

Implementing Hadoop Systems Case Study: Revisited

A Data Analysis Case Study at ElectroNova Energy Solutions

December 2021
(revisited July 2024)

Abdulrahman Khan, Nathan Hill

Table Of Contents

Table Of Contents	1
Executive Summary	2
Dataset Overview	2
Requirements	4
System Design Methodology	4
Methodology	4
Data Onboarding	5
Data Analysis: MapReduce Analysis	7
Data Analysis: HiveQL Queries Analysis	9
Data Analysis: Spark Analysis	12
Data Pipelining: Flume Configuration	13
Data Pipelining: Sqoop Export	17
Data Analysis: MySQL Analysis	19
Conclusion and Summary	20
References	20

Executive Summary

This case study details the development and implementation of a proof of concept solution designed to meet ElectroNova's Big Data storage and analysis needs. Our team has crafted a cutting-edge solution by leveraging a range of technologies from the Hadoop ecosystem. This report covers the necessary hardware specifications, including required disk space and recommended server configurations for effective data analysis. Additionally, the report provides an in-depth look at the software solutions integrated into the proof of concept and includes examples of data analysis conducted using these technologies.

Introduction

ElectroNova is a leading company in household electricity management, dedicated to innovation and sustainability. For over 20 years, ElectroNova has been providing essential electricity solutions that power the lives and businesses of millions of consumers across Ontario. Their commitment to excellence ensures that their services are integral to the daily lives of individuals and organizations throughout the region.

The company aims to enhance the storage and analysis of household electricity consumption data and is embarking on a project to explore big data technologies. To address their data requirements, a proof-of-concept solution is necessary. This report outlines the project requirements and provides a detailed analysis of the recommended solutions. It also describes the processes implemented in the proof-of-concept solution and how these align with the company's business needs.

Dataset Overview

Log from Hydro Meters

ElectroNova captures data from smart meters installed in every home it services. These meters record cumulative electricity consumption data, commonly referred to as meter readings. Previously, the company would send a representative to each residence monthly to manually read and record these readings. This process was time-consuming, resource-intensive, and prone to errors and missed readings.

To improve efficiency, ElectroNova has now implemented smart meters that electronically transmit meter readings to the company's database every 10 seconds. The metering system outputs the data as text files in comma-separated values (CSV) format. Each

household's meter reading log data is contained in a separate file. Energy reading is in kilowatt-hours (kWh) and it is cumulative.

Log_ID	House_ID	ConDate	ConTime	Energy_reading	Flag
1	9	15-05-30	00:00:00	11000.001444	0
2	9	15-05-30	00:00:10	11000.002888	0
3	9	15-05-30	00:00:20	11000.004332	0
4	9	15-05-30	00:00:30	11000.005776	0
5	9	15-05-30	00:00:40	11000.00722	0
6	9	15-05-30	00:00:50	11000.008664	0
7	9	15-05-30	00:01:00	11000.010108	0
8	9	15-05-30	00:01:10	11000.011552	0
9	9	15-05-30	00:01:20	11000.012996	0
10	9	15-05-30	00:01:30	11000.01444	0
11	9	15-05-30	00:01:40	11000.015884	0
12	9	15-05-30	00:01:50	11000.017328	0
13	9	15-05-30	00:02:00	11000.018772	0
14	9	15-05-30	00:02:10	11000.020216	0

Weather Information

ElectroNova has access to a database that maintains hourly records of weather information. This data is stored in a table named "HourlyWeather," which is updated on an hourly basis. Sample extracts of weather data, categorized by location, are provided for inclusion in the new system design. Each record in the table includes the date, hour of recording, temperature, humidity, barometric pressure, and a description of the conditions.

date	hour	temperature	humidity	pressure
2015-01-01	01	0.7	88	102.31
2015-01-01	02	1.0	88	102.26
2015-01-01	03	0.7	90	102.22
2015-01-01	04	0.6	90	102.18
2015-01-01	05	-0.4	91	102.11
2015-01-01	06	-1.0	92	102.05
2015-01-01	07	0.2	91	102.03
2015-01-01	08	0.3	92	101.96
2015-01-01	09	-0.1	91	101.96

House Information

ElectroNova has a database with basic housing information on the houses it services.

	A	B	C	D	E	F	G	H	I	J	K
1	House	FirstReading	LastReading	Cover House	Facing	Region	RUs	EVs	SN	HVAC	
2	1	2012-06-01	2015-10-03	1	bungalow	South	YVR		1	1	FAGF + FPG + HP
3	2	2016-06-09	2019-11-20	0.994	duplex	North	YVR		0	2	IFRHG + FPG NAC
4	3	2015-01-27	2018-01-29	0.987	modern	South	YVR		2	IFRHG + 1 B	NAC
5	4	2015-01-30	2018-01-29	0.995	character	West	YVR		1	FAGF + IFRH	NAC
6	5	2015-01-30	2018-01-29	0.995	modern	South	YVR		1	IFRHG	NAC
7	6	2015-01-30	2018-01-29	0.997	apartment	SW	YVR		0	BHE	NAC
8	7	2015-01-30	2018-01-29	0.997							
9	8	2015-02-21	2018-02-20	0.987	character	South	YVR		0	FAGF	PAC
10	9	2015-05-01	2018-02-21	0.996	special	South	YVR		0	IFRHG + FPG	NAC
11	10	2015-02-21	2018-02-20	0.995	special	South	YVR		0	FAGF	NAC
12	11	2015-02-21	2018-02-20	0.99	duplex	North	YVR		0	FAGF + IFRH	NAC
13	12	2015-02-21	2018-02-20	0.992	apartment	NW	YVR		0	IFRHG	NAC
14	13	2015-02-21	2018-02-20	0.998	special	North	YVR		1	FAGF	NAC

Requirements

ElectroNova requires a proof-of-concept system design that will securely store company data and facilitate timely data analysis functions. The company seeks to determine the necessary storage capacity in megabytes (MB) and the server nodes required to execute analysis functions on their data.

The system must be capable of storing data for **500,000 housing units over the next 15 years** and supporting comprehensive **analysis of the entire dataset**. For the proof of concept, ElectroNova wishes to evaluate the system's capability to generate aggregate consumption data for the past day and month, as well as to identify and flag any errors in meter logs in near real-time.

System Design Methodology

Methodology

The research and design team convened to identify an optimal solution to meet the company's needs and ultimately selected a suite of Hadoop technologies. We will leverage the Hadoop Distributed File System (HDFS) for reliable data storage and utilize MapReduce programs for comprehensive data analysis across the entire dataset. Additionally, Apache Hive will be implemented to facilitate the execution of HiveQL queries on the data. To enhance real-time data collection, Apache Flume will be integrated into the system to continuously load electricity consumption data into our database.

Proof of Concept Data Subset

Due to the constraints of the cluster provided for the proof-of-concept design, the system design team has opted to work with a subset of the available data. This subset, selected to prevent overloading the temporary cluster, is designed to accurately represent the company's overall dataset while reducing the volume of data processed. The subset includes data for all houses, focusing exclusively on the months of June and December each year. It is further restricted to specific days: the 6th, 12th, 18th, 24th, and 30th of each month. This approach provides approximately 10 days of data per year for each house.

To facilitate data organization, a Java program was developed during the initial onboarding phase. This program verifies whether the data aligns with the subset criteria and organizes it into the appropriate file structure.

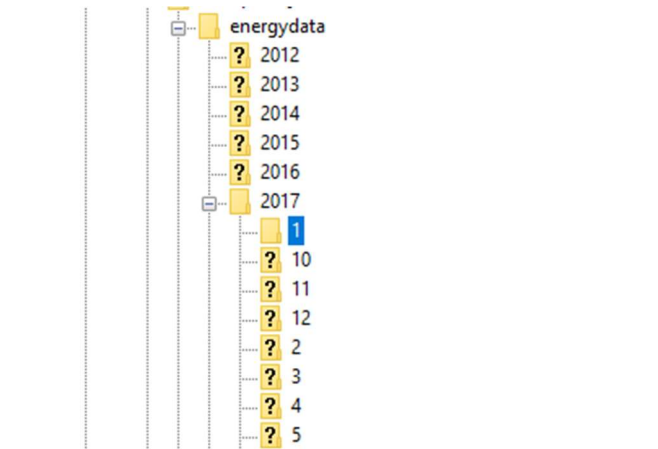
```
// 5 days a month and only 2 months a year per house  
if (Integer.parseInt(day) % 6 == 0 && Integer.parseInt(month) % 6 == 0) {
```

By working with this subset, the cluster will be better equipped to handle the data analysis tasks, thus accelerating project progress.

Data Onboarding

The programs for this analysis can be found in the Task11 project file, in this repository

To onboard the data, the system design team developed a Java program to organize the housing data into a structured format: energydata/year/month/day.csv. This program extracts relevant data from the dataset, verifies its placement within the designated directory, and categorizes it into the appropriate folder within the energydata directory structure. The final organized data, produced by this program, was subsequently uploaded to our server cluster.



Filename	Filesize	Filetype	Last modified
..			
12.txt	12,242,125	TXT File	2021-10-27 10:25:04 PM
15.txt	12,243,823	TXT File	2021-10-27 10:25:19 PM
17.txt	11,966,666	TXT File	2021-10-27 10:25:24 PM
18.txt	12,242,261	TXT File	2021-10-27 10:25:39 PM
6.txt	11,967,601	TXT File	2021-10-27 10:25:43 PM

System Design

To estimate the data requirements for our system design, we start with the dataset contained in **Consumption_1.txt**, which includes data for one household from June 2012 to September 2015, spanning 39 months. The file size for this period is 459 MB (481,337,183 bytes). Based on this, we estimate the data requires approximately 12 MB of disk space per month or 141 MB per year.

Given that our design must support 500 households over 14 years, we calculate the total disk space required as follows: multiplying the annual disk space (141 MB) by 14 years yields 1,974 MB per household. For 500 households, this totals approximately 987,000 MB, or roughly 1 TB.

To account for data redundancy, we apply a replication factor of 3, resulting in a total of 2,961,000 MB, or approximately 3 TB. Adding a 20% buffer for HDFS overhead, the final estimated disk space required is 3.6 TB.

The Hadoop Distributed File System (HDFS) will be configured with 4 data nodes and 1 name node to manage this storage.

Data Analysis: MapReduce Analysis

The programs for this analysis can be found in the Task2 project file, in this repository

Maximum Hourly Electricity Consumption

To determine the maximum hourly electricity consumption for each house, we will employ two MapReduce programs, each with its own reducer function. By dividing the data analysis into two separate tasks, we can effectively use key-value pairs to maintain data integrity throughout the process. This approach will enable us to compare values and identify the maximum hourly consumption for each house accurately.

The first Mapper and Reducer files collect the Energy Consumption for a singular day. This is achieved by using the key pair value of the house ID, the date and the hour at which the reading was taken.

```
houseByDateByHour = values[1] + "\t" + values[2] + "\t" + values[3].substring(0, 2);
```

When we get to the reducer, we need to combine the values to get the Energy Consumption for that specific hour. Since the energy reading values are cumulative, we will need to subtract the previous intervals reading with the current one. To achieve this, we utilize a CustomWriter class to hold the previous reading's data in order to subtract the two.

```
hourElectricConsumption += (cw.getElectricConsumption() - amount.getElectricConsumption());  
cw = new CustomWritable(amount.getElectricConsumption());
```

The second Mapper and Reducer takes the output from the first MapReduce program, which looks like this.

12	2016-06-06	05	11279.416
12	2016-06-06	07	11280.166
12	2016-06-06	09	11280.926
12	2016-06-06	10	11281.416
12	2016-06-06	12	11282.106
12	2016-06-06	14	11282.926
12	2016-06-06	16	11283.726
12	2016-06-06	18	11284.486
12	2016-06-06	21	11285.706
12	2016-06-06	23	11286.376

The mapper for the second program assigns a key value of the house ID. The reducer in this program looks at the values and assigns the largest value it can find to the key value pair, using the Math.max() method.


```
hadoopuser@hd-master:~$ hadoop fs -cat /EnergyData2_5/part-r-00000
1      34844.547
12     18243.688
16     11106.81
18     45588.117
```

Using these two programs, we can calculate the maximum daily electricity consumption for each house.

Average Daily Electricity Consumption for Every House

The MapReduce program for this data analysis is like the previous one. It will require two MapReduce programs.

The first Mapper and Reducer program will collect the data and find the daily consumption for every house. The mapper will use a key value of the house ID and the date.

```
houseByDate = values[1] + "\t" + values[2];
```

Since we are calculating the consumption for the day, the Reducer will utilize a Custom Writer class which will handle calculating the average Consumption.

The second Mapper and Reducer will collect the output from the first MapReduce program and calculate the average daily consumption for each house.

```
hadoopuser@hd-master:~$ hadoop fs -ls /DailyData3
Found 3 items
-rw-r--r--  3 hadoopuser supergroup          0 2021-11-09 04:10 /DailyData3/_SUCCESS
-rw-r--r--  3 hadoopuser supergroup    2292 2021-11-09 04:10 /DailyData3/part-r-00000
-rw-r--r--  3 hadoopuser supergroup    2285 2021-11-09 04:10 /DailyData3/part-r-00001
hadoopuser@hd-master:~$ hadoop fs -cat /DailyData3/part-r-00000
1      2012-06-06      4.5992204E7
1      2012-06-24      4.9176736E7
1      2012-12-12      8.7945808E7
1      2012-12-18      8.9382768E7
1      2012-12-30      9.2319304E7
1      2013-06-12      1.33490872E8
1      2013-06-18      1.34757312E8
```

To do this, the Mapper will assign the Key value of the house ID. Since we are calculating the average consumption, the reducer will keep a counter variable which will be used to calculate the average consumption over the days. The Reducer class will then assign the average daily consumption to the house ID key value.

Data Analysis: HiveQL Queries Analysis

The HiveQL queries were executed entirely within the HDFS file system and the Hive environment. This process involved using SQL statements to create the necessary tables and construct the queries needed to obtain the requested data.

The initial task required the creation of an external table to manage data from April 2015, which will serve as the primary dataset for subsequent queries. Documentation of this process includes both screenshots and SQL queries.

Creation of external table for initial data

```
hive> CREATE EXTERNAL TABLE EnergyData1(LOG_ID INTEGER,  
  > HOUSE_ID INTEGER,  
  > CONDATE DATE,  
  > CONHOUR VARCHAR(15),  
  > ENERGY_READING DOUBLE,  
  > FLAG INTEGER)  
  > ROW FORMAT DELIMITED  
  > FIELDS TERMINATED BY ','  
  > LINES TERMINATED BY '\n'  
  > LOCATION '/GroupProject/energydata/2015/4';  
OK  
Time taken: 0.66 seconds  
hive> DESCRIBE EnergyData1;  
OK  
log_id          int  
house_id        condint  
condate         date  
conhour         varchar(15)  
energy_reading  double  
flag            int  
Time taken: 0.181 seconds, Fetched: 6 row(s)  
hive> _
```

Creating a table to hold the sum data

```
hive> CREATE TABLE EnergyDataSum(LOG_ID INTEGER,
> HOUSE_ID INTEGER,
> CONDATE DATE,
> CONHOUR VARCHAR(15),
> ENERGY_READING DOUBLE,
> FLAG INTEGER)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> LINES TERMINATED BY '\n'
> STORED AS TEXTFILE;
```

OK

Time taken: 0.923 seconds

```
hive> show tables;
```

OK

energydata1

energydatasum

Time taken: 0.162 seconds, Fetched: 2 row(s)

```
hive> DESCRIBE energydatasum;
```

OK

```
log_id          int
house_id        int
condate         date
conhour         varchar(15)
energy_reading  double
flag           int
```

Time taken: 0.155 seconds, Fetched: 6 row(s)

```
hive>
```

Populating a partition in the new table with daily data from the external table

```
hive> INSERT INTO TABLE energydatasum PARTITION (conhour = '23:59:50') SELECT log_id, house_id, condate, energy_reading,
Query ID = hadoopuser_20211108211630_856d2876-36c5-45d0-88c3-d6394517e82c
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1631646730601_0005, Tracking URL = http://hd-master:8088/proxy/application_1631646730601_0005/
Kill Command = /opt/hadoop/bin/mapred job -kill job_1631646730601_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2021-11-08 21:16:59,945 Stage-1 map = 0%, reduce = 0%
2021-11-08 21:17:26,720 Stage-1 map = 26%, reduce = 0%, Cumulative CPU 13.24 sec
2021-11-08 21:17:31,409 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 19.09 sec
MapReduce Total cumulative CPU time: 19 seconds 280 msec
Ended Job = job_1631646730601_0005
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://hd-master:9820/user/hive/warehouse/groupproject.db/energydatasum/conhour=23%3A59%3A50/.hi
Loading data to table groupproject.energydatasum partition (conhour=23:59:50)
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 19.28 sec HDFS Read: 45342314 HDFS Write: 73 SUCCESS
Total MapReduce CPU Time Spent: 19 seconds 280 msec
OK
Time taken: 67.184 seconds
```

Average consumption across all houses in the most recent 30 days

The average in the last 30 days by house is:

House 1: 33019.14

House 2: 9063.07

House 12: 5853.29

House 18: 7562.65

```
OK
1      33019.14320047042
6      9063.068721824913
12     5853.290986933939
18     7562.645754745034
```

Total electricity consumption in the most recent 30 days by time of day

```
Total MapReduce CPU Time Spent: 1 minutes 40 seconds 680 msec
OK
1      2.140750192355158E9
6      5.875727484450114E8
12     3.795090006584333E8
18     4.904616774939297E8
Time taken: 554.853 seconds, Fetched: 4 row(s)
hive> _
```

Consumption during mid peak 7am to 11am and 5pm to 7pm

```
Total MapReduce CPU Time Spent: 1 minutes 55 seconds 90 msec
OK
1      1.4272413426915448E9
6      3.917419364968621E8
12     2.5295743285974312E8
18     3.265941245589079E8
Time taken: 413.07 seconds, Fetched: 4 row(s)
```

Consumption between 5pm to 7pm only

```
Total MapReduce CPU Time Spent: 1 minutes 33 seconds 450 msec
OK
1      7.143456479066855E8
6      1.9606817067949444E8
12     1.2666940594105265E8
18     1.638186975911911E8
Time taken: 407.034 seconds, Fetched: 4 row(s)
```

Data Analysis: Spark Analysis

We demonstrated Spark's capabilities by re-executing the MapReduce programs using Spark. Initially, we created a subset of data from one month of the year 2012. This subset was then utilized to generate various Resilient Distributed Datasets (RDDs), which were structured and reduced to include only the necessary column data pairs. Finally, we conducted the analysis and presented the results to showcase Spark's effectiveness in managing and analyzing the data.

Actions performed to run the analysis

```
scala> val lines = sc.textFile("/sparkWork/energydata2/2012/6/1.txt")
lines: org.apache.spark.rdd.RDD[String] = /sparkWork/energydata2/2012/6/1.txt MapPartitionsRDD[1] at textFile at <console>:24
```

RDD 1 is created from our data subset.

```
scala> val records = lines.map(In => In.split("\t"))
records: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[2] at map at <console>:25
```

RDD 2 is created to sort the data into columns

```
scala> val pairs = records.map(rec => try{(rec(1).toInt + "," + rec(2).toString, rec(4).toFloat)} catch { case e1: Exception => ("0", 0.0f)})
pairs: org.apache.spark.rdd.RDD[(String, Float)] = MapPartitionsRDD[7] at map at <console>:25
```

Reducing data to only required pairs

```
scala> val mean = pairs.reduceByKey((x,y) => (x+y)/2)
mean: org.apache.spark.rdd.RDD[(String, Float)] = ShuffledRDD[9] at reduceByKey at <console>:25
scala> mean.foreach(println(_))
```

Reducing to find mean

```
scala> val maxElec = pairs.reduceByKey((x,y) => Math.max(x,y))
maxElec: org.apache.spark.rdd.RDD[(String, Float)] = ShuffledRDD[10] at reduceByKey at <console>:25
scala> maxElec.foreach(println(_))
```

Average energy consumed per house

```
scala> mean.foreach(println(_))
(1,2012-06-24,5689.4297)
(0,0.0)
(1,2012-06-30,5821.249)
(1,2012-06-06,5317.8086)
(1,2012-06-18,5573.5376)
(1,2012-06-12,5455.492)
```


Data Pipelining: Flume Configuration

Here is our Flume Flow configuration file. This file reads files from a spoolDir directory. It goes through line by line and uses regex interceptors to determine the timestamp and the flag. It uses this information to place the output in its respective HDFS directory.

```
agent1.sources = source1
agent1.sinks = sinkhdfs1 sinkhdfs2 sinkhdfs3
agent1.channels = channel1 channel2 channel3

# LINK FOR SOURCE AND SINK
agent1.sources.source1.channels = channel1 channel2 channel3
agent1.sinks.sinkhdfs1.channel = channel1
agent1.sinks.sinkhdfs2.channel = channel2
agent1.sinks.sinkhdfs3.channel = channel3

# Define the source
agent1.sources.source1.type = spooldir
agent1.sources.source1.spoolDir = /home/hadoopuser/group_project/flumeSource

# Define interceptor to extract
agent1.sources.source1.interceptors = i1 i2
agent1.sources.source1.interceptors.i1.type = regex_extractor
agent1.sources.source1.interceptors.i2.type = regex_extractor
# timestamp
agent1.sources.source1.interceptors.i1.regex = ^?(\\d\\d\\d\\d\\d\\d-\\d\\d-\\d\\d)
agent1.sources.source1.interceptors.i1.serializers = tt
agent1.sources.source1.interceptors.i1.serializers.tt.type=
org.apache.flume.interceptor.RegexExtractorInterceptorMillisSerializer
agent1.sources.source1.interceptors.i1.serializers.tt.name = timestamp
agent1.sources.source1.interceptors.i1.serializers.tt.pattern = yyyy-MM-dd
# flag
agent1.sources.source1.interceptors.i2.regex = (.{1}$)
agent1.sources.source1.interceptors.i2.serializers = flag
agent1.sources.source1.interceptors.i2.serializers.flag.name = flag

agent1.sources.source1.selector.type = multiplexing
agent1.sources.source1.selector.header = flag
agent1.sources.source1.selector.mapping.0 = channel1 channel2
```

```

agent1.sources.source1.selector.mapping.1 = channel3
# yyyyymmddhhmmss
# consumption_6_20200101235850
# SINK 1 - YEAR MONTH
agent1.sinks.sinkhdfs1.type = hdfs
agent1.sinks.sinkhdfs1.hdfs.path = /group_project/YearMonth/%Y/%m
agent1.sinks.sinkhdfs1.hdfs.filePrefix = consumption
agent1.sinks.sinkhdfs1.hdfs.fileSuffix = .txt
agent1.sinks.sinkhdfs1.hdfs.rollInterval = 0
agent1.sinks.sinkhdfs1.hdfs.rollCount = 0
agent1.sinks.sinkhdfs1.hdfs.rollSize = 1000000
agent1.sinks.sinkhdfs1.hdfs.inUsePrefix = _
agent1.sinks.sinkhdfs1.hdfs.fileType = DataStream
# SINK 2 - YEAR MONTH DAY
agent1.sinks.sinkhdfs2.type = hdfs
agent1.sinks.sinkhdfs2.hdfs.path = /group_project/YearMonthDay/%Y/%m/%d
agent1.sinks.sinkhdfs2.hdfs.filePrefix = consumption
agent1.sinks.sinkhdfs2.hdfs.fileSuffix = .txt
agent1.sinks.sinkhdfs2.hdfs.rollInterval = 0
agent1.sinks.sinkhdfs2.hdfs.rollCount = 0
agent1.sinks.sinkhdfs2.hdfs.rollSize = 1000000
agent1.sinks.sinkhdfs2.hdfs.inUsePrefix = _
agent1.sinks.sinkhdfs2.hdfs.fileType = DataStream
# SINK - FLAGGED
agent1.sinks.sinkhdfs3.type = hdfs
agent1.sinks.sinkhdfs3.hdfs.path = group_project/Flagged
agent1.sinks.sinkhdfs3.hdfs.filePrefix = consumption
agent1.sinks.sinkhdfs3.hdfs.fileSuffix = .txt
agent1.sinks.sinkhdfs3.hdfs.rollInterval = 0
agent1.sinks.sinkhdfs3.hdfs.rollCount = 0
agent1.sinks.sinkhdfs3.hdfs.rollSize = 1000000
agent1.sinks.sinkhdfs3.hdfs.inUsePrefix = _
agent1.sinks.sinkhdfs3.hdfs.fileType = DataStream

agent1.channels.channel1.type = memory
agent1.channels.channel2.type = memory
agent1.channels.channel3.type = memory

```

The consumption data is loaded into these three HDFS folders.

- ## Running the flume config

15

Inside YearMonth/

```
hadoopuser@hd-master:~$ hadoop fs -ls -R /group_project/YearMonth
drwxr-xr-x  - hadoopuser supergroup          0 2021-12-11 22:46 /group_project/YearMonth/2016
drwxr-xr-x  - hadoopuser supergroup          0 2021-12-11 22:52 /group_project/YearMonth/2016/06
-rw-r--r--  3 hadoopuser supergroup        29125 2021-12-11 22:47 /group_project/YearMonth/2016/06/consumption.1639262797501.txt
-rw-r--r--  3 hadoopuser supergroup        29125 2021-12-11 22:52 /group_project/YearMonth/2016/06/consumption.1639263101018.txt
```

Inside YearMonthDay/

[illegible]

Inside Flagged/

```
hadoopuser@hd-master:~$ hadoop fs -ls /group_project/Flagged
Found 1 items
-rw-r--r--   3 hadoopuser supergroup          275 2021-12-11 22:55 /group_project/Flagged/_consumption.1639263303146.txt.tmp
hadoopuser@hd-master:~$ hadoop fs -cat /group_project/Flagged/_consumption.1639263303146.txt.tmp
4176651 12      2015-06-18      10:48:20      11413.737256   1
4176652 12      2015-06-18      10:48:30      11413.740312   1
4176653 12      2015-06-18      10:48:40      11413.743368   1
4176654 12      2015-06-18      10:48:50      11413.746424   1
4176655 12      2015-06-18      10:49:00      11413.74948    1
4176656 12      2015-06-18      10:49:10      11413.752536   1
hadoopuser@hd-master:~$
```

Data Pipelining: Sqoop Export

To export the data inside of HDFS to MySQL, we used Sqoop. First, we created a database and table in SQL that will support our data. We called the database task5 and we created a table called "energydata".

```
mysql> use task5;
Database changed
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| task5      |
+-----+
1 row in set (0.00 sec)

mysql> |
```

Exporting data to MySQL from HDFS using sqoop

We then utilized Sqoop's export functionality to export data from HDFS to MySQL. This is the command we ran.

```
sqoop export --connect jdbc:mysql://localhost/task5 --username root --password
Sher1dan@ --table energydata --columns
"LOGID","HOUSEID","CONDATE","CONHOUR","ENERGYREADING","FLAG" --m 1 --export-dir
/group_project/YearMonth/2016/12 --input-fields-terminated-by '\001'
```

This command connects to our mysql database, it populates the fields inside the energydata table by exporting data out of HDFS. The directory we specified was YearMonth/2016/12. This brings in all the data for a singular month. We set the input-fields-terminated-by parameter to '\001' which is the octal representation of a horizontal tab space. Running this command, we get the following output showing our mappers and reducers running to export the data into mysql.

```

hadoopuser@hadoop-master:~$ sqoop export --connect jdbc:mysql://localhost/task5 --username root --password Sheridan@ --table energydata --columns "LOGID","HOUSEID","CONDAT","CONHOUR","ENERGYREADING","FLAG" --m 1 --export-dir /group_project/yearMonthDay/2016/06/18 --input-fields-terminated-by '\001'
2021-12-10 20:48:33,120 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
2021-12-10 20:48:33,161 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
2021-12-10 20:48:33,523 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
2021-12-10 20:48:33,537 INFO tool.CodeGenTool: Beginning code generation
Loading class 'com.mysql.jdbc.Driver'. This is deprecated. The new driver class is 'com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and manual loading of the driver class is generally unnecessary.
2021-12-10 20:48:34,125 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM 'energydata' AS t LIMIT 1
2021-12-10 20:48:34,167 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM 'energydata' AS t LIMIT 1
2021-12-10 20:48:34,176 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /opt/hadoop
Note: /tmp/hadoop-hadoopuser/compile/CC7b868e7b1cfc9c39169f5ab7949e/energydata.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2021-12-10 20:48:36,567 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hadoopuser/compile/CC7b868e7b1cfc9c39169f5ab7949e/energydata.jar
2021-12-10 20:48:36,581 INFO mapreduce.ExportJobBase: Beginning export of energydata
2021-12-10 20:48:36,581 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2021-12-10 20:48:36,593 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
2021-12-10 20:48:42,225 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is deprecated. Instead, use mapreduce.reduce.speculative
2021-12-10 20:48:42,225 INFO Configuration.deprecation: mapred.map.tasks.speculative.execution is deprecated. Instead, use mapreduce.map.speculative
2021-12-10 20:48:42,229 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
2021-12-10 20:48:42,797 INFO client.DefaultHadoopHwallocatorProxyProvider: Connecting to ResourceManager at /8.9.6.9:8032
2021-12-10 20:48:44,137 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoopuser/.staging/job_163223793470_0025
2021-12-10 20:48:47,141 INFO InputFileInputFormat: Total input files to process : 865
2021-12-10 20:48:48,808 INFO InputFileInputFormat: Total input files to process : 865
2021-12-10 20:48:52,166 INFO mapreduce.JobSubmitter: number of splits=1
2021-12-10 20:48:52,171 INFO Configuration.deprecation: mapred.map.tasks.speculative.execution is deprecated. Instead, use mapreduce.map.speculative
2021-12-10 20:48:53,114 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_163223793470_0025
2021-12-10 20:48:53,114 INFO mapreduce.JobSubmitter: Executing with tokens: {}
2021-12-10 20:48:54,638 INFO conf.Configuration: resource-types.xml not found
2021-12-10 20:48:54,638 INFO resource.ResourceUtil: Unable to find 'resource-types.xml'.
2021-12-10 20:48:55,594 INFO tool.HazelClientImpl: Submitted application application_163223793470_0025
2021-12-10 20:48:55,602 INFO mapreduce.Job: The url to track the job: http://hd-master:8080/proxy/application_163223793470_0025/
2021-12-10 20:48:55,602 INFO mapreduce.Job: Running job: job_163223793470_0025
2021-12-10 20:49:23,920 INFO mapreduce.Job: Job job_163223793470_0025 running in uber mode : false
2021-12-10 20:49:23,927 INFO mapreduce.Job: map 0% reduce 0%
2021-12-10 20:49:49,963 INFO mapreduce.Job: map 10% reduce 0%
2021-12-10 20:49:56,608 INFO mapreduce.Job: map 40% reduce 0%
2021-12-10 20:50:02,184 INFO mapreduce.Job: map 60% reduce 0%
2021-12-10 20:50:08,563 INFO mapreduce.Job: map 90% reduce 0%
2021-12-10 20:50:09,713 INFO mapreduce.Job: map 100% reduce 0%
2021-12-10 20:50:10,726 INFO mapreduce.Job: Job job_163223793470_0025 completed successfully
2021-12-10 20:50:10,792 INFO mapreduce.Job: Counter: 3
File System Counters
  FILE: Number of bytes read=0
  FILE: Number of bytes written=272151
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
HDFS: Number of bytes read=485797
HDFS: Number of bytes written=0
HDFS: Number of read operations=2596
HDFS: Number of large read operations=0
HDFS: Number of write operations=0
HDFS: Number of bytes read erasure-coded=0
Job Counters
  Launched map tasks=1
  Rack-local map tasks=1
  Total time spent by all maps in occupied slots (ms)=42241
  Total time spent by all reduces in occupied slots (ms)=0
  Total time spent by all map tasks (ms)=42241
  Total vcores-millisecs taken by all map tasks=42241
  Total megabyte-millisecs taken by all map tasks=1254794

```

49,759 fields have been exported to our MySQL table

```

mysql> select count(*) from energydata;
+-----+
| count(*) |
+-----+
| 49759 |
+-----+
1 row in set (0.06 sec)

```

Data Analysis: MySQL Analysis

Average daily consumption

```
SELECT
    HOUSEID, avg(DailyConsumption) as Average_Daily_Consumption
FROM
    (
        SELECT
            HOUSEID,
            CONDATE,
            (MAX(ENERGYREADING) - MIN(ENERGYREADING)) as DailyConsumption
        FROM energydata
        GROUP BY HOUSEID, CONDATE
    ) as t
GROUP BY HOUSEID;
```

```
mysql> OUSESELECT HOUSEID, avg(DailyConsumption) as Average_Daily_Consumption
-> from (SELECT
-> HOUSEID, CONDATE,(MAX(ENERGYREADING) - MIN(ENERGYREADING)) as DailyConsumption
-> FROM energydata
-> GROUP BY HOUSEID, CONDATE) as t
-> group by HOUSEID;
```

HOUSEID	Average_Daily_Consumption
17	4.2638100000
15	15.5463800000
18	30.7662195000
6	8.7999625000

```
4 rows in set (0.13 sec)
```

Leveraging Sqoop in conjunction with MySQL significantly facilitates data analysis for drawing business insights and providing users with detailed information on their consumption data. This integration streamlines the process of transferring and querying data, enabling efficient and effective analysis to meet business requirements.

Conclusion and Summary

Big Data Technologies fundamentally revolve around usability and scalability. Our proposed solution for ElectroNova details how the company can leverage these technologies to derive actionable insights and drive business growth. The solution encompasses the physical hardware requirements, including calculations for necessary storage space and processing power to handle the data analysis efficiently.

We have integrated several technologies from the Hadoop ecosystem to enable versatile data analysis. This includes:

- **Hadoop Distributed File System (HDFS):** Utilized to securely store and manage the company's extensive database.
- **MapReduce:** Implemented to facilitate rapid and complex analysis across the entire dataset.
- **Apache Spark:** Provides an alternative for running sophisticated data analyses with enhanced performance.
- **Apache Hive:** Allows for HiveQL queries and supports data export for MySQL queries, offering flexibility in data manipulation.
- **Apache Flume:** Designed to handle real-time data from smart meters, ensuring timely and accurate data processing.

Our solution addresses ElectroNova's need for quick, intelligent, and efficient data processing. It is tailored to meet the company's current requirements and is scalable to accommodate future growth.

References

The sample data used in this project is modified from the HUE dataset. The modifications made include changing the hourly aggregate consumption log to sampling at 10-second intervals. The hourly consumption was split equally across 10-second intervals within the hour. The modified data will only be used for educational purposes and is not intended to be used for any real life research work.

Dataset reference:

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).

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:F2ursln9woKDFzaliyA5EA== [fileUNF]