Xiaohong Deng

Big Data MANAGEMENT AND PROCESSING

My Project

# Introduction:

This project aims to establish a robust system capable of efficiently handling and storing data from over half a million houses over the next decade. Considering that each house's annual data is approximately 100MB, the projected data size for the entire project is an astounding 500,000,000 megabytes or 500 terabytes. To meet the challenges posed by such vast datasets, I have chosen to leverage the power of Hadoop technologies.

Key Hadoop components, including HDFS, Flume, MapReduce, Spark, and Hive, have been strategically incorporated into this project's design. This deliberate choice reflects a commitment to harnessing Hadoop's renowned attributes—scalability, cost-effectiveness, speed, fault tolerance, and flexibility. By adopting these technologies, I aim to address the unique requirements of this project, ensuring an effective solution for storing, managing, and analyzing the immense volume of data generated by household electricity consumption.

# Task 1: Load Data to HDFS

## Load historical data to HDFS.

The historical data is around 3 GB, and it is structured data. Hadoop CLI is a straightforward and optimized solutions for moving the historical data to HDFS instead of Flume as Flume is more useful to move real-time streaming data from multiple sources into HDFS.

### Step1 using SSMS to combine historical datasets and split, then store into /energydata/year/month.

--Create new database for my project in SSMS

CREATE DATABASE energydata;

USE energydata;

-- Import Flat File to load data from consumption\_1.txt into the consumption\_1 table

-- Import Flat File to load data from consumption\_6.txt into the consumption\_6 table

-- Import Flat File to load data from consumption\_12.txt into the consumption\_12 table

-- Import Flat File to load data from consumption\_15.txt into the consumption\_15 table

-- Import Flat File to load data from consumption\_16.txt into the consumption\_16 table

-- Import Flat File to load data from consumption\_17.txt into the consumption\_17 table

-- Import Flat File to load data from consumption\_18.txt into the consumption\_18 table

-- Create a new table named UnionTable by combining data from 7 different tables

-- Use SELECT INTO to create UnionTable

SELECT \* INTO UnionTable

FROM (

SELECT \* FROM dbo.consumption\_1

UNION

SELECT \* FROM dbo.consumption\_6

UNION

SELECT \* FROM dbo.consumption\_12

UNION

SELECT \* FROM dbo.consumption\_15

UNION

SELECT \* FROM dbo.consumption\_16

UNION

SELECT \* FROM dbo.consumption\_17

UNION

SELECT \* FROM dbo.consumption\_18

) AS CombinedData;

-- Create a new table named c1\_1 by selecting data from UnionTable

-- where the CONDATE is in January 2015 and HOUSE\_ID is 1

SELECT \*

INTO c1\_1

FROM UnionTable

WHERE CONDATE LIKE '2015-01%'

AND HOUSE\_ID = 1;

-- Download the data from the table c1\_1 to a .txt file on the local machine

-- Use the bcp utility to export data from the c1\_1 table to a text file

bcp energydata.dbo.c1\_1 out C:\Users\Admin\BigDataDegree\myProject\c1\_1.txt -c -U sa -P Sql21rlt@1 -S 10.31.155.223\SQLSERVER1

# Use Hadoop CLI to copy data from the local machine to HDFS

# Copy each file to the corresponding directory on HDFS

for year in {2015..2019}

do

for month in {1..12}

do

hdfs dfs -mkdir -p /energydata/"$year"/"$month"/

for file in /home/hadoopuser/upload-example/energydata/"$year"/"$month"/\*

do

hdfs dfs -copyFromLocal "$file" /energydata/"$year"/"$month"/"${file##\*/}"

done

done

done

## Load real-time data (every 10 seconds) to HDFS

Utilizing Flume to ingest real-time streaming data from various sources into HDFS, as discussed in Task 4.

# Task 2: Use MapReduce to perform the analyses

In this task, MapReduce is employed for analyzing the energy dataset in October 2015. MapReduce serves as a programming model and processing framework specifically designed for parallel and distributed processing of large datasets, facilitating efficient and scalable data analysis across a cluster of computing nodes.

For Task 2\_1, I begin by developing a MapReduce application to determine the hourly electricity consumption for each house and subsequently create another application to find the maximum values.

## 2\_1: Determine the maximum hourly electricity consumption for all the houses across all dates.

### Create a map reduce project – **EnergyHourlyUsage** to calculate the hourly usage for each house

#### MaxMinReading.java

package mapreduce;  
  
import org.apache.hadoop.io.Writable;  
import java.io.DataInput;  
import java.io.DataOutput;  
import java.io.IOException;  
  
public class MaxMinReading implements Writable {  
  
 float maxReading;  
 float minReading;  
  
 public MaxMinReading(){  
 this.maxReading= 0.0f;  
 this.minReading = 0.0f;  
 }  
  
 public MaxMinReading(float val1, float val2) {  
 this.maxReading = val1;  
 this.minReading = val2;  
 }  
  
 public float getMaxReading() {  
 return maxReading;  
 }  
  
 public float getMinReading() {  
 return minReading;  
 }  
  
 @Override  
 public void write(DataOutput dataOutput) throws IOException {  
 dataOutput.writeFloat(maxReading);  
 dataOutput.writeFloat(minReading);  
 }  
  
 @Override  
 public void readFields(DataInput dataInput) throws IOException {  
 maxReading = dataInput.readFloat();  
 minReading = dataInput.readFloat();  
 }  
}

#### ReadingMapper.java

package mapreduce;  
  
import org.apache.hadoop.io.FloatWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;  
  
import java.io.IOException;  
  
public class ReadingMapper extends Mapper<LongWritable, Text, Text, MaxMinReading> {  
 @Override  
 protected void map(LongWritable key, Text value, Mapper.Context context)  
 throws IOException, InterruptedException {  
 String[] values = value.toString().split("\t");  
 String readingbyhousebydatebyhour;  
 float reading;  
 try {  
 readingbyhousebydatebyhour =  
 values[1]+"\t"+values[2]+"\t"+values[3].substring(0,2);  
 reading = Float.*parseFloat*(values[4]);  
 }  
 catch (Exception e){  
 readingbyhousebydatebyhour = "9999";  
 reading = 0.0F;  
 }  
 context.write(new Text(readingbyhousebydatebyhour),  
 new MaxMinReading(reading, reading));  
 }  
}

#### ReadingCombiner.java

package mapreduce;  
  
import mapreduce.MaxMinReading;  
import org.apache.hadoop.io.FloatWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Reducer;  
  
import java.io.IOException;  
  
public class ReadingCombiner extends Reducer<Text, MaxMinReading, Text, MaxMinReading> {  
 @Override  
 protected void reduce(Text key, Iterable<MaxMinReading> maxMinReading,  
 Context context) throws IOException, InterruptedException {  
 float minReading = 999999999F;  
 float maxReading = 0.0F;  
 for (MaxMinReading m: maxMinReading) {  
 minReading = Math.*min*(minReading,m.getMinReading());  
 maxReading = Math.*max*(maxReading,m.getMaxReading());  
 }  
 context.write(key, new MaxMinReading(maxReading,minReading));  
 }  
}

#### ReadingReducer.java

package mapreduce;  
  
import mapreduce.MaxMinReading;  
import org.apache.hadoop.io.\*;  
import org.apache.hadoop.mapreduce.Reducer;  
import java.io.IOException;  
  
  
public class ReadingReducer extends Reducer<Text, MaxMinReading, Text, FloatWritable> {  
  
 @Override  
 protected void reduce(Text key, Iterable<MaxMinReading> maxMinReading,  
 Context context)  
 throws IOException, InterruptedException {  
 float minReading = 999999999F;  
 float maxReading = 0.0F;  
 float usage = 0.0F;  
  
 for (MaxMinReading mm: maxMinReading) {  
 minReading = Math.*min*(minReading, mm.getMinReading());  
 maxReading = Math.*max*(maxReading, mm.getMaxReading());  
 }  
 usage = maxReading - minReading;  
 context.write(key, new FloatWritable(usage));  
 }  
}

#### RunMapReduceJob.java

package mapreduce;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.\*;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
public class RunMapReduceJob {  
 public static void main(String[] args) throws Exception {  
 new RunMapReduceJob().run(args);  
 }  
  
 public void run(String[] args) throws Exception {  
 Path inPath = new Path(args[0]);  
 Path outPath = new Path(args[1]);  
 System.*out*.println(inPath.toString());  
 System.*out*.println(outPath.toString());  
 Configuration conf = new Configuration();  
 Job job = Job.*getInstance*(conf);  
  
 job.setJarByClass(RunMapReduceJob.class);  
 job.setJobName("Usage");  
  
 job.setMapperClass(ReadingMapper.class);  
 job.setCombinerClass(ReadingCombiner.class);  
 job.setReducerClass(ReadingReducer.class);  
 job.setNumReduceTasks(2);  
  
 FileInputFormat.*addInputPath*(job, inPath);  
 FileOutputFormat.*setOutputPath*(job, outPath);  
  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(FloatWritable.class);  
  
 job.setMapOutputKeyClass(Text.class);  
 job.setMapOutputValueClass(MaxMinReading.class);  
  
 System.*exit*(job.waitForCompletion(true) ? 1 : 0);  
 }  
}

### Copy EnergyHourlyUsage.jar into Hadoop

### Execute Jar file in terminal to run MapReduce program and get the successful response

hadoop jar /home/hadoopuser/bigdata-examples/myProject/EnergyHourlyUsage.jar hdfs://192.168.56.5:9820/energydata/2015/10 hdfs://192.168.56.5:9820/myProject/outshourlyenergy

### Check the result of the hourly electricity consumption for all houses in 201510

hdfs dfs -ls /myProject/outshourlyenergy

hdfs dfs -head /myProject/outshourlyenergy/part-r-00000

A screenshot of a computer

Description automatically generated

### Create a map reduce project – EnergyHourlyUsageMax to calculate the max

#### UsageMapper.java

package mapreduce**;**import org.apache.hadoop.io.\***;**import org.apache.hadoop.mapreduce.Mapper**;**import java.io.IOException**;**public class UsageMapper extends Mapper<LongWritable**,** Text**,** Text**,** FloatWritable> {  
 @Override  
 protected void map(LongWritable key**,** Text value**,** Mapper.Context context)  
 throws IOException**,** InterruptedException {  
 String[] values = value.toString().split("\t")**;** String usagebyhousebydatebyhour**;** float usage**;** try {  
 usagebyhousebydatebyhour =values[**0**]**;** usage= Float.*parseFloat*(values[**3**])**;** }  
 catch (Exception e){  
 usagebyhousebydatebyhour = "NA"**;** usage = **0;** }  
 context.write(new Text(usagebyhousebydatebyhour)**,** new FloatWritable(usage))**;** }  
}

#### UsageReducer.java

package mapreduce**;**import org.apache.hadoop.io.FloatWritable**;**import org.apache.hadoop.io.Text**;**import org.apache.hadoop.mapreduce.Reducer**;**import java.io.IOException**;**public class UsageReducer extends Reducer<Text**,** FloatWritable**,** Text**,** FloatWritable> {  
 @Override  
 protected void reduce(Text key**,** Iterable<FloatWritable> usage**,** Context context)  
 throws IOException**,** InterruptedException {  
 float maxUsage= **0;** for (FloatWritable u: usage) {  
 maxUsage= Math.*max*(maxUsage**,** u.get())**;** }  
 context.write(key**,** new FloatWritable(maxUsage))**;** }  
}

#### RunMapReduceJob

package mapreduce**;**import org.apache.hadoop.conf.Configuration**;**import org.apache.hadoop.fs.Path**;**import org.apache.hadoop.io.FloatWritable**;**import org.apache.hadoop.io.Text**;**import org.apache.hadoop.mapreduce.Job**;**import org.apache.hadoop.mapreduce.lib.input.FileInputFormat**;**import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat**;**public class RunMapReduceJob {  
 public static void main(String[] args) throws Exception {  
 new RunMapReduceJob().run(args)**;** }  
  
 public void run(String[] args) throws Exception {  
 Path inPath = new Path(args[**0**])**;** Path outPath = new Path(args[**1**])**;** System.*out*.println(inPath.toString())**;** System.*out*.println(outPath.toString())**;** Configuration conf = new Configuration()**;** Job job = Job.*getInstance*(conf)**;** job.setJarByClass(RunMapReduceJob.class)**;** job.setJobName("Usages")**;** job.setMapperClass(UsageMapper.class)**;** job.setReducerClass(UsageReducer.class)**;** job.setNumReduceTasks(**1**)**;** FileInputFormat.*addInputPath*(job**,** inPath)**;** FileOutputFormat.*setOutputPath*(job**,** outPath)**;** job.setOutputKeyClass(Text.class)**;** job.setOutputValueClass(FloatWritable.class)**;** System.*exit*(job.waitForCompletion(true) ? **1** : **0**)**;** }  
}

### Copy EnergyHourlyUsageMax.jar into Hadoop

### Execute Jar file in terminal to run MapReduce program and get the successful response

hadoop jar /home/hadoopuser/bigdata-examples/myProject/EnergyHourlyUsageMax.jar hdfs://192.168.56.5:9820/myProject/outshourlyenergy/ hdfs://192.168.56.5:9820/myProject/outshourlyenergymax

### Check the result of the maximum hourly electricity consumption for all houses in 201510

hdfs dfs -ls /myProject/outshourlyenergymax

hdfs dfs -cat /myProject/outshourlyenergymax/part-r-00000

A computer screen with numbers and letters

Description automatically generated

# Task 3. Use Apache Hive to perform the analyses

In this task, I leverage Apache Hive to employ HiveQL (the Hive Query Language) for processing and analyzing household electricity consumption meter data in April 2015 and October 2015 due to its ability to provide a high-level, declarative interface for efficient querying and processing of large-scale datasets stored in Hadoop Distributed File System (HDFS).

Initially, I establish an external table and subsequently utilize its query results to create tables tailored to meet the diverse requirements of specific tasks.

## 3\_1: Create a table with each housing unit’s total daily electricity consumption for each of the days in April 2015

### Create and use database

CREATE DATABASE myproject;

USE myproject;

### Check energydata in 201504 created in Task1

A screen shot of a computer

Description automatically generated

### Create external table energy\_ext201504

SET hive.mapred.supports.subdirectories=TRUE;

SET mapred.input.dir.recursive=TRUE;

CREATE EXTERNAL TABLE energy\_ext201504(log\_id INTEGER,house\_id INTEGER,condate DATE ,conhour string,energy\_reading DOUBLE,flag INTEGER)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

LOCATION '/energydata/2015/4/';

A black screen with white text

Description automatically generated

### Check external table energy\_ext201504

select \* from energy\_ext201504 LIMIT 5;

A screen shot of a computer

Description automatically generated

### Create a table task3\_1 with each housing unit’s daily consumption for each of the days in April 2015.

CREATE TABLE task3\_1 AS SELECT HOUSE\_ID, CONDATE, MAX(ENERGY\_READING)-MIN(ENERGY\_READING) AS DAILY\_CONSUMPTION FROM energy\_ext201504 GROUP BY HOUSE\_ID, CONDATE;

SELECT \* FROM task3\_1 WHERE HOUSE\_ID < 7;

A screenshot of a computer

Description automatically generated

## 3\_2: Create a table with each housing unit’s total daily electricity consumption for each of the most recent 30 days. The table should include a column with the corresponding average daily temperature for that housing unit’s region

### Check energydata in 201510 from Task1

A screen shot of a computer

Description automatically generated

### Create external table energy\_ext201510 and check it

SET hive.mapred.supports.subdirectories=TRUE;

SET mapred.input.dir.recursive=TRUE;

CREATE EXTERNAL TABLE energy\_ext201510(log\_id INTEGER,house\_id INTEGER,condate DATE ,conhour string,energy\_reading DOUBLE,flag INTEGER)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

LOCATION '/energydata/2015/10/';

A black background with white text

Description automatically generated

select \* from energy\_ext201510 LIMIT 5;

select \* from energy\_ext201510 where LOG\_ID = 1952158;

*A screen shot of a computer

Description automatically generated*

### Create a table with each housing unit’s daily consumption for each of the MOST RECENT 30DAYS.

/\*

This task only uses 201510 data because there is a memory issue when I am trying all data.

Assuming today is 10/31/2015, so the most recent 30 days is 10/2/2015~10/31/2015

\*/

CREATE TABLE daily\_comsumtion\_most30 AS SELECT HOUSE\_ID, CONDATE, MAX(ENERGY\_READING)-MIN(ENERGY\_READING) AS DAILY\_CONSUMPTION FROM energy\_ext201510

WHERE CONDATE BETWEEN '2015-10-02' AND '2015-10-31'

GROUP BY HOUSE\_ID, CONDATE

ORDER BY CONDATE;

DESCRIBE EXTENDED daily\_comsumtion\_most30;

DESCRIBE daily\_comsumtion\_most30;

SELECT \* FROM daily\_comsumtion\_most30 LIMIT 5;

A screenshot of a computer

Description automatically generated

## 3\_3: Create a table with the average daily electricity consumption for each housing unit for the most recent 30 days collectively. Compare that value to the average for all housing units of that house type.

### Check the table daily\_comsumtion\_most30 from Task3.2

SELECT \* FROM daily\_comsumtion\_most30;

A screenshot of a computer

Description automatically generated

### Create a table with the average daily temperature

CREATE TABLE avg\_consumption\_most30 AS SELECT house\_id, AVG(DAILY\_CONSUMPTION) AS average\_daily\_temperature

FROM daily\_comsumtion\_most30

GROUP BY house\_id;

DESCRIBE EXTENDED avg\_consumption\_most30;

SELECT \* FROM avg\_consumption\_most30;

Text

Description automatically generated

## 3\_4: Create a table with the most recent 30 days’ total electricity consumption by time of day, and includes the fee.

### Check the table energy\_ext201510 from Task3.2

SELECT \* FROM energy\_ext201510 LIMIT 5;

Text

Description automatically generated

### Create a table with time name for different period

CREATE TABLE consumption\_period AS SELECT house\_id, condate,

CASE WHEN CAST(SUBSTRING(conhour,1,2)AS INTEGER)>=19 OR CAST(SUBSTRING(conhour,1,2)AS INTEGER)<7 THEN 1 WHEN CAST(SUBSTRING(conhour,1,2)AS INTEGER)>=7 AND CAST(SUBSTRING(conhour,1,2)AS INTEGER)<11 THEN 2 WHEN CAST(SUBSTRING(conhour,1,2)AS INTEGER)>=11 AND CAST(SUBSTRING(conhour,1,2)AS INTEGER)<17 THEN 3 ELSE 4 END AS period,

energy\_reading

FROM energy\_ext201510;

SELECT \* FROM consumption\_period WHERE HOUSE\_ID = 1 AND PERIOD = 1 LIMIT 5;

A black screen with white text

Description automatically generated

### Create a table with the total consumption by a period of day

CREATE TABLE timely\_consumption AS SELECT house\_id, condate, period, MAX(energy\_reading)-MIN(energy\_reading) AS timely\_consumption

FROM consumption\_period

GROUP BY house\_id, condate, period

ORDER BY house\_id, condate, period;

SELECT \* FROM timely\_consumption LIMIT 5;

SELECT \* FROM timely\_consumption LIMIT 5;Text

Description automatically generated with low confidence

### Create a table with the total consumption Fee by period of day

CREATE TABLE consumption\_period\_fee AS SELECT house\_id, condate, period, timely\_consumption,

CASE WHEN period = 1 THEN timely\_consumption \* 8.2

WHEN period = 2 THEN timely\_consumption \* 11.3

WHEN period = 3 THEN timely\_consumption \* 17.0

WHEN period = 4 THEN timely\_consumption \* 11.3

ELSE timely\_consumption\* 0

END AS Fee

FROM timely\_consumption;

SELECT \* FROM consumption\_period\_fee LIMIT 5;

A screen shot of a computer

Description automatically generated

# Task 4: Design a Flume agent to load the data to HDFS

In this task, I leverage Apache Flume to design a Flume agent and create a Flume flow that partitions the energy data by year and month for Task 4\_1. Task 4\_2 involves configuring a Flume flow to extract records with a specified flag value of 1. These two tasks aim to practice Flume functions using historical data. Flume excels at handling and transporting large volumes of streaming data, making it a suitable choice for the ingestion of real-time data into HDFS.

## 4\_1 Load all the consumption data to HDFS by year and month

### Create the source directory

mkdir -p /home/hadoopuser/bigdata-examples/myProject/consumptions/

### Create the target directory in HDFS

hdfs dfs -mkdir /myProject/energydata

### Edit Configuration file

sudo nano $FLUME\_HOME/conf/myProjectFlume.conf

# Description: Put events in either one of two HDFS sinks based on content of the event

agent1.sources = source1

agent1.sinks = sink-hdfs1

agent1.channels = channel1

# Link the channels to the source: 1 source, 1 channel

agent1.sources.source1.channels = channel1

# Set channel for each of the sinks

agent1.sinks.sink-hdfs1.channel = channel1

# Define the source

agent1.sources.source1.type = spooldir

agent1.sources.source1.spoolDir=/home/hadoopuser/bigdata-examples/myProject/consumptions/

# Define interceptor to extract year, month.

# Reference - regex and serializers: https://flume.apache.org/FlumeUserGuide.html

agent1.sources.source1.interceptors = i1

agent1.sources.source1.interceptors.i1.type = regex\_extractor

agent1.sources.source1.interceptors.i1.regex = (\\d{4})-(\\d{2})

agent1.sources.source1.interceptors.i1.serializers = t1 t2

agent1.sources.source1.interceptors.i1.serializers.t1.name = year

agent1.sources.source1.interceptors.i1.serializers.t2.name = month

# Define the HDFS sink 1

# Reference - escape sequences: https://flume.apache.org/FlumeUserGuide.html

agent1.sinks.sink-hdfs1.type = hdfs

agent1.sinks.sink-hdfs1.hdfs.path = /myProject/energydata/%{year}/%{month}

agent1.sinks.sink-hdfs1.hdfs.filePrefix = c

agent1.sinks.sink-hdfs1.hdfs.fileSuffix = .txt

agent1.sinks.sink-hdfs1.hdfs.rollInterval = 15

agent1.sinks.sink-hdfs1.hdfs.inUsePrefix = \_

agent1.sinks.sink-hdfs1.hdfs.fileType = DataStream

# The channel type file will write data to file for durability and guarantee.

agent1.channels.channel1.type = memory

### Start Flume

*cd $FLUME\_HOME/conf*

*flume-ng agent --conf-file $FLUME\_HOME/conf/*myProjectFlume.conf *--name agent1 --conf $FLUME\_HOME/conf -Dflume.root.logger=INFO,console*

### Result

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

## 4\_2 Load all the records that have flag = 1 to a separate directory.

### Create the source directory

mkdir -p /home/hadoopuser/bigdata-examples/myProject/flag

### Create the target directory in HDFS

hdfs dfs -mkdir /myProject/flag1

### Edit Configuration

sudo nano $FLUME\_HOME/conf/myProjectFlumeFlag.conf

# Description: Put events in either one of two HDFS sinks based on content of the event

agent2.sources = source1

agent2.sinks = sink-hdfs1

agent2.channels = channel1

# Link the channels to the source: 1 source, 1 channel

agent2.sources.source1.channels = channel1

# Set channel for each of the sinks

agent2.sinks.sink-hdfs1.channel = channel1

# Define the source

agent2.sources.source1.type = spooldir

agent2.sources.source1.spoolDir = /home/hadoopuser/bigdata-examples/myProject/flag

# Define interceptor to extract a field named'division'. A serializer will extract the text.

agent2.sources.source1.interceptors = i1

agent2.sources.source1.interceptors.i1.type = regex\_extractor

#regex to pick up the last 1 character

agent2.sources.source1.interceptors.i1.regex = (.{1}$)

agent2.sources.source1.interceptors.i1.serializers = t

agent2.sources.source1.interceptors.i1.serializers.t.name = division

#Selector type = multiplexing allows us to map events

agent2.sources.source1.selector.type = multiplexing

agent2.sources.source1.selector.header = division

agent2.sources.source1.selector.mapping.1 = channel1

# Define the HDFS sink 1

agent2.sinks.sink-hdfs1.type = hdfs

agent2.sinks.sink-hdfs1.hdfs.path = /myProject/flag1

agent2.sinks.sink-hdfs1.hdfs.filePrefix = flag

agent2.sinks.sink-hdfs1.hdfs.fileSuffix = .txt

agent2.sinks.sink-hdfs1.hdfs.rollInterval = 1800

agent2.sinks.sink-hdfs1.hdfs.inUsePrefix = \_

agent2.sinks.sink-hdfs1.hdfs.fileType = DataStream

# The channel type file will write data to file for durability and guarantee.

agent2.channels.channel1.type = memory

### Start Flume

*cd $FLUME\_HOME/conf*

*flume-ng agent --conf-file $FLUME\_HOME/conf/*myProjectFlumeFlag.conf *--name agent2 --conf $FLUME\_HOME/conf -Dflume.root.logger=INFO,console*

### Result

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

# Task 5: Use Spark instead of MapReduce to analyse the data

In this task, I transition from MapReduce to Spark for the analysis of the data previously examined in Task 2. Focusing on a partial dataset from September 2015, the objective is to test the code's functionality using Spark RDD. Furthermore, I will be leveraging Structured Spark Streaming for the real-time analysis of data.

## 5\_1 Determine the maximum hourly electricity consumption for all the houses across all dates using Spark

### Check energydata in 201510 created in Task1

hadoop fs -ls /energydata/2015/10

hadoop fs -head /energydata/2015/10/c12\_10.txt

Graphical user interface

Description automatically generated with medium confidence

### Direct to the spark/bin folder

cd /opt/hadoop/spark/bin

### Start the Spark shell

./spark-shell --master=local

### Import the libraries so that sql can be used within Spark shell

import org.apache.spark.sql.types.\_

import org.apache.spark.sql.Row

### Create RDD from a text file.

val prodFile = "/energydata/2015/10/"

### Create data frame and create the temp collection

val prodFrame= spark.read.format("csv").option("sep", "\t").option("inferSchema", "true").option("header", "false").load(prodFile)

prodFrame.printSchema()

prodFrame.createOrReplaceTempView("energydata201510")

Text

Description automatically generated

### Create query to get the daily electricity consumption for every house in 2015 Oct

val firstSQL = spark.sql("SELECT MAX(hourlyConsumption) FROM (SELECT \_c1 AS house, \_c2 AS day, SUBSTRING(\_c3,0,2) AS hour, MAX(\_c4)-MIN(\_c4) AS hourlyConsumption FROM energydata201510 GROUP BY house, day,hour ORDER BY house, day, hour) AS hourlyData;")

firstSQL.show()

Text

Description automatically generated

## 5\_2 Determine the average daily electricity consumption for every house using Spark

### Check energydata in 201510 created in Task1

hadoop fs -ls /energydata/2015/10

hadoop fs -head /energydata/2015/10/c12\_10.txt

A screen shot of a computer

Description automatically generated

### Direct to the spark/bin folder

cd /opt/hadoop/spark/bin

### Start the Spark shell

./spark-shell --master=local

### Import the libraries so that sql can be used within Spark shell

import org.apache.spark.sql.types.\_

import org.apache.spark.sql.Row

### Create RDD from a text file.

val prodFile = "/energydata/2015/10/"

### Create data frame and create the temp collection

val prodFrame= spark.read.format("csv").option("sep", "\t").option("inferSchema", "true").option("header", "false").load(prodFile)

prodFrame.printSchema()

prodFrame.createOrReplaceTempView("energydata201510")

Text

Description automatically generated

### Create query to get the daily electricity consumption for every house in 2015 Oct

val firstSQL = spark.sql("SELECT AVG(dailyConsumption) FROM (SELECT \_c1 AS house, \_c2 AS day, MAX(\_c4)-MIN(\_c4) AS dailyConsumption FROM energydata201510 GROUP BY house, day ORDER BY house, day) AS dailyData GROUP BY house;")

firstSQL.show()

Text

Description automatically generated

## 5\_3 Create the daily electricity consumption for every house in 2015 Apr using Spark

### Check energydata in 201504 created in Task1

Graphical user interface, text

Description automatically generated

### Direct to the spark/bin folder

cd /opt/hadoop/spark/bin

### Start the Spark shell

./spark-shell --master=local

### Import the libraries so that sql can be used within Spark shell

import org.apache.spark.sql.types.\_

import org.apache.spark.sql.Row

### Create RDD from a text file.

val prodFile = "/energydata/2015/4/"

### Create data frame and create the temp collection

val prodFrame= spark.read.format("csv").option("sep", "\t").option("inferSchema", "true").option("header", "false").load(prodFile)

prodFrame.printSchema()

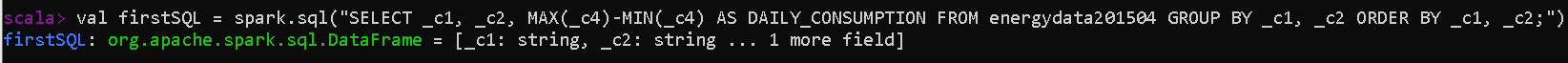
prodFrame.createOrReplaceTempView("energydata201504")

Text

Description automatically generated

### Create query to get the daily electricity consumption for every house in 2015 Apr

val firstSQL = spark.sql("SELECT \_c1, \_c2, MAX(\_c4)-MIN(\_c4) AS DAILY\_CONSUMPTION FROM energydata201504 GROUP BY \_c1, \_c2 ORDER BY \_c1, \_c2;")



firstSQL.show()

A picture containing text

Description automatically generated

## 5\_4 Using structured Spark to real-time track the data with FLAG 1 for each HOUSE\_ID

### Create a source directory

mkdir -p /home/hadoopuser/bigdata-examples/myProject/structuredSpark

### Importing library and starting spark session

//Start Spark

cd /opt/hadoop/spark/bin

./spark-shell --master=local

// Import libraries

import org.apache.spark.sql.functions.\_

import org.apache.spark.sql.SparkSession

import spark.implicits.\_

import org.apache.spark.sql.types.\_

//input data

val spark=SparkSession.builder() .appName("StructuredSparkStreaming") .getOrCreate()

//define the schema before creating the dataframe. can’t use inferschema

val schema = StructType(Array(

StructField("LOG\_ID", StringType, nullable = true),

StructField("HOUSE\_ID", IntegerType, nullable = true),

StructField("CONDATE", StringType, nullable = true),

StructField("CONHOUR", StringType, nullable = true),

StructField("ENERGY\_READING", FloatType, nullable = true),

StructField("FLAG", IntegerType, nullable = true)

))

//load the data files from the given directory

val df=spark.readStream.format(source="CSV").option("header",false).option("delimiter","\t")

.option("inferSchema","false").schema(schema).load("file:///home/hadoopuser/bigdata-examples/myProject/structuredSpark/")

df.createOrReplaceTempView("MyTable")

val flag1 = spark.sql("""SELECT CONDATE, HOUSE\_ID, COUNT(FLAG) AS FLAG\_COUNT FROM MyTable WHERE FLAG = 1 GROUP BY CONDATE,HOUSE\_ID""")

// note that we are choosing to show the output in the console and choosing the append mode.

val query= flag1.writeStream.outputMode("complete").format("console")

.option("checkpointLocation","file:///home/hadoopuser/bigdata-examples/myProject/structuredSpark/\_checkpoint").start()

### Drag files to the source directory one by one and real-time observe the output

A screenshot of a computer

Description automatically generated

# Critical thinking section

## 5\_1. Hadoop

Why did I choose Hadoop? Hadoop was selected as the foundation for this project due to its capabilities in handling large-scale distributed data processing. Its distributed file system, HDFS, provides fault tolerance and scalability, making it ideal for storing and managing vast amounts of structured and unstructured data.

## 5\_2. HADOOP CLI

For Task 1, why did I choose HADOOP CLI to upload the historical datasets to HDFS but not FLUME? The decision to use Hadoop CLI for uploading historical datasets to HDFS was driven by the simplicity and efficiency it offers for batch processing. Hadoop CLI provides a straightforward command-line interface for moving large amounts of data, making it well-suited for one-time bulk transfers. Flume, on the other hand, is designed for real-time streaming data ingestion, making it less optimal for the one-time upload of historical data.

## 5\_3. MapReduce

MapReduce was chosen for Task 2 due to its effectiveness as a programming model and processing framework for parallel and distributed processing. Its ability to scale horizontally across a cluster of computing nodes makes it suitable for analyzing large datasets, such as the energy dataset in October 2015.

## 5\_4. Hive

For Task 3, why did I choose Hive? Apache Hive was selected for Task 3 because of its high-level, declarative interface (HiveQL) for efficient querying and processing of large-scale datasets. It simplifies the analysis of household electricity consumption meter data in April 2015 and October 2015, offering a convenient way to interact with data stored in HDFS.

## 5\_5. Flume

Apache Flume was utilized in Task 4 to design a Flume agent and create a Flume flow for handling and transporting large volumes of streaming data. Flume excels at ingesting real-time data into HDFS, making it a suitable choice for partitioning energy data by year and month and extracting records with specific flag values.

## 5\_6. Spark

The transition from MapReduce to Spark for Task 5, specifically using Spark RDD for batch processing and Structured Spark Streaming for real-time analysis, was motivated by Spark's superior performance and ease of use. Spark processes data in-memory, resulting in faster computations compared to the disk-based processing of MapReduce. Additionally, Spark provides a unified and more user-friendly API for both batch and streaming processing, enhancing overall productivity and code maintainability. This shift reflects the industry trend towards adopting Spark as a more efficient and versatile alternative to MapReduce.

In this project, I had the opportunity to practice many big data software tools, such as Hadoop CLI, Hive, MapReduce, Flume, Spark, etc. I can understand what they are, why I need them, and how I can use them. It would be useful for me to apply this knowledge in my future job.

# Reference

A. L. (2021, October 6). *How to plan the capacity of a Hadoop cluster?* Edureka. Retrieved October 28, 2021, from <https://www.edureka.co/blog/hadoop-cluster-capacity-planning/#factor>

Chawla, M. (2017, July 30). *Hadoop cluster capacity planning of data nodes for batch and in-memory processes - dzone big data*. dzone.com. Retrieved October 28, 2021, from <https://dzone.com/articles/hadoop-cluster-capacity-planning>

Makonin, Stephen, 2018, "HUE: The Hourly Usage of Energy Dataset for Buildings in British

Columbia", https://doi.org/10.7910/DVN/N3HGRN, Harvard Dataverse, V5,

UNF:6:F2ursIn9woKDFzaliyA5EA== [fileUNF]

Siva, Technical strengths include Hadoop, View all posts by Siva →, 29, V., Siva Post author February 29, . . . 7, R. (2015, February 26). *Formula to calculate HDFS nodes storage*. Retrieved October 28, 2021, from <http://hadooptutorial.info/formula-to-calculate-hdfs-nodes-storage/>

Stephen Makonin, 2019, “HUE: The Hourly Usage of Energy Dataset for Buildings in British

Columbia,” Data in Brief, vol. 23, no. 103744, pp. 1-4 (2019).

Thinkpad X1 Carbon (6 gen) platform specifications - lenovo. (n.d.). Retrieved October 28, 2021, from <https://psref.lenovo.com/syspool/Sys/PDF/ThinkPad/ThinkPad_X1_Carbon_6th_Gen/ThinkPad_X1_Carbon_6th_Gen_Spec.PDF>