ANALYSIS OF H1B VISA APPLICATIONS

Using Hadoop Ecosystem

Presented By:

Shannon Michelle D'souza

Student ID : S180010900133

Registration No : R180010900252

NIIT Centre : Borivali West

Date Of Submission : 15th January 2018

Table of Contents

INTRODUCTION

BIG DATA

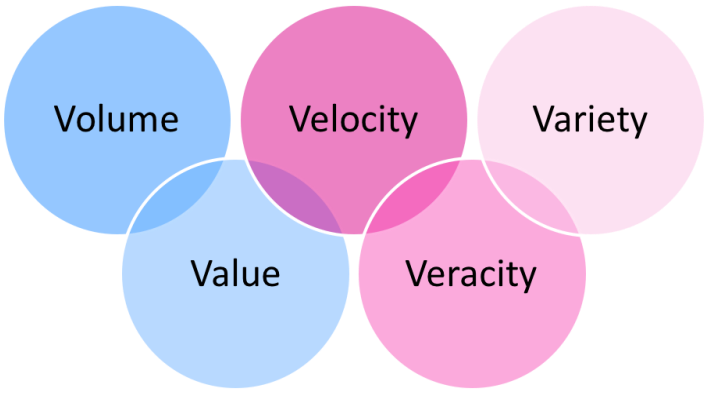
A term used to describe a collection of data that is huge in size and yet growing exponentially with time. Such a data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently. Big data requires a different processing approach, one that uses massive parallelism on readily-available hardware (commodity machines).

It could be found in any of the three forms:

1. Structured
2. Unstructured
3. Semi-structured

Big data is often described using five Vs: Volume, Velocity, Variety, Veracity, and Value.

- Volume - refers to the vast amounts of data generated every second
- Velocity - refers to the speed at which new data is generated and the speed at which data moves around.
- Variety - refers to the different types of data we can now use.
- Veracity - refers to the messiness or trustworthiness of the data.
- Value – Having access to Big Data is well and good but unless we can extract value from it, it is useless.



One of the essential components of a big data system is its processing frameworks. Processing frameworks are responsible for performing computations over the data in the system, either by reading from non-volatile storage or as it is ingested into the system.

Some of the top open source Big Data processing frameworks being used today are:

- Apache Hadoop
- Apache Storm
- Apache Samza
- Apache Spark
- Apache Flink

CHALLENGES OF BIG DATA WITH RDBMS

Relational databases are so commonly used in most organizations these days that many people may not even be aware that there are other types of databases. Relational databases perform transaction update functions very well, particularly handling the difficult issues of consistency during the update.

However, relational databases struggle with the efficiency of certain operations key to Big Data management:

1. It is very expensive to set up and maintain a database system
2. They don't scale well to very large sizes
3. They cannot handle unstructured data very well (i.e. google type searching) nor can they deal with data in unexpected formats well
4. It is difficult to implement certain kinds of basic queries using SQL and relational databases
5. Some relational databases have limits on field lengths. When designing the database, one has to specify the amount of data you can fit into a field. Some names or search queries are shorter than the actual, and this can lead to data loss

HADOOP

Hadoop is an open-source, Java-based programming framework that allows you to first store Big Data in a distributed environment so that you can process it parallelly. It is part of the Apache project sponsored by the Apache Software Foundation.

HISTORY OF HADOOP

Hadoop was created by computer scientists Doug Cutting and Mike Cafarella in 2006 to support distribution for the Nutch search engine. Hadoop has its origins in Apache Nutch which is an open source web search engine itself a part of the Lucene project. It was inspired by Google's MapReduce.

FEATURES OF HADOOP

- **Open-source**
  Apache Hadoop is an open source project which means that its code can be modified according to business requirements.

- **Distributed Processing**
  The data storage is maintained in a distributed manner in HDFS across the cluster and is processed in parallel on a cluster of nodes.

- **Fault Tolerance**
  When any node goes down, the data on that node can be recovered easily from other nodes. Failures of a particular node or task are recovered automatically by the framework. By default, three replicas of each block are stored across the cluster in Hadoop and it's changed only when required.

- **Reliability**
  Due to replication of data in the cluster, data can be reliable which is stored on a cluster of machines despite machine failures.

- **High Availability**
  Data is available and accessible even there occurs a hardware failure due to multiple copies of data. If any incidents occurred such as if your machine or few hardware crashes, then data will be accessed from another path.

- **Scalability**
  Hadoop is highly scalable and hardware can be easily added to the nodes. It also provides horizontal scalability which means new nodes can be added to the cluster without any downtime**.**

- **Economic**
  Hadoop is not very expensive as it runs on a cluster of commodity hardware. We do not require any specialized machine for it. Hadoop provides huge cost reduction since it is very easy to add more nodes on the top here. So, if the requirement increases, then there is an increase of nodes, without any downtime and without any much of pre-planning.

- **Easy to use**
  No need of client to deal with distributed computing, the framework takes care of all the things.

- **Data Locality**
  Hadoop works on data locality principle which states that the movement of computation of data instead of data to computation. When a client submits his algorithm, then the algorithm is moved to data in the cluster instead of bringing data to the location where the algorithm is submitted and then processing it.

PROS AND CONS OF HADOOP

PROS:

1. **Range of data sources**
   The data collected from various sources will be of structured or unstructured form. A lot of time would need to be allotted in order to convert all the collected data into a single format. Hadoop saves this time as it can derive valuable data from any form of data.

2. **Cost-effective**
   Hadoop is a cost-effective solution for data storage purposes. This helps in the long run because it stores the entire raw data generated by a company. If the company changes the direction of its processes in the future, it can easily refer to the raw data and take the necessary steps. This would not have been possible in the traditional approach because the raw data would have been deleted due to increase in expenses.

3. **Speed**
   Hadoop uses a storage system wherein the data is stored on a distributed file system. Since the tools used for the processing of data are located on same servers as the data, the processing operation is carried out at a faster rate. Therefore, you can process terabytes of data within minutes using Hadoop.

4. **Multiple copies**
   Hadoop automatically duplicates the data that is stored in it and creates multiple copies. This is done to ensure that in case there is a failure, data is not lost. Hadoop understands that the data stored by the company is important and should not be lost unless the company discards it.

CONS:

1. **Lack of preventive measures**
   In Hadoop, the security measures are disabled by default. The person responsible for data analytics should be aware of this fact and take the required measures to secure the data.

2. **Small Data concerns**
   Hadoop is one such platform wherein only large business that generates big data can utilize its functions. It cannot efficiently perform in small data environments.

3. **Risky functioning**
   Java is one of the most widely used programming languages. It has also been connected to various controversies because cybercriminals can easily exploit the frameworks that are built on Java. Hadoop is one such framework that is built entirely on Java. Therefore, the platform is vulnerable and can cause unforeseen damages.

PROJECT DEFINITION

PROJECT SUMMARY

To analyze H1B Visa application dataset on the basis of certain objectives which are to be achieved using Hadoop technologies, specifically MapReduce, Hive, Pig and Sqoop

PREREQUISITE

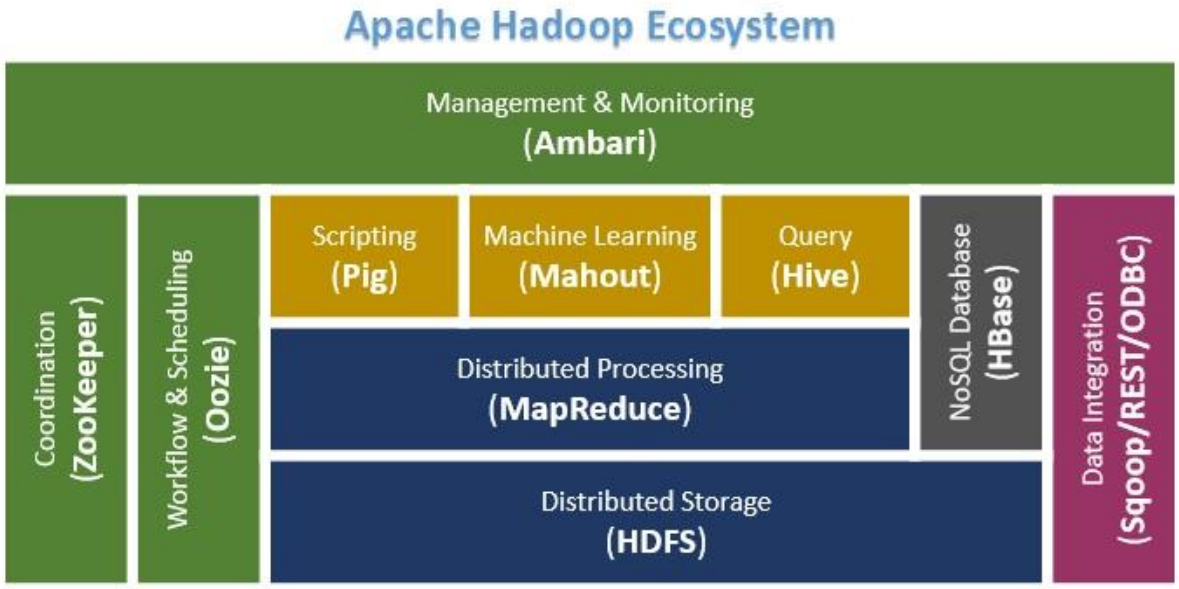
HARDWARE REQUIREMENT:

1. Operating System: Ubuntu 14.04 LTS
2. VMware Player

SOFTWARE REQUIREMENT:

1. Java Version: 1.7.0_151
2. Hadoop Version: 2.6.0
3. Hive Version: 1.2.1
4. Pig Version: 0.13.0

HADOOP TECHNOLOGY USED



Hadoop has gained its popularity due to its ability to store, analyze and access a large amount of data, quickly and cost-effectively through clusters of commodity hardware. Apache Hadoop, however, is actually a collection of several components and not just a single product. These components are broadly used to make Hadoop more usable. Most importantly, they are all open source.

The following Hadoop components are used in order to achieve the objectives of this project:

1. **Hadoop Distributed File System (HDFS)**
   HDFS is a distributed file-system that provides high throughput access to data. When data is pushed to HDFS, it automatically splits up into multiple blocks and stores/replicates the data thus ensuring high availability and fault tolerance.

   Main Components of HDFS:

   a. ***NameNode***
      It acts as the master of the system. It maintains the name system i.e., directories and files and manages the blocks which are present on the DataNodes.

   b. ***DataNodes***
      They are the slaves which are deployed on each machine and provide the actual storage. They are responsible for serving read and write requests for the clients.

2. **MapReduce**
   Hadoop MapReduce is a software framework for easily writing applications which process big amounts of data in parallel on large clusters of commodity hardware in

a reliable, fault-tolerant manner. In terms of programming, there are two functions which are most common in MapReduce:

a. *The Map Task*
   Master computer or node takes input and divides it into smaller parts and distribute it on other worker nodes. All worker nodes solve their own small problem and give their answers to the master node.

b. *The Reduce Task*
   Master node combines all answers coming from worker node and forms it in some form of output which is the answer to our big distributed problem.

Generally, both the input and the output are reserved in a file-system. The framework is responsible for scheduling tasks, monitoring them and even re-executes the failed tasks.

3. **Hive**
   Hive provides an SQL like interface to Hadoop. It is a data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems.
   It provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL.

4. **Pig**
   Pig is a platform for analyzing and querying huge data sets that consist of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. Pig's built-in operations can make sense of semi-structured data, such as log files, and the language is extensible using Java to add support for custom data types and transformations.

5. **Sqoop (MySQL-to-Hadoop)**
   Sqoop is a tool designed for efficiently transferring structured data from SQL Server and SQL Azure to HDFS and then uses it in MapReduce and Hive jobs. One can even use Sqoop to move data from HDFS to SQL Server.

PROJECT DESCRIPTION

The H1B is an employment-based, non-immigrant visa category for temporary foreign workers in the United States. For a foreign national to apply for H1B visa, a US employer must offer a job and petition for an H1B visa with the US immigration department. This is the most common visa status applied for and held by international students once they complete college/ higher education (Masters, Ph.D.) and work in a full-time position.

DATA SET:
The dataset used for this case study contains nearly 3 million records of H1B Visa applicants between the year 2011 to 2016.

COLUMN DEFINITION:

1. **S. No**: Serial number of the application
2. **Case Status**: Status associated with the last significant event or decision. Valid values include:
    (i) Certified = Employer filed the Labour Condition Application (LCA), which was approved by Department of Labour (DOL)
    (ii) Certified Withdrawn = LCA was approved but later withdrawn by employer
    (iii) Withdrawn = LCA was withdrawn by employer before approval
    (iv) Denied = LCA was denied by DOL
3. **Employer Name**: Name of employer submitting labor condition application
4. **SOC Name**: The Occupational name associated with the Standard Occupational Classification (SOC)_code.
    SOC_CODE is the occupational code associated with the job being requested for temporary labor condition, as classified by SOC System.
5. **Job Title**: Title of the job
6. **Full Time Position**: Valid values include:
    (i) Y = Full Time Position
    (ii) N = Part Time Position
7. **Prevailing Wage**: Prevailing Wage for the job being requested for temporary labor condition. The wage is listed at annual scale in USD. The prevailing wage for a job position is defined as the average wage paid to similarly employed workers in the requested occupation in the area of intended employment. It is based on the employer's minimum requirements for the position.

8. **Year**: Year in which the H1B visa petition was filed

9. **Worksite**: City and State information of the foreign worker's intended area of employment

10. **Longitude**: Longitude of the Worksite

11. **Latitude**: Latitude of the Worksite

PROJECT OUTLINE

Title	**Analysis of H1B Visa Applications Using Hadoop Ecosystem**
Input	H1B Applications data set from 2011 to 2016
Data Elements	S No, Case Status, Employer Name, SOC Name, Job Title, Full-Time Position, Prevailing Wage, Year, Work Site, Longitude, Latitude
Analysis Relevance	Government, Immigration, Employment
Purpose	To provide a perspective on: 1. Application trends for each year 2. Job Trends 3. Average growth rate for applications over a period of 5 years 4. Probability of an H1B Visa application being certified 5. Top employers applying for the visa 6. The success rate of acceptance for each employer 7. Any changes to be made in the H1B Visa policies
Methodology	Agile

PROJECT IMPLEMENTATION

ASSUMPTIONS

1. Hadoop Cluster is running
2. Ecosystem Products (Pig, Hive, and Sqoop) are installed
3. MySQL Database is installed
4. H1B Visa Data available on HDFS in GZIP format

CLEANING OF RAW DATA

Initially, H1B Visa data is available as a .gzip compressed file. Using 'tar' command, the actual dataset (h1b.csv) is extracted and loaded on HDFS.

In the original H1B Visa data, a few columns are enclosed within double quotes. Moreover, commas are used to separate different column values and even within columns whose values are enclosed within double quotes. This could cause a hindrance during the actual analysis stage.

To overcome the above situation, Hive technology is used to clean and process this raw data into a ready-to-use form. The following steps are followed to achieve the final data which is used for further analysis:

STEP 1: Create a table 'h1b_app1' which uses SerDe properties to load the data from h1b.csv.

Total number of records = 3002458

STEP 2: Create a table 'h1b_app2' which loads data from table 'h1b_app1' while filtering out records with case status = 'NA' and removing unnecessary tab spaces present in any of the column values.

Total number of records = 3002445

STEP 3: Create a table 'h1b_app3' which loads data from table 'h1b_app2' while converting the case status of those records with values: 'PENDING QUALITY AND COMPLIANCE REVIEW – UNASSIGNED', 'REJECTED' and 'INVALIDATED' to 'DENIED' and also set the prevailing wage to a default value (100,000 USD) if not available.

STEP 4: Create a table 'h1b_app4' which loads data from table 'h1b_app3' while converting all lowercase string values (except values of employer name column) to uppercase.

DATA ANALYSIS

Query 1_A

OBJECTIVE: To find if the number of petitions with Data Engineer job title is increasing over time

TECHNOLOGY USED: MapReduce (in Java)

IMPLEMENTATION LOGIC:

Job 1: Obtains the application count for Data Engineer jobs for each year

1. H1B data is read one record at a time by the map() method.
2. Only those records that contain Data Engineer as job title are passed to the Reducer.
3. In the reduce() method, the total application count for each year is generated.
4. Key: Year
   Value: Count

Job 2: Calculates the overall average growth for a 5 year time period

1. The output of Job 1 is taken as input here.
2. In the map() method, the growth percent is calculated for each year.
3. In the reduce() method, the sum of growth percent and the number of records is obtained.
4. The overall average growth percentage calculation is performed in the cleanup() method.
5. Final output is generated

EXPECTED OUTPUT FORMAT:

<year> <application count> <growth percentage>

Query 1_B

OBJECTIVE: To find the top 5 job titles which have the highest average growth in applications

TECHNOLOGY USED: MapReduce (in Java)

IMPLEMENTATION LOGIC:

Job 1: Obtains the year wise application count for each job title

1. H1B data is read one record at a time by the map() method.
2. Job title and year of each record are extracted and passed to the Reducer.
3. In the reduce() method, the total application count for each year is generated for each job title.
4. Key: Job title
   Value: Application count for each year

Job 2: Calculates the overall average growth for each job title

1. The output of Job 1 is taken as input here.
2. In the map() method, the growth percent is calculated for each year and the overall average percentage is generated.

   Key: Average growth
   Value: Job title

3. In the reduce() method, the records are sorted by the average growth in descending order.
4. Only the first 5 records are written as job output.

   Key: Job title
   Value: Average growth

5. Final output is generated

EXPECTED OUTPUT FORMAT:

<job title> <average growth percentage>

Query 2_A

OBJECTIVE: To find which part of the US has the most Data Engineer jobs for each year

TECHNOLOGY USED: Hive

IMPLEMENTATION LOGIC:

A. <u>When the user wants an overall analysis (all years)</u>

1. Extract worksite, year and application count [using COUNT()] from the table 'h1b_app4'
2. Filter the data for:
    i. case status = 'CERTIFIED'
    ii. job title contains the string 'Data Engineer'
3. Group the data by worksite
4. Order the data in the descending order of application count
5. Display the final output using a SELECT query

B. <u>When the user wants the analysis for a specific year</u>

1. Take the user's year choice as parameter
2. Extract worksite, year and application count [using COUNT()] from the table 'h1b_app4'
3. Filter the data for:
    i. case status = 'CERTIFIED'
    ii. job title contains the string 'Data Engineer'
    iii. year depending on the user's choice
4. Group the data by worksite and year
5. Order the data in the descending order of application count
6. Display the final output using a SELECT query

EXPECTED OUTPUT FORMAT:

<worksite> <year> <application count>

Query 2_B

OBJECTIVE: To find the top 5 locations in the US which have got certified visa for each year

TECHNOLOGY USED: Pig

IMPLEMENTATION LOGIC:

A. When the user wants an overall analysis (all years)

1. Load the data from /niit_final_project/cleaned_data into bag 'bag1'
2. For each record in 'bag1', extract the columns: case status, SOC name, job title, worksite and year and store in 'bag2'
3. Filter 'bag2' on case status = 'CERTIFIED'
4. Group the filtered data on worksite, case status and year
5. Generate the application count for each group and store it in bag 'countby2'
6. Order 'countby2' data in descending of application count
7. Generate the final top 5 output.
8. Store the final result in HDFS

B. When the user wants the analysis for a specific year

1. Take the user's year choice as parameter
2. Load the data from /niit_final_project/cleaned_data into bag 'bag1'
3. For each record in 'bag1', extract the columns: case status, SOC name, job title, worksite and year and store in 'bag2'
4. Filter 'bag2' on case status = 'CERTIFIED'
5. Group the filtered data on worksite, case status and year
6. Generate the application count for each group and store it in bag 'countby2'
7. Filter 'countby2' on year selected by the user
8. Order the filtered data in descending of application count
9. Generate the final top 5 output.
10. Store the final result in HDFS

EXPECTED OUTPUT FORMAT:

<worksite> <case status> <year> <application count>

Query 3

OBJECTIVE: To find which industry (SOC Name) has the most number of Data Scientist positions

TECHNOLOGY USED: Hive

IMPLEMENTATION LOGIC:

1. Extract SOC name and application count [using COUNT()] from the table 'h1b_app4'
2. Filter the data for:
    i. case status = 'CERTIFIED'
    ii. job title contains the string 'Data Scientist'
3. Group the data by SOC name
4. Order the data in the descending order of application count
5. Display the final output using a SELECT query

EXPECTED OUTPUT FORMAT:

<soc name> <application count>

Query 4

OBJECTIVE: To find the top 5 employers that file the most number of petitions each year

TECHNOLOGY USED: Pig

IMPLEMENTATION LOGIC:

A. When the user wants an overall analysis (all years)

1. Load the data from /niit_final_project/cleaned_data into bag 'bag1'
2. Group the 'bag1' data on year and employer name
3. Generate the application count for each group and store it in bag 'countby2'
4. Order 'countby2' data in descending of application count
5. Generate the final top 5 output.
6. Store the final result in HDFS

B. When the user wants the analysis for a specific year

1. Take the user's year choice as parameter
2. Load the data from /niit_final_project/cleaned_data into bag 'bag1'
3. Group the 'bag1' data on year and employer name
4. Generate the application count for each group and store it in bag 'countby2'
5. Filter 'countby2' on year selected by the user
6. Order the filtered data in descending of application count
7. Generate the final top 5 output.
8. Store the final result in HDFS

EXPECTED OUTPUT FORMAT:

<year> <employer name> <application count>

Query 5

OBJECTIVE: To find the most popular top 10 job positions for H1B visa applications for each year

a) for all the applications
b) for only certified applications

TECHNOLOGY USED: Hive

IMPLEMENTATION LOGIC:

A. When the user wants the analysis on all applications

I. Overall Analysis

1. Extract year, job title and application count [using COUNT()] from the table 'h1b_app4'
2. Group the data by year and job title
3. Order the data in the descending order of application count
4. Display top 10 records as final output using a SELECT query

II. Year-specific Analysis

1. Take the user's year choice as parameter
2. Extract year, job title and application count [using COUNT()] from the table 'h1b_app4'
3. Filter the data for year based on the value of parameter passed by user
4. Group the data by year and job title
5. Order the data in the descending order of application count
6. Display top 10 records as final output using a SELECT query

B. When the user wants the analysis of certified applications only

I. Overall Analysis

1. Extract year, job title and application count [using COUNT()] from the table 'h1b_app4'
2. Filter the data for case status = 'CERTIFIED'
3. Group the data by year and job title
4. Order the data in the descending order of application count
5. Display top 10 records as final output using a SELECT query

II. *Year-specific Analysis*

1. Take the user's year choice as parameter
2. Extract year, job title and application count [using COUNT()] from the table 'h1b_app4'
3. Filter the data for:
    i. case status = 'CERTIFIED'
    ii. year based on the value of the parameter passed by the user
4. Group the data by year and job title
5. Order the data in the descending order of application count
6. Display top 10 records as final output using a SELECT query

EXPECTED OUTPUT FORMAT:

<year> <job title> <application count>

Query 6

OBJECTIVE: To find the count and the percentage of each case status on total applications for each year. Also, to create a line graph depicting the pattern of ALL the cases over the period of time.

TECHNOLOGY USED: MapReduce (in Java)

IMPLEMENTATION LOGIC:

Job 1: Obtains the case status wise application count for each year

1. User's year choice is taken as a parameter.
2. H1B data is read one record at a time by the map() method.
3. The year and case status of each record are extracted and passed to the Reducer depending on year choice parameter.
4. In the reduce() method, the total application count for each case status is generated for each year.
5. Key: Year
   Value: Application count for each case status, total application count

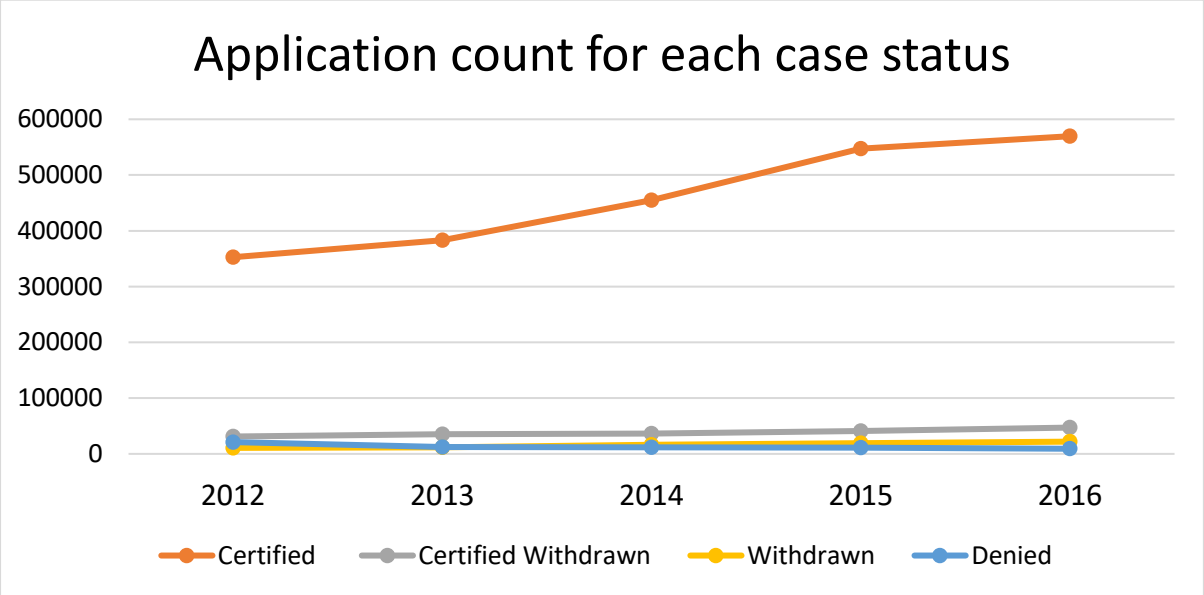
Job 2: Calculates the case status wise count percentage for each year

1. The output of Job 1 is taken as input here.
2. In the map() method, the percentage of application count for each case status is calculated for each year.
3. In the reduce() method, the final output is generated and written as job output.
4. Key: Year
   Value: Application count for each case status along with its percentage

EXPECTED OUTPUT FORMAT:

<year> <certified> <certified-withdrawn> <withdrawn> <denied>
<certified %> <certified-withdrawn %> <withdrawn %> <denied %>

OUTPUT GRAPH:



Query 7

OBJECTIVE: To find the number of applications for each year. Use a bar graph to depict the same

TECHNOLOGY USED: Hive and LibreOffice Calc

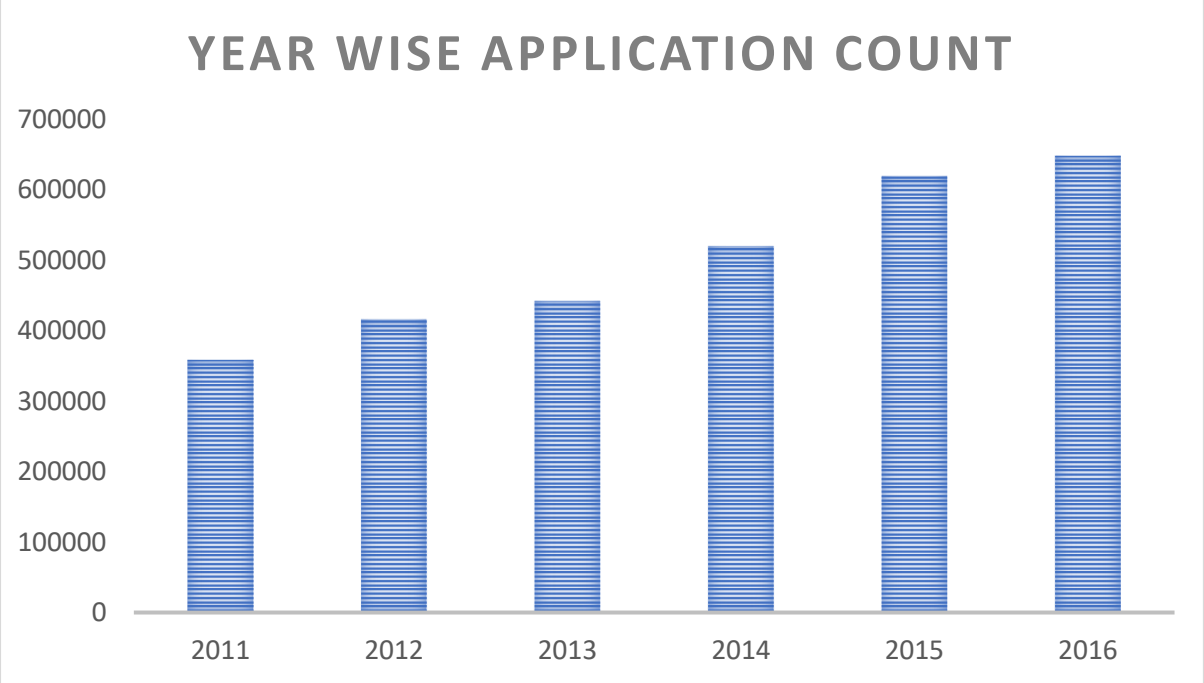
IMPLEMENTATION LOGIC:

1. Extract year and application count [using COUNT()] from the table ‘h1b_app4’
2. Group the data by year
3. Order the data in the ascending order of year value
4. Display the final output using a SELECT query

EXPECTED OUTPUT FORMAT:

<soc name> <application count>

BAR GRAPH:



Query 8

OBJECTIVE: To find the average prevailing wage for each job for each year (take part time and full time separate). Also, Arrange the output in descending order.

TECHNOLOGY USED: MapReduce (in Java)

IMPLEMENTATION LOGIC:

Job 1: Calculates the average prevailing wage for each job title

1. User's year choice and full-time position choice are taken as parameters.
2. H1B data is read one record at a time by the map() method.
3. The job title, year, full-time position and case status of each record are extracted and passed to the Reducer depending on year choice full-time position choice parameters.
4. In the reduce() method, the average prevailing wage for each job title is generated.
5. Key: Job title, Year, Full-time position
   Value: Average prevailing wage

Job 2: Sorts the data in descending order of average prevailing wage

1. The output of Job 1 is taken as input here.
2. In the map() method, the key-value pair is reversed with:

   Key: Average prevailing wage
   Value: Job title, Year, Full-time position

3. In the reduce() method, the records are sorted by average prevailing wage in descending order.
4. Key: Job title, Year, Full-time position
   Value: Average prevailing wage
5. The final output is generated and written as job output.

EXPECTED OUTPUT FORMAT:

<job title> <year> <full time position> <average prevailing wage>

Query 9:

OBJECTIVE: To find all the employers, along with the number of petitions, who have a success rate of more than 70% in petitions (total petitions filed 1000 OR more than 1000)

TECHNOLOGY USED: Pig

IMPLEMENTATION LOGIC:

1. Load the data from /niit_final_project/cleaned_data into bag 'bag1'
2. For each record in 'bag1', extract the columns: case status, employer name, and store in 'bag2'
3. Group the 'bag2' data on employer name
4. Generate the total application count for each employer name and store it in bag 'totalcountbyemp'
5. Similarly, generate the application count for each employer for case status = 'CERTIFIED' and 'CERTIFIED-WITHDRAWN' and store the result in bag 'certcountbyemp' and 'certwithcountbyemp' respectively
6. Join the data from bags 'certcountbyemp', 'certwithcountbyemp' and 'totalcountbyemp' by taking employer name as the common joining key
7. Remove the extra employer name columns to generate bag 'finalempbag' with 6 columns
8. Filter the joined data for total application count >= 1000
9. Calculate the success rate for the filtered data by the below formula:

   Success Rate = (Certified + Certified-withdrawn) / Total * 100

   and store the result in bag 'successrate_bag'
10. Filter 'successrate_bag' for success rate > 70.000
11. Order the filtered data in descending order of success rate.
12. Generate the final output.
13. Store the final result in HDFS

EXPECTED OUTPUT FORMAT:

<employer name> <certified> <certified-withdrawn> <total> <success rate>

Query 10:

OBJECTIVE: To find all the job positions, along with the number of petitions, which have a success rate of more than 70% in petitions (total petitions filed 1000 OR more than 1000)

TECHNOLOGY USED: Pig

IMPLEMENTATION LOGIC:

1. Load the data from /niit_final_project/cleaned_data into bag 'bag1'
2. For each record in 'bag1', extract the columns: case status, job title, and store in 'bag2'
3. Group the 'bag2' data on job title
4. Generate the total application count for each job title and store it in bag 'totalcountbyjob'
5. Similarly, generate the application count for each job title for case status = 'CERTIFIED' and 'CERTIFIED-WITHDRAWN' and store the result in bag 'certcountbyjob' and 'certwithcountbyjob' respectively
6. Join the data from bags 'certcountbyjob', 'certwithcountbyjob' and 'totalcountbyjob' by taking job title as the common joining key
7. Remove the extra job title columns to generate bag 'finaljobbag' with 6 columns
8. Filter the joined data for total application count >= 1000
9. Calculate the success rate for the filtered data by the below formula:

   Success Rate = (Certified + Certified-withdrawn) / Total * 100

   and store the result in bag 'successrate_bag'
10. Filter 'successrate_bag' for success rate > 70.000
11. Order the filtered data in descending order of success rate.
12. Generate the final output.
13. Store the final result in HDFS

EXPECTED OUTPUT FORMAT:

<job title> <certified> <certified-withdrawn> <total> <success rate>

Query 11:

OBJECTIVE: To export the result of Query 10 to MySQL database

TECHNOLOGY USED: Sqoop and MySQL

IMPLEMENTATION LOGIC:

1. Create a database ‘h1b_project’ in MySQL
2. Truncate and drop table ‘query10_output’ if it exists
3. Create a table ‘query10_output’ in the current ‘h1b_project’ database
4. Use the sqoop export command to load the result of query 10 (stored in HDFS) into MySQL table ‘query10_output’
5. Display the contents of table ‘query10_output’ using a SELECT query

STRUCTURE OF TABLE ‘query10_output’:

FIELD	TYPE	NULL	KEY	DEFAULT	EXTRA
job_title	varchar(60)	NO	PRI	NULL	
certified_count	int(11)	NO		NULL	
certified_withdrawn_count	int(11)	NO		NULL	
total_count	int(11)	NO		NULL	
success_rate	float	NO		NULL	

EXPECTED OUTPUT FORMAT:

<job title> <certified> <certified-withdrawn> <total> <success rate>

CONCLUSION

Hadoop is powerful because it is extensible and it is easy to integrate with any component. Its popularity is due in part to its ability to store, analyze and access large amounts of data, quickly and cost-effectively across clusters of commodity hardware. Apache Hadoop is not actually a single product but instead a collection of several components. When all these components are merged, it makes Hadoop very user-friendly.

Hadoop is basically 2 things: Distributed File System (HDFS) + Processing framework (MapReduce)**.**

MapReduce is a computing framework which consists of MapReduce jobs that contain mainly 2 phases: Map phase and Reduce phase. A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then inputted to the reduce tasks. Both the input and the output of a job are stored in HDFS. MapReduce works really well with data that must be represented as a key-value pair and also when aggregation operations are to be performed on it.

Hive provides data warehousing facilities on top of an existing Hadoop cluster. It also provides an SQL like interface which makes work easier, especially if the programmer is from an SQL background. One can create tables in Hive and store data in traditional structured RDBMS way.

Pig, on the other hand, is simply a dataflow language that allows the processing of enormous amounts of data very easily and quickly. Pig basically has 2 parts: Pig Interpreter and the language, PigLatin. Pig makes life a lot easier, especially when step-by-step output is to be found. In fact, one doesn’t require in-depth programming knowledge in order to work with Pig.

Internally, both Hive and Pig queries are converted into MapReduce jobs.

Sqoop is a big data ingestion tool that offers the capability to extract data from non-Hadoop data stores, transform the data into a form usable by Hadoop, and then load the data into HDFS. In short, it is an ETL tool (Extract, Transform, Load).

WEBOGRAPHY

IMAGES:

1. https://s3.amazonaws.com/files.dezyre.com/images/blog/Big+Data+and+Hadoop+Training+Hadoop+Components+and+Architecture_1.png
2. https://www.pinkelephantasia.com/wp-content/uploads/2017/10/Five-Vs-Big-Data.png

LINKS:

1. https://www.guru99.com/what-is-big-data.html
2. https://www.techopedia.com/definition/27745/big-data
3. https://www.digitalocean.com/community/tutorials/hadoop-storm-samza-spark-and-flink-big-data-frameworks-compared
4. https://dzone.com/articles/introduction-to-hadoop-1
5. https://intellipaat.com/tutorial/hadoop-tutorial/introduction-hadoop/
6. https://www.edureka.co/blog/what-is-hadoop/
7. http://bigdata-madesimple.com/why-use-hadoop-top-pros-and-cons-of-hadoop/
8. https://devops.com/bigdata-understanding-hadoop-ecosystem/
9. https://www.techwalla.com/articles/disadvantages-of-a-relational-database