    },
  },
]
```

	avg_ucan_members	avg_emea_members	avg_latm_members	avg_apac_members	avg_global_members
1	72077470.588235289	64184470.588235296	36358823.529411763	26281294.117647059	198902058.82352942

Figure 5: Analysis 2 query and result from Hive

From the result in Hive, we could observe that the UCAN region has the most average members from 2019-2023.

4.3.3 ANALYSIS 3: Average ARPU for different regions from 2019-2023



```
grunt> grouped_data = GROUP Revenue BY 1;
grunt> avg_data2 = FOREACH grouped_data GENERATE group AS dummy_key, AVG(Revenue.UCAN_ARPU) AS AVG_UCAN_ARPU, AVG
(Revenue.EMEA_ARPU) AS AVG_EMEA_ARPU, AVG(Revenue.LATM_ARPU) AS AVG_LATM_ARPU, AVG(Revenue.APAC_ARPU) AS AVG_APAC
_ARPU;
grunt> dump avg_data2;
```

```
select
avg(ucan_arpu) AS AVG_UCAN_ARPU,
avg(emea_arpu) AS AVG_EMEA_ARPU,
avg(latm_arpu) AS AVG_LATM_ARPU,
avg(apac_arpu) AS AVG_APAC_ARPU
FROM netflix_revenue;
```

```
spark-sql (netflix)> SELECT
                   > AVG(ucan_arpu) AS AVG_UCAN_ARPU,
                   > AVG(emea_arpu) AS AVG_EMEA_ARPU,
                   > AVG(latm_arpu) AS AVG_LATM_ARPU,
                   > AVG(apac_arpu) AS AVG_APAC_ARPU
                   > FROM netflix_revenue;
```

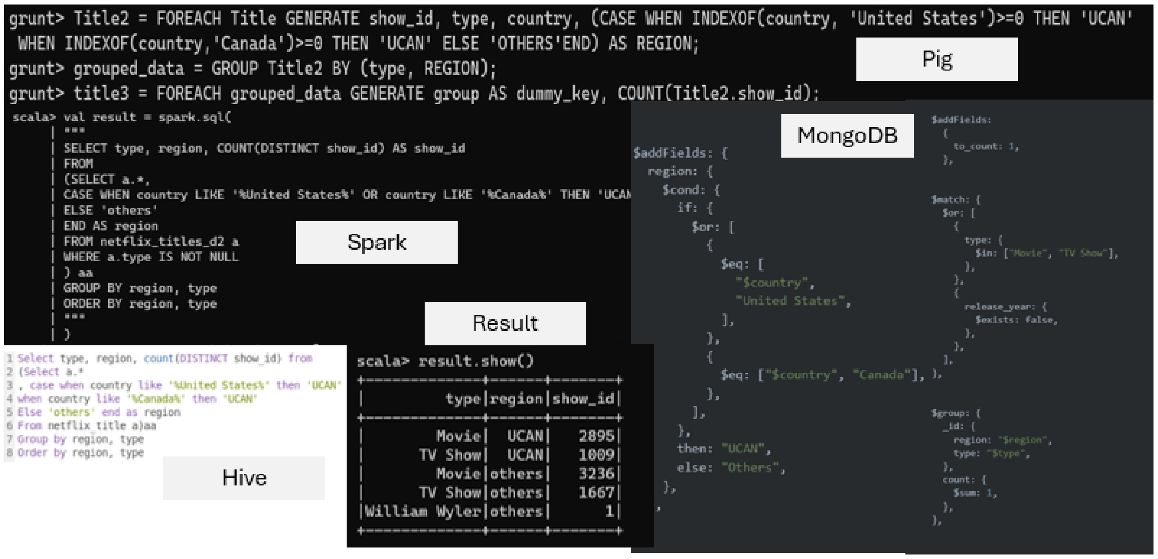
```
_id: null
avgUCANARPU: 14.200588235294118
avgEMEAARPU: 10.91
avgLATMARPU: 8.004705882352942
avgAPACARPU: 9.049999999999999
```

```
[
  {
    $match: {
      Date: {
        $gte: "01-01-19",
        $lt: "31-12-23",
      },
    },
  },
  {
    $group: {
      _id: null,
      avgUCANARPU: {
        $avg: "$UCAN ARPU",
      },
      avgEMEAARPU: {
        $avg: "$EMEA ARPU",
      },
      avgLATMARPU: {
        $avg: "$LATM  ARPU",
      },
      avgAPACARPU: {
        $avg: "$APAC  ARPU",
      },
    },
  },
]
```

Figure 6: Analysis 3 query and result from MongoDB

From the result analysis using MongoDB, the highest average ARPU from 2019-2023 is 14.2 which is also from UCAN region.

4.3.4 ANALYSIS 4: Comparison of number of movies and TV shows from 2019 to 2023 using group by type and region



```
grunt> Title2 = FOREACH Title GENERATE show_id, type, country, (CASE WHEN INDEXOF(country, 'United States')>=0 THEN 'UCAN'
 WHEN INDEXOF(country,'Canada')>=0 THEN 'UCAN' ELSE 'OTHERS'END) AS REGION;
grunt> grouped_data = GROUP Title2 BY (type, REGION);
grunt> title3 = FOREACH grouped_data GENERATE group AS dummy_key, COUNT(Title2.show_id);
```

```
scala> val result = spark.sql(
     | """
     | SELECT type, region, COUNT(DISTINCT show_id) AS show_id
     | FROM
     | (SELECT a.*,
     | CASE WHEN country LIKE '%United States%' OR country LIKE '%Canada%' THEN 'UCAN
     | ELSE 'others'
     | END AS region
     | FROM netflix_titles_d2 a
     | WHERE a.type IS NOT NULL
     | ) aa
     | GROUP BY region, type
     | ORDER BY region, type
     | """
     | )
```

```
Select type, region, count(DISTINCT show_id) from
(Select a.*
, case when country like '%United States%' then 'UCAN'
when country like '%Canada%' then 'UCAN'
Else 'others' end as region
From netflix_title a)aa
Group by region, type
Order by region, type
```

```
scala> result.show()
+------------+------+-------+
|        type|region|show_id|
+------------+------+-------+
|       Movie|  UCAN|   2895|
|     TV Show|  UCAN|   1009|
|       Movie|others|   3236|
|     TV Show|others|   1667|
|William Wyler|others|     1|
+------------+------+-------+
```

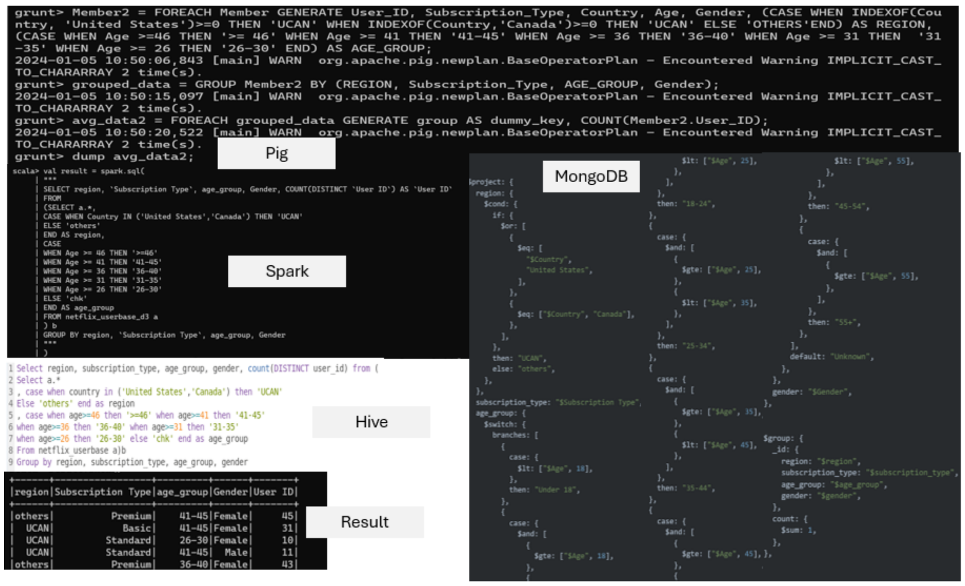
```
$addFields: {
  region: {
    $cond: {
      if: {
        $or: [
          {
            $eq: [
              "$country",
              "United States",
            ],
          },
          {
            $eq: ["$country", "Canada"], },
          },
        ],
      },
      then: "UCAN",
      else: "Others",
    },
  },
```

```
$addFields:
  {
    to_count: 1,
  },

$match: {
  $or: [
    {
      type: {
        $in: ["Movie", "TV Show"],
      },
    },
    {
      release_year: {
        $exists: false,
      },
    },
  ],
},

$group: {
  _id: {
    region: "$region",
    type: "$type",
  },
  count: {
    $sum: 1,
  },
},
```

Figure 7: Analysis 4 query and result from Spark

As from the analysis from Spark, the highest number of show type is Movie from UCAN.

4.3.5 ANALYSIS 5: Comparison of demographical data using group by region, subscription type, age and gender



```
grunt> Member2 = FOREACH Member GENERATE User_ID, Subscription_Type, Country, Age, Gender, (CASE WHEN INDEXOF(Cou
ntry, 'United States')>=0 THEN 'UCAN' WHEN INDEXOF(Country,'Canada')>=0 THEN 'UCAN' ELSE 'OTHERS'END) AS REGION,
(CASE WHEN Age >=46 THEN '>= 46' WHEN Age >= 41 THEN '41-45' WHEN Age >= 36 THEN '36-40' WHEN Age >= 31 THEN  '31
-35' WHEN Age >= 26 THEN '26-30' END) AS AGE_GROUP;
2024-01-05 10:50:06,843 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_
TO_CHARARRAY 2 time(s).
grunt> grouped_data = GROUP Member2 BY (REGION, Subscription_Type, AGE_GROUP, Gender);
2024-01-05 10:50:15,097 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_
TO_CHARARRAY 2 time(s).
grunt> avg_data2 = FOREACH grouped_data GENERATE group AS dummy_key, COUNT(Member2.User_ID);
2024-01-05 10:50:20,522 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_
TO_CHARARRAY 2 time(s).
grunt> dump avg_data2;
```

```
scala> val result = spark.sql(
     | """
     | SELECT region, `Subscription Type`, age_group, Gender, COUNT(DISTINCT `User ID`) AS `User ID`
     | FROM
     | (SELECT a.*,
     | CASE WHEN Country IN ('United States','Canada') THEN 'UCAN'
     | ELSE 'others'
     | END AS region,
     | CASE
     | WHEN Age >= 46 THEN '>=46'
     | WHEN Age >= 41 THEN '41-45'
     | WHEN Age >= 36 THEN '36-40'
     | WHEN Age >= 31 THEN '31-35'
     | WHEN Age >= 26 THEN '26-30'
     | ELSE 'chk'
     | END AS age_group
     | FROM netflix_userbase_d3 a
     | ) b
     | GROUP BY region, `Subscription Type`, age_group, Gender
     | """
     | )
```

```
Select region, subscription_type, age_group, gender, count(DISTINCT user_id) from (
Select a.*
, case when country in ('United States','Canada') then 'UCAN'
Else 'others' end as region
, case when age>=46 then '>=46' when age>=41 then '41-45'
when age>=36 then '36-40' when age>=31 then '31-35'
when age>=26 then '26-30' else 'chk' end as age_group
From netflix_userbase a)b
Group by region, subscription_type, age_group, gender
```

```
+------+-----------------+---------+------+-------+
|region|Subscription Type|age_group|Gender|User ID|
+------+-----------------+---------+------+-------+
|others|          Premium|    41-45|Female|     45|
|  UCAN|            Basic|    41-45|Female|     31|
|  UCAN|         Standard|    26-30|Female|     10|
|  UCAN|         Standard|    41-45|  Male|     11|
|others|          Premium|    36-40|Female|     43|
```

```
$project: {
  region: {
    $cond: {
      if: {
        $or: [
          {
            $eq: [
              "$Country",
              "United States",
            ],
          },
          {
            $eq: ["$Country", "Canada"],
          },
        ],
      },
      then: "UCAN",
      else: "others",
    },
  },
  subscription_type: "$Subscription Type",
  age_group: {
    $switch: {
      branches: [
        {
          case: {
            $lt: ["$Age", 18],
          },
          then: "Under 18",
        },
        {
          case: {
            $and: [
              {
                $gte: ["$Age", 18],
              },
              {
```

```
                $lt: ["$Age", 25],
              },
            ],
          },
          then: "18-24",
        },
        {
          case: {
            $and: [
              {
                $gte: ["$Age", 25],
              },
              {
                $lt: ["$Age", 35],
              },
            ],
          },
          then: "25-34",
        },
        {
          case: {
            $and: [
              {
                $gte: ["$Age", 35],
              },
              {
                $lt: ["$Age", 45],
              },
            ],
          },
          then: "35-44",
        },
        {
          case: {
            $and: [
              {
                $gte: ["$Age", 45], },
              },
              {
```

```
                $lt: ["$Age", 55],
              },
            ],
          },
          then: "45-54",
        },
        {
          case: {
            $and: [
              {
                $gte: ["$Age", 55],
              },
            ],
          },
          then: "55+",
        },
      ],
      default: "Unknown",
    },
  },
  gender: "$Gender",
},

$group: {
  _id: {
    region: "$region",
    subscription_type: "$subscription_type",
    age_group: "$age_group",
    gender: "$gender",
  },
  count: {
    $sum: 1,
  },
},
```

Figure 8: Analysis 5 query and result from Spark

From the 5th analysis, the result show that most of the Netflix members are Female between 26-30 age group, from UCAN region and subscript to basic membership of Netflix.

4.3.6 DATA ANALYSIS RESULT

Three datasets were used for this study.

Dataset 1: Netflix subscriber figures and revenue growth by region from 2019 to 2023. This dataset has 15 columns and 17 rows.

Dataset 2: Listings of movies and tv shows on Netflix on different years and in different countries. This dataset has 12 columns and 8,807 rows.

Dataset 3: Netflix user data from different countries with subscription types. This dataset has 10 columns and 2,500 rows.

Dataset	Hive	Pig	MongoDB	Apache Spark
Dataset 1	**0s**	17s	**0s**	3s
Dataset 2	**0s**	12s	**0s**	2s
Dataset 3	**0s**	16s	**0s**	1s

Table 1: Storage Time Comparison

Query	Hive	Pig	MongoDB	Apache Spark
Q1	84s	24s	**0.002s**	0.687s
Q2	63s	21s	**0.001s**	0.236s
Q3	66s	17s	**0.001s**	0.278s
Q4	135s	21s	**0.042s**	1.280s
Q5	84s	16s	**0.059s**	2.630s
Total	432s	99s	**0.104s**	5.078s

Table 2: Query Time Comparison

In terms of performing queries, MongoDB has the best performance followed by Apache Spark, Hive and Pig. MongoDB offers easy access to data, facilitates horizontal scaling through efficient database sharding, and demonstrates superior query execution speed when compared to other tools. Pig is only better than Hive as Apache Pig offers more optimization and control on the data flow than Hive. Although Pig does not perform well in both storage and query execution, it shows detailed explanation for each running step on the screen which makes it easier to debug.

Apache Spark was chosen for our project for several key reasons although we understand that MongoDB has the best performed in time when executing query. First, it's user-friendly, which makes it easier for our team to handle complex data tasks. Spark's design simplifies working with big datasets, allowing us to focus more on the analysis rather than on figuring out how to use the tool. This is especially helpful when dealing with intricate data processes that our project requires. In addition to its user-friendliness, Spark's usability is a significant advantage for our project. Unlike MongoDB, which often requires complex JSON formats for queries, Spark provides a more accessible SQL-like query structure. This feature makes it easier for team members familiar with SQL to adapt and work efficiently. Furthermore, Spark supports HiveQL syntax, as highlighted on its official website Spark SQL, which adds to its usability, especially for those already acquainted with Hive.

In terms of performance, both Spark and MongoDB outperform Hive and Pig. However, when comparing Spark with MongoDB, Spark's user-friendly SQL-like interface gives it an edge in usability. This usability, combined with Spark's versatility in handling various data tasks, efficient big data management, and seamless integration with other tools, makes it a more suitable choice for our project.

While MongoDB excels in speed for some tasks, Spark's overall usability and functionality align better with our project's complex and diverse data analysis needs.

From the analysis, we can conclude that revenue around the globe has been increasing with United States having the highest amount of revenue and number of members. The summary also shows that Netflix is currently focusing more on Movies than on TV shows. Among movies from the USA, Netflix seems to prefer movies produced during the 2010s and newly released. The choice of movies seems to be helping Netflix in retaining existing customers and attracting new customers. For Netflix to achieve higher revenue growth, they should diversify the content library with more genres to attract a broader audience. Forming partnerships with famous filmmakers, directors and writers to create high-quality original content can generate buzz and attract subscribers who are fans of the creators.

5 LIMITATIONS AND CHALLENGES

5.1 DATASET AND ENVIRONMENT LIMITATIONS

Security and Privacy

While Hadoop is now widely used around the world, the security provided is still very poor and this matter has been widely negligible. Regarding HDFS, the security system is very poor and HDFS needs to be protected from vulnerabilities and breaches by providing a Firewall, providing an Intrusion detection system or encrypting blocks and nodes (Weets, J. F. et al, 2015).

Scalability and Storage Capacity

Large datasets can pose challenges in terms of storage, processing and analysis. Handling massive volumes of data may require specialized storage and distributed processing frameworks. The sole real scalability threat can be the limitations of the memory space available for the NameNode. The quantity of the metadata in the NameNode is limited since it is stored in memory (Weets, J. F. et al, 2015).

Hadoop Limitations

Hadoop cannot handle updates (deletion, insertion or updating a record) since it uses HDFS. HDFS operates on the Write-Once, Read-Many data access model. This indicates that once the file is created, written and closed, it cannot be edited again except for appending the data. Hadoop also does not provide access to record sets that are subsets of the total data. To perform analysis on a file in Hadoop, the whole files must be read or loaded. Hence, it is not suitable for interactive or ad-hoc queries (Sethi & Maposa, 2015).

Small and Independent Sample Size

Dataset 3 only consists of 2,500 subscribers, which is a limited number of observations that may reduce the study's statistical and analytical power. Moreover, the three datasets are treated in isolation, there is no integration or interaction among them in the study. This reduces the ability to build a contextual comprehensive understanding of the study, leading to incomplete insights into a broader picture.

5.2 CHALLENGES IN LEARNING DIFFERENT SYNTAX

MongoDB uses the dollar sign ($) before operator statements or when referring to a field in the document. Notably, it lacks a dedicated create database function, which is common in other big data tools such as Hive, Pig, and Spark. As a result, creating new databases requires careful consideration, particularly when incorporating new indices. MongoDB has query limitations, including a lack of join operations and the ability to perform machine learning modelling. Moreover, MongoDB differs from traditional SQL methodologies in that it handles complex queries using an aggregation pipeline rather than SQL queries.

As for Pig, although it can be quite a powerful and simple language to use, the downside is that it is something new to learn and master compared to Hive. Hive requires very few lines of code because of its SQL like resemblance. Pig requires multiple queries to perform a task compared to other tools such as group by syntax. Besides that, the output is not in common tabular form and exists without a header, making it difficult to read.

On the other hand, when executing a query in spark-SQL, it exclusively displays row results without presenting column headers. This characteristic, while providing a concise output, may pose challenges for users who prefer a more detailed representation of query results. Moreover, when utilizing Spark in spark-shell with Scala, SQL operations are embedded as strings within Scala code. While this approach facilitates integration, it results in longer queries, potentially making code maintenance more intricate. Additionally, errors in SQL queries are only checked at runtime, introducing the possibility of encountering issues during the execution phase rather than at the coding stage. This runtime error checking, though dynamic, may contribute to a less streamlined development process and highlights the importance of thorough testing to identify potential errors early in the development cycle.

6 CONCLUSIONS

In conclusion, usability emerges as a pivotal factor distinguishing Spark and MongoDB from alternative big data processing tools like Hive and Pig. Notably, Spark offers a user-friendly experience with its SQL-like query structure, a notable departure from MongoDB's long JSON format. The support for HiveQL syntax further enhances Spark's usability, facilitating seamless integration for users familiar with Hive. When evaluating processing time, both Spark and MongoDB demonstrate superior efficiency compared to traditional tools like Hive and Pig. However, in the comparison between Spark and MongoDB, Spark stands out with enhanced usability, underscoring its adaptability and ease of use in diverse data processing scenarios. This advantageous combination of SQL-like queries, support for HiveQL, and improved processing efficiency positions Spark as a compelling choice for developers and data engineers seeking a versatile and user-centric solution

7 REFERENCES

Apache Spark. (2023). Apache SparkTM - unified engine for large-scale data analytics. Apache SparkTM - Unified Engine for large-scale data analytics. https://spark.apache.org/

Databricks. (2023). What is SPARK SQL? https://www.databricks.com/glossary/what-is-spark-sql#:~:text=Spark%20SQL%20is%20a%20Spark,on%20existing%20deployments%20and%20data

GeeksforGeeks. (2023). MongoDB: An introduction. What is MongoDB – Working and Features. https://www.geeksforgeeks.org/mongodb-an-introduction/?ref=gcse

Laskowski, J. (2023). Spark-shell shell script. spark-shell - The Internals of Spark Core. https://books.japila.pl/apache-spark-internals/tools/spark-shell/

Maposa, T. T., & Sethi, M. (2015). SQL-on-Hadoop: The Most Probable Future in Big Data Analytics. *Advances in Computer Science and Information Technology*. 2(13): 9-14.

Pol, U. R. (2016). Big Data Analysis: Comparison of Hadoop MapReduce, Pig and Hive. *International Journal of Innovative Research in Science, Engineering and Technology*, 5(6),9687-93.

Sadq, Z. M. (2013). Analyzing Netflix's Strategy. *International Journal of Science and Research (IJSR)*, 2013, pp. 2319-7064.

Weets, J. F., Kakhani, M. K., & Kumar, A. (2015). Limitations and challenges of HDFS and MapReduce. *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, Greater Noida, India, 2015, pp. 545-549, doi: 10.1109/ICGCIoT.2015.7380524.