**Project Topic: Boston Airbnb Dataset analysis using Hadoop MapReduce**

**Introduction:**The project aims at performing data analysis on the Boston Airbnb dataset using the Hadoop MapReduce Design patterns such as Summarization, Filtering, Organizational and Join Patterns. Also implemented Pig to perform basic filtering and grouping the data/

The dataset selected includes the following:

* Listings, including full descriptions and average review score
* Reviews, including unique id for each reviewer and detailed comments
* Calendar, including listing id and the price and availability for that day

**Acknowledgement:**This dataset is part of Airbnb Inside, and the original source can be found at <http://insideairbnb.com/get-the-data.html>

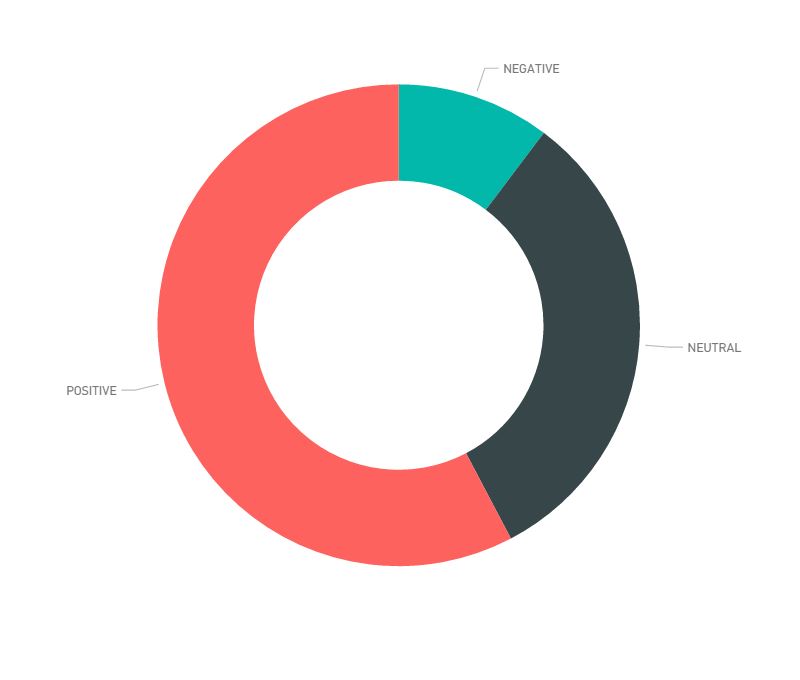
**Analysis Details:**

1. **Sentiment Analysis:**

Performed Sentiment Analysis on the Reviews posted by the users on the listing available in the Airbnb Dataset after their stay. Distributed Cache is used to perform this analysis. Used a dictionary called AFINN, which consists of 2500 words rated from +5 to -5, depending on their meaning. (It can be downloaded from <https://drive.google.com/file/d/0ByJLBTmJojjzZ0d1RVdBTDVjT28/view> )

We declare a HashMap to store the key and value from AFINN and then check for the words in the review against the map., grading each review depending on the range. Finally in the reducer we will sum all the sentiment score from each review for the listing and depending on the total we classify the sentiment as Positive (value > 401), Negative (value < 100) and Neutral( 100 < value < 400).

Graphical Representation of the analysis:



public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

private URI[] files;

private HashMap<String, String> AFINN\_map = new HashMap<String, String>();

public void setup(Context context) throws IOException {

files = DistributedCache.getCacheFiles(context.getConfiguration());

Path path = new Path(files[0]);

FileSystem fs = FileSystem.get(context.getConfiguration());

FSDataInputStream in = fs.open(path);

BufferedReader br = new BufferedReader(new InputStreamReader(in));

String line = "";

while ((line = br.readLine()) != null) {

String splits[] = line.split("\t");

AFINN\_map.put(splits[0], splits[1]);

}

br.close();

in.close();

}

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String listingId;

String review;

String rows[] = value.toString().split(",");

if (rows.length < 2) {

return;

}

listingId = rows[0];

review = rows[1];

String[] splits = review.split(" ");

int sentimentSum = 0;

for (String word : splits) {

if (AFINN\_map.containsKey(word)){

Integer count = new Integer(AFINN\_map.get(word));

sentimentSum += count;

}

}

context.write(new Text(listingId), new IntWritable(sentimentSum));

}

public boolean isAlpha(String name) {

return name.matches("[a-zA-Z]+");

}

}

public static class Reduce extends Reducer<Text, IntWritable, Text, Text> {

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int totalsentiment = 0;

Text result = new Text("NEUTRAL");

for (IntWritable val : values) {

totalsentiment += val.get();

}

if (totalsentiment >= 401) {

result.set(totalsentiment + "\tPOSITIVE");

} else if (totalsentiment < 401 && totalsentiment >= 100 ) {

result.set(totalsentiment + "\tNEUTRAL");

}else if(totalsentiment < 100) {

result.set(totalsentiment + "\tNEGATIVE");

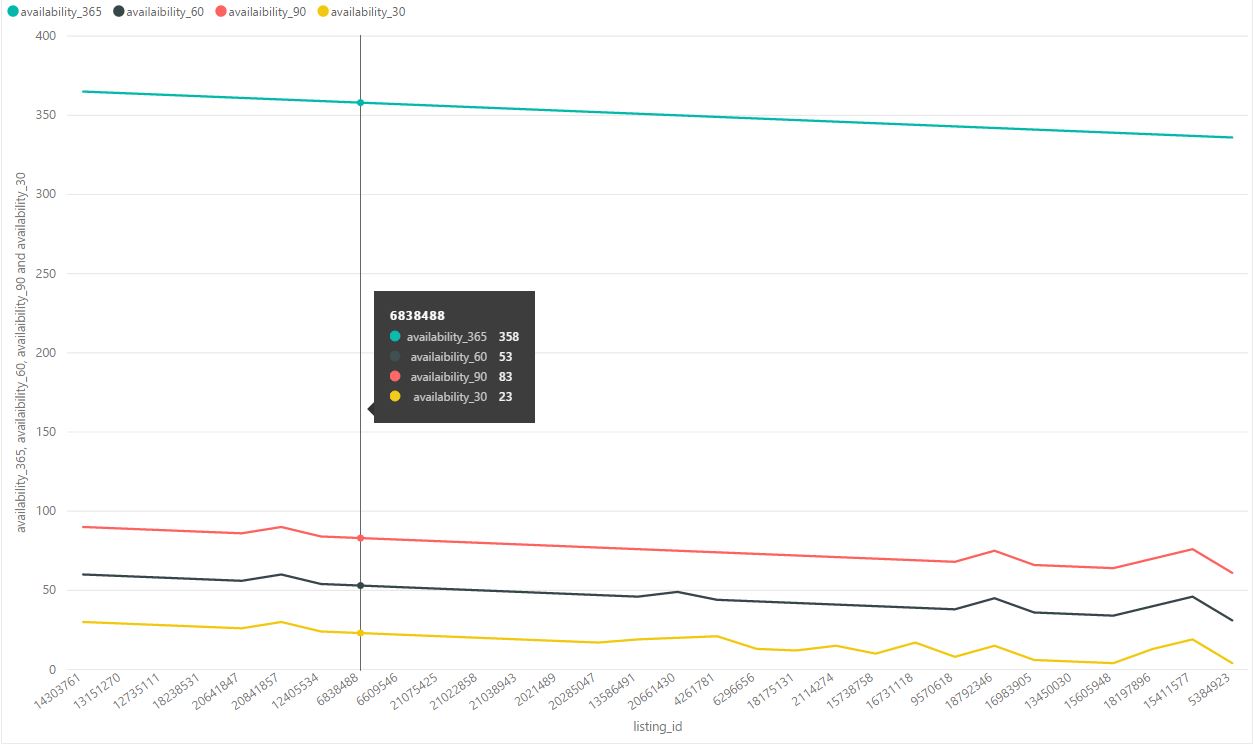
}

context.write(key, result);

}

}

1. **Filtering - TopK Pattern:**Deduced the top 30 listings provided in the dataset that are available for the most number of days in a year. Using this analysis, we can find out the listings that are not housed by any one.

****

public class TopThirtyMapper extends Mapper<Object, Text, NullWritable, Text> {

private TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] row = value.toString().split(",");

String listingId = row[0];

if(listingId.equals("id")) {

return;

}

int availability365 = Integer.parseInt(row[4]);

repToRecordMap.put(availability365, new Text(value));

if (repToRecordMap.size() > 30) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

@Override

protected void cleanup(Context context) throws IOException, InterruptedException {

for (Text t : repToRecordMap.values()) {

context.write(NullWritable.get(), t);

}

}

}

public class TopThirtyReducer extends Reducer<NullWritable, Text, NullWritable, Text> {

private TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

@Override

public void reduce(NullWritable key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

for (Text value : values) {

String[] row = value.toString().split(",");

int availability365 = Integer.parseInt(row[4]);

repToRecordMap.put(availability365, new Text(value));

if (repToRecordMap.size() > 30) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

for (Text t : repToRecordMap.descendingMap().values()) {

context.write(NullWritable.get(), t);

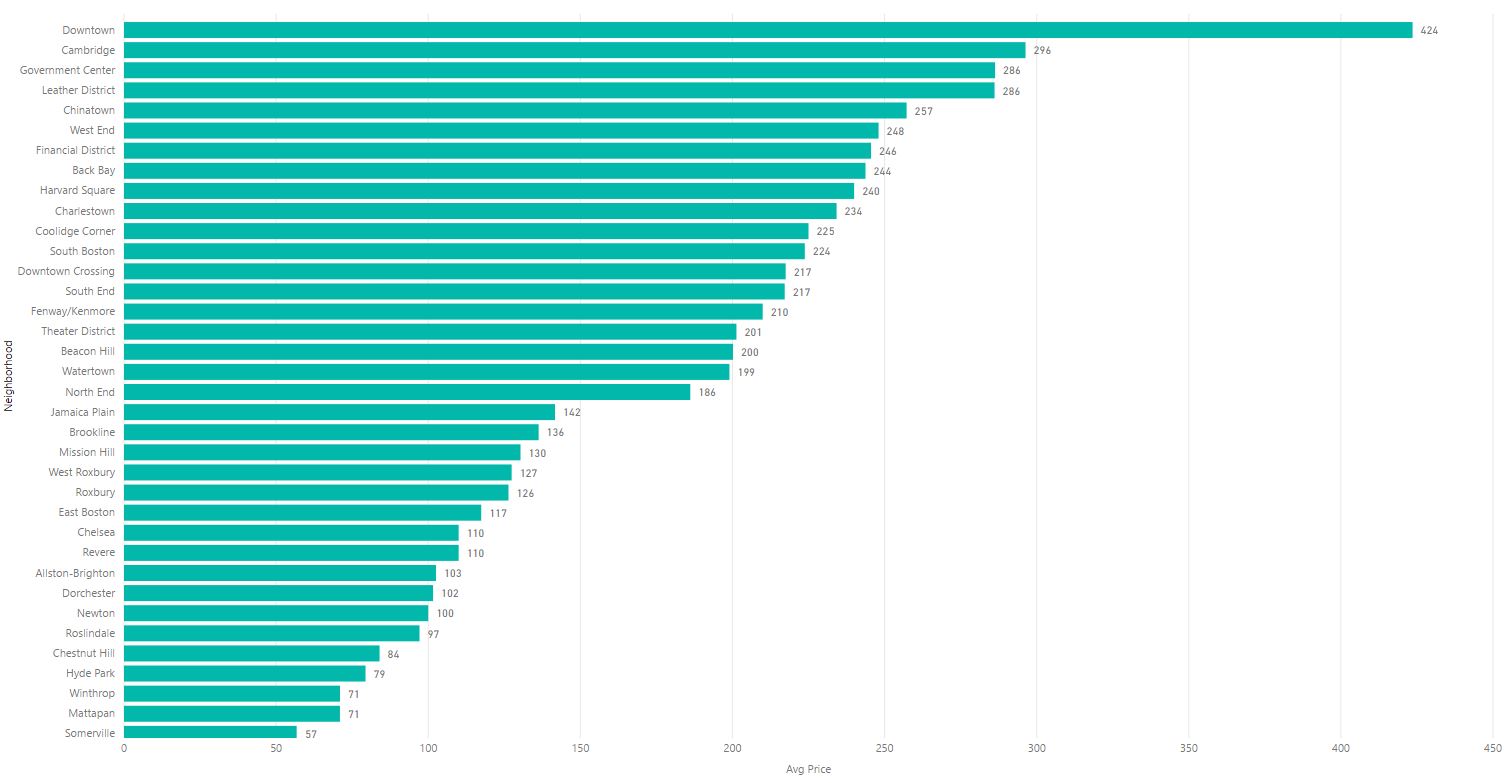
}

}

}

1. **Summarization Pattern – Average Price per Neighborhood**

Calculated the average price of the listings grouped by the neighborhood. It was observed that Downtown and Cambridge have the most expensive listings.

****

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

FloatWritable price = new FloatWritable();

String[] fields = value.toString().split(",");

Text neighborhood = new Text(fields[1]);

if(!neighborhood.toString().contains("neighbourhood")) {

price.set(Float.parseFloat(fields[2]));

context.write(neighborhood, price);

}

}

public void reduce(Text key, Iterable<FloatWritable> values, Context context)

throws IOException, InterruptedException {

float sum = 0;

int count = 0;

for (FloatWritable val : values) {

sum += val.get();

count++;

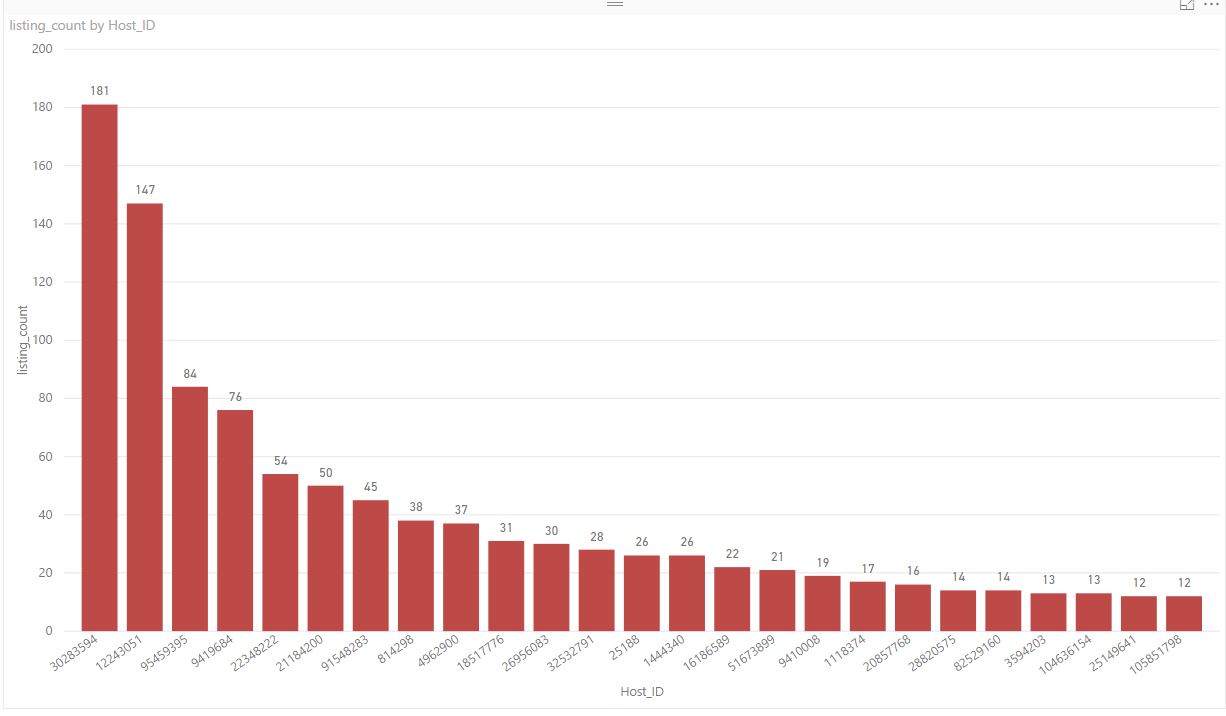
}

average.set(sum / (float) count);

context.write(key, average);

}

1. **Secondary Sorting using Chaining – Calculating Top 25 Hosts with Maximum number of listings posted on AirBnb**

First Mapreduce job calculates the total number of listings posted by each user and then the second mapreduce job sorts the data using a Custom WritableComparable object.****

Mapper1:  
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

Text hostId = new Text();

IntWritable listingCount = new IntWritable(1);

String fields[] = value.toString().split(",");

if (!fields[1].equals("host\_id")) {

hostId.set(fields[1]);

context.write(hostId, listingCount);

}

}

Reducer 1:  
public void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

total.set(sum);

context.write(key, total);

}

Mapper 2:  
protected void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException {

Text hostId = new Text();

FloatWritable listingCount = new FloatWritable();

if (values.toString().length() > 0) {

try {

String fields[] = values.toString().split("\t");

hostId.set(fields[0]);

listingCount.set(Float.parseFloat(fields[1]));

TopHostCustomWritable data = new TopHostCustomWritable(hostId.toString(), listingCount.get());

context.write(data, new IntWritable(1));

} catch (Exception ex) {

ex.printStackTrace();

}

}

}  
Reducer 2:  
protected void reduce(TopHostCustomWritable key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

for (IntWritable value : values) {

if (count < 26) {

context.write(key, NullWritable.get());

count++;

} else {

break;

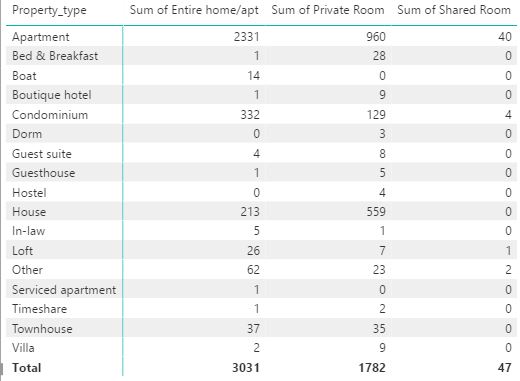
}

}

}

1. **Partitioning Pattern and Numerical Summarization Pattern – Calculating the Number of Room\_type per property type**

There are all together 17 different property types in the provided dataset and each property type can have the room type belonging to any of the three room types:  
Entire home/apt, Private Room, Shared Room

****

Mapper 1:  
public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

String[] row = value.toString().split(",");

String propertyType = row[1];

if(!propertyType.equals("property\_type")) {

if (propertyType.trim().equals("Apartment")) {

outkey.set(1);

}

else if (propertyType.trim().equals("Bed & Breakfast")) {

outkey.set(2);

}

else if (propertyType.trim().equals("Boat")) {

outkey.set(3);

}

else if (propertyType.trim().equals("Boutique hotel")) {

outkey.set(4);

}

else if (propertyType.trim().equals("Condominium")) {

outkey.set(5);

}

else if (propertyType.trim().equals("Dorm")) {

outkey.set(6);

}

else if (propertyType.trim().equals("Guest suite")) {

outkey.set(7);

}

else if (propertyType.trim().equals("Guesthouse")) {

outkey.set(8);

}

else if (propertyType.trim().equals("Hostel")) {

outkey.set(9);

}

else if (propertyType.trim().equals("House")) {

outkey.set(10);

}

else if (propertyType.trim().equals("In-law")) {

outkey.set(11);

}

else if (propertyType.trim().equals("Loft")) {

outkey.set(12);

}

else if (propertyType.trim().equals("Other")) {

outkey.set(13);

}

else if (propertyType.trim().equals("Serviced apartment")) {

outkey.set(14);

}

else if (propertyType.trim().equals("Timeshare")) {

outkey.set(15);

}

else if (propertyType.trim().equals("Townhouse")) {

outkey.set(16);

}

else if (propertyType.trim().equals("Villa")) {

outkey.set(17);

}

context.write(outkey, value);

}

Reducer 1:  
protected void reduce(IntWritable key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

for (Text value : values) {

context.write(value, NullWritable.get());

}

}

Pratitioner:

public int getPartition(IntWritable key, Text value, int numPartitions) {

return key.get() - propertytype;

}

Mapper 2:

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String[] fields = value.toString().split(",");

Text roomType = new Text(fields[1] + " : " + fields[2]);

context.write(roomType, one);

}

Reducer 2:

public void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

total.set(sum);

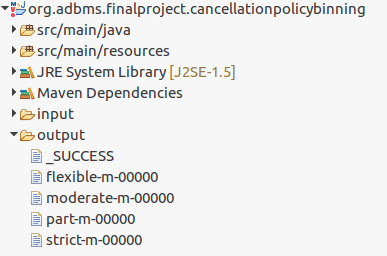
context.write(key, total);

}

1. **Binning Pattern – Bins based on the Cancellation Policy**

Each of the listings in the Airbnb dataset has either of the three types of cancellation policy:

Flexible, Moderate and Strict. This is a map-only job.



public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] row = value.toString().split(",");

String cancellationPolicy = row[1];

if(cancellationPolicy.equals("cancellation\_policy")) {

return;

}

if (cancellationPolicy.equalsIgnoreCase("moderate")) {

mos.write("bins", value, NullWritable.get(), "moderate");

}

if (cancellationPolicy.equalsIgnoreCase("flexible")) {

mos.write("bins", value, NullWritable.get(), "flexible");

}

if (cancellationPolicy.equalsIgnoreCase("strict")) {

mos.write("bins", value, NullWritable.get(), "strict");

}

}

protected void cleanup(Context context) throws IOException, InterruptedException {

mos.close();

}

1. **Data Organization Pattern – Structured to Hierarchical Pattern**

Managed to join the two files ‘listing.csv’ and ‘reviews.csv’ and create a xml structure with listing being the root element and the reviews for that listing as it’s child element.

**O/p snippet:**  
<listing listingid="10033715"><review>This is an amazing apartment near Fenway Park. It looks just as the pictures show. Host was communicative and responsive. 24 hour door man secure and beautifully decorated. According to the doorman this is located in a corporate apartment complex with 20 Corporate apartments and the rest full time tenants. Parking is attached to the building but expensive (just like all parking in Boston). I would highly recommend!</review><review>Everything is good.</review></listing>

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] input=value.toString().split(",");

String listingId=input[0];

if(listingId.equals("id"))

return;

outKey.set(listingId);

outValue.set("L"+listingId);

context.write(outKey,outValue);

}

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] input = value.toString().split(",");

if(input.length != 6) {

return;

}

String listingId = input[0];

String review = input[5];

outKey.set(listingId);

outValue.set("R" + review);

context.write(outKey, outValue);

}

public void reduce(Text key, Iterable<Text> value, Context context) throws IOException, InterruptedException {

listing = null;

reviews.clear();

for (Text text : value) {

if (text.charAt(0) == 'L') {

listing = text.toString().substring(1, text.toString().length()).trim();

} else {

reviews.add(text.toString().substring(1, text.toString().length()).trim());

}

}

if (listing != null) {

try {

String moviesWithTags = nestedElements(listing, reviews);

context.write(new Text(moviesWithTags), NullWritable.get());

} catch (ParserConfigurationException e) {

e.printStackTrace();

} catch (SAXException e) {

e.printStackTrace();

} catch (TransformerException e) {

e.printStackTrace();

}

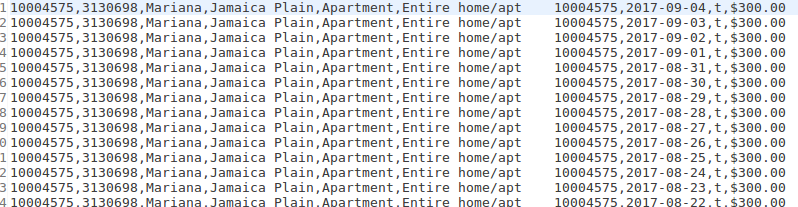
}

}

1. **Join Pattern – Inner Join on two Datasets**

Performed Inner join on two Datasets(listings.csv and calendar.csv) thus enhancing the data by providing the availability of the listings in the year 2018 from Jan to Dec along with the price, if it is available.

O/p Snippet:



public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] values = value.toString().split(",");

String listingId = values[0];

if (listingId.equals("id")) {

return;

}

outkey.set(listingId);

outvalue.set("L|" + value);

context.write(outkey, outvalue);

}

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] rows = value.toString().split(",");

String listingId = rows[0];

if (listingId.equals("listing\_id")) {

return;

}

outkey.set(listingId);

outvalue.set("C|" + value);

context.write(outkey, outvalue);

}

public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

listA.clear();

listB.clear();

while (values.iterator().hasNext()) {

tmp = values.iterator().next();

if ((Character.toString((char) tmp.charAt(0)).equals("L"))

&& (Character.toString((char) tmp.charAt(1)).equals("|"))) {

listA.add(new Text(tmp.toString().substring(2)));

}

if ((Character.toString((char) tmp.charAt(0)).equals("C"))

&& (Character.toString((char) tmp.charAt(1)).equals("|"))) {

listB.add(new Text(tmp.toString().substring(2)));

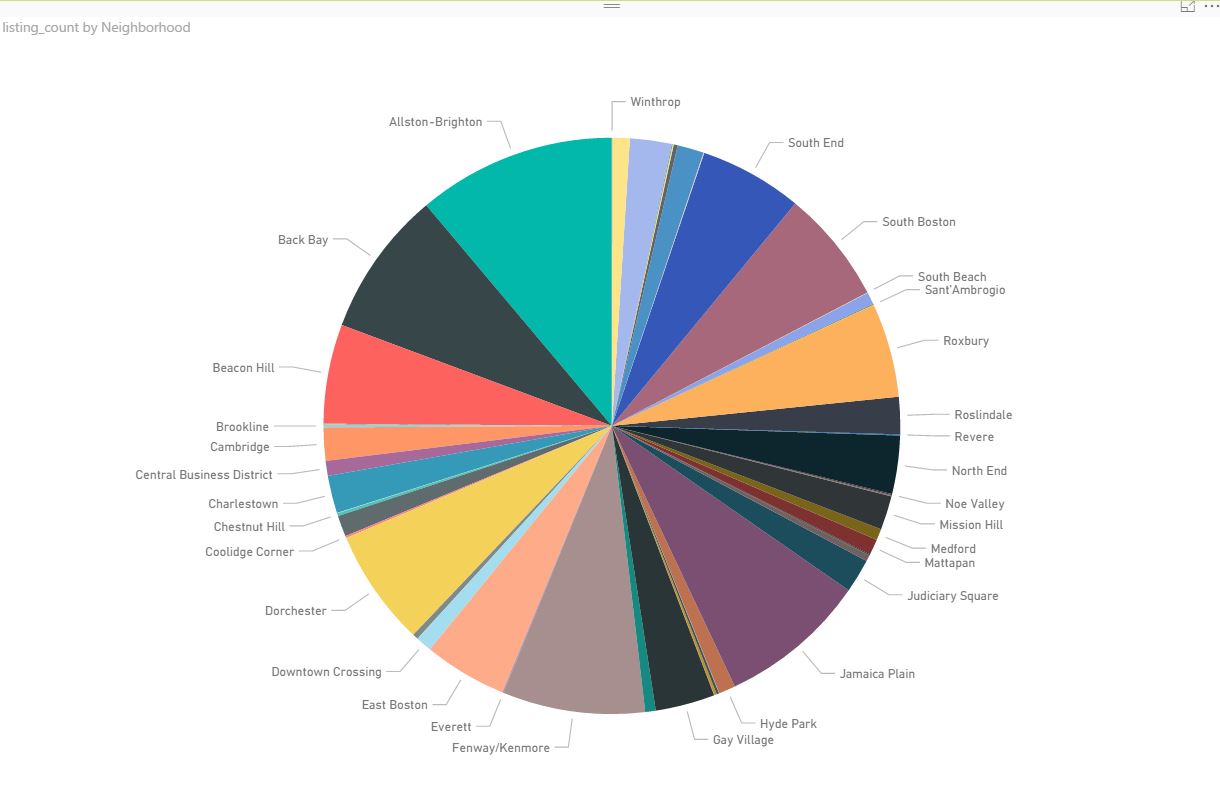
}

}

executeJoinLogic(context);

}

1. **Analysis Using Pig – Calculate the number of listings per neighborhood**

****

query\_data = LOAD '/home/dhanashree/info7275/input/listings\_analysis.csv' USING PigStorage(',');

filtered\_data = FILTER query\_data BY ($4 IS NOT NULL);

grouped\_data = GROUP filtered\_data BY $4;

summed = FOREACH grouped\_data GENERATE group, COUNT(filtered\_data) AS views;

sorted = ORDER summed BY views desc;

STORE sorted INTO '/home/dhanashree/info7275/output';

**Appendix:**

1. **Sentiment Analysis:**

package org.adbms.finalproject.affinsentimentanalysis;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.URI;

import java.util.HashMap;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.filecache.DistributedCache;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.Mapper.Context;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class Driver extends Configured implements Tool {

public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

private URI[] files;

private HashMap<String, String> AFINN\_map = new HashMap<String, String>();

public void setup(Context context) throws IOException {

files = DistributedCache.getCacheFiles(context.getConfiguration());

Path path = new Path(files[0]);

FileSystem fs = FileSystem.get(context.getConfiguration());

FSDataInputStream in = fs.open(path);

BufferedReader br = new BufferedReader(new InputStreamReader(in));

String line = "";

while ((line = br.readLine()) != null) {

String splits[] = line.split("\t");

AFINN\_map.put(splits[0], splits[1]);

}

br.close();

in.close();

}

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String listingId;

String review;

String rows[] = value.toString().split(",");

if (rows.length < 2) {

return;

}

listingId = rows[0];

review = rows[1];

String[] splits = review.split(" ");

int sentimentSum = 0;

for (String word : splits) {

if (AFINN\_map.containsKey(word)){

Integer count = new Integer(AFINN\_map.get(word));

sentimentSum += count;

}

}

context.write(new Text(listingId), new IntWritable(sentimentSum));

}

public boolean isAlpha(String name) {

return name.matches("[a-zA-Z]+");

}

}

public static class Reduce extends Reducer<Text, IntWritable, Text, Text> {

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int totalsentiment = 0;

Text result = new Text("NEUTRAL");

for (IntWritable val : values) {

totalsentiment += val.get();

}

if (totalsentiment >= 401) {

result.set(totalsentiment + "\tPOSITIVE");

} else if (totalsentiment < 401 && totalsentiment >= 100 ) {

result.set(totalsentiment + "\tNEUTRAL");

}else if(totalsentiment < 100) {

result.set(totalsentiment + "\tNEGATIVE");

}

context.write(key, result);

}

}

public static void main(String[] args) throws Exception

{

ToolRunner.run(new Driver(), args);

}

public int run(String[] args) throws Exception {

// TODO Auto-generated method stub

Configuration conf = new Configuration();

if (args.length != 3) {

System.err.println("Usage: Parse <affin> <in> <out>");

System.exit(2);

}

DistributedCache.addCacheFile(new URI(args[0]), conf);

Job job = new Job(conf, "SentimentAnalysis");

job.setJarByClass(Driver.class);

job.setMapperClass(Map.class);

job.setReducerClass(Reduce.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(args[1]));

Path outputDir = new Path(args[2]);

FileOutputFormat.setOutputPath(job, outputDir);

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(outputDir))

hdfs.delete(outputDir, true);

System.exit(job.waitForCompletion(true) ? 0 : 1);

return 0;

}

}

1. **Filtering (TopK) Pattern:**

package org.adbms.finalproject.topthirtylistingswithmaxavailability;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.NullWritable;

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 Driver {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

if (args.length != 2) {

System.err.println("Usage: TopTenDriver <in> <out>");

System.exit(2);

}

Path inputPath = new Path(args[0]);

Path outputDir = new Path(args[1]);

Job job = new Job(conf, "Top Ten Users by Reputation");

job.setJarByClass(TopThirtyMapper.class);

job.setMapperClass(TopThirtyMapper.class);

job.setReducerClass(TopThirtyReducer.class);

job.setNumReduceTasks(1);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, inputPath);

FileOutputFormat.setOutputPath(job, outputDir);

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(outputDir))

hdfs.delete(outputDir, true);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

package org.adbms.finalproject.topthirtylistingswithmaxavailability;

import java.io.IOException;

import java.util.TreeMap;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class TopThirtyMapper extends Mapper<Object, Text, NullWritable, Text> {

// Our output key and value Writables

private TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] row = value.toString().split(",");

String listingId = row[0];

if(listingId.equals("id")) {

return;

}

int availability365 = Integer.parseInt(row[4]);

repToRecordMap.put(availability365, new Text(value));

if (repToRecordMap.size() > 30) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

@Override

protected void cleanup(Context context) throws IOException, InterruptedException {

for (Text t : repToRecordMap.values()) {

context.write(NullWritable.get(), t);

}

}

}

package org.adbms.finalproject.topthirtylistingswithmaxavailability;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import java.util.TreeMap;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class TopThirtyReducer extends Reducer<NullWritable, Text, NullWritable, Text> {

private TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

@Override

public void reduce(NullWritable key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

for (Text value : values) {

String[] row = value.toString().split(",");

int availability365 = Integer.parseInt(row[4]);

repToRecordMap.put(availability365, new Text(value));

if (repToRecordMap.size() > 30) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

for (Text t : repToRecordMap.descendingMap().values()) {

context.write(NullWritable.get(), t);

}

}

}

1. **Summarization Pattern (Average):**

package org.adbms.finalproject.averagepriceperneighborhood;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.log4j.PropertyConfigurator;

public class Driver {

public static void main(String[] args) throws Exception {

if (args.length != 2) {

System.err.println("Usage: Average Price per neighborhood <input path> <output path>");

System.exit(-1);

}

Path inputPath = new Path(args[0]);

Path outputDir = new Path(args[1]);

// Create configuration

Configuration conf = new Configuration(true);

FileSystem hdfs = FileSystem.get(conf);

// Create job

Job job = new Job(conf, "AveragePricePerNeighborhood");

job.setJarByClass(NeighborhoodAveragePriceMapper.class);

// Setup MapReduce

job.setMapperClass(NeighborhoodAveragePriceMapper.class);

job.setReducerClass(NeighborhoodAveragePriceReducer.class);

job.setNumReduceTasks(1);

// Specify key / value

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(FloatWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(FloatWritable.class);

// Input

FileInputFormat.addInputPath(job, inputPath);

job.setInputFormatClass(TextInputFormat.class);

// Output

FileOutputFormat.setOutputPath(job, outputDir);

// Delete output if exists

if (hdfs.exists(outputDir))

hdfs.delete(outputDir, true);

// Execute job

int code = job.waitForCompletion(true) ? 0 : 1;

System.exit(code);

}

}

package org.adbms.finalproject.averagepriceperneighborhood;

import java.io.IOException;

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 org.apache.hadoop.mapreduce.Mapper.Context;

public class NeighborhoodAveragePriceMapper extends Mapper<LongWritable, Text, Text, FloatWritable> {

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

FloatWritable price = new FloatWritable();

String[] fields = value.toString().split(",");

Text neighborhood = new Text(fields[1]);

if(!neighborhood.toString().contains("neighbourhood")) {

price.set(Float.parseFloat(fields[2]));

context.write(neighborhood, price);

}

}

}

package org.adbms.finalproject.averagepriceperneighborhood;

import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.Reducer.Context;

public class NeighborhoodAveragePriceReducer extends Reducer<Text, FloatWritable, Text, FloatWritable> {

private FloatWritable average = new FloatWritable();

public void reduce(Text key, Iterable<FloatWritable> values, Context context)

throws IOException, InterruptedException {

float sum = 0;

int count = 0;

for (FloatWritable val : values) {

sum += val.get();

count++;

}

average.set(sum / (float) count);

context.write(key, average);

}

}

1. **Secondary Sorting using Chaining:**

package tophosts;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.NullWritable;

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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver {

public static void main(String[] args) throws Exception {

if (args.length != 3) {

System.err.println("Usage: TopHostWithMostListings <input path> <intermediate> <output path>");

System.exit(-1);

}

Path inputPath = new Path(args[0]);

Path output\_intermediate = new Path(args[1]);

Path output\_dir = new Path(args[2]);

// Create configuration

Configuration conf = new Configuration(true);

// Create job

Job job = new Job(conf, "TopHostWithMostListings");

job.setJarByClass(TophostWithMostListingsMapper1.class);

// Setup MapReduce

job.setMapperClass(TophostWithMostListingsMapper1.class);

job.setReducerClass(TophostWithMostListingsReducer1.class);

// Specify key / value

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

// Input

FileInputFormat.addInputPath(job, inputPath);

job.setInputFormatClass(TextInputFormat.class);

// Output

FileOutputFormat.setOutputPath(job, output\_intermediate);

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(output\_intermediate))

hdfs.delete(output\_intermediate, true);

if (hdfs.exists(output\_dir))

hdfs.delete(output\_dir, true);

boolean complete = job.waitForCompletion(true);

Configuration conf2 = new Configuration();

Job job2 = Job.getInstance(conf2, "Chaining");

if (complete) {

// Setup MapReduce

job2.setJarByClass(TophostWithMostListingsMapper2.class);

job2.setMapperClass(TophostWithMostListingsMapper2.class);

job2.setReducerClass(TophostWithMostListingsReducer2.class);

// Specify key / value

job2.setMapOutputKeyClass(TopHostCustomWritable.class);

job2.setMapOutputValueClass(IntWritable.class);

job2.setOutputKeyClass(TopHostCustomWritable.class);

job2.setOutputValueClass(NullWritable.class);

// Input

FileInputFormat.addInputPath(job2, output\_intermediate);

job2.setInputFormatClass(TextInputFormat.class);

FileOutputFormat.setOutputPath(job2, output\_dir);

//System.exit(job2.waitForCompletion(true) ? 0 : 1);

int code = job2.waitForCompletion(true) ? 0 : 1;

System.exit(code);

}

}

}

package tophosts;

import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class TophostWithMostListingsMapper1 extends Mapper<LongWritable, Text, Text, IntWritable> {

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

Text hostId = new Text();

IntWritable listingCount = new IntWritable(1);

String fields[] = value.toString().split(",");

if (!fields[1].equals("host\_id")) {

hostId.set(fields[1]);

context.write(hostId, listingCount);

}

}

}

package tophosts;

import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class TophostWithMostListingsMapper1 extends Mapper<LongWritable, Text, Text, IntWritable> {

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

Text hostId = new Text();

IntWritable listingCount = new IntWritable(1);

String fields[] = value.toString().split(",");

if (!fields[1].equals("host\_id")) {

hostId.set(fields[1]);

context.write(hostId, listingCount);

}

}

}

package tophosts;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class TophostWithMostListingsReducer1 extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable total = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

total.set(sum);

context.write(key, total);

}

}

package tophosts;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.Writable;

import org.apache.hadoop.io.WritableComparable;

public class TopHostCustomWritable implements Writable, WritableComparable<TopHostCustomWritable>{

private String hostId;

private Float listingCount;

public TopHostCustomWritable() {

}

public TopHostCustomWritable(String m, Float l) {

this.hostId = m;

this.listingCount = l;

}

public String getHostId() {

return hostId;

}

public void setHostId(String hostId) {

this.hostId = hostId;

}

public Float getListingCount() {

return listingCount;

}

public void setListingCount(Float listingCount) {

this.listingCount = listingCount;

}

public void write(DataOutput d) throws IOException {

d.writeUTF(hostId);

d.writeFloat(listingCount);

}

public void readFields(DataInput di) throws IOException {

hostId = di.readUTF();

listingCount = di.readFloat();

}

public int compareTo(TopHostCustomWritable o) {

return -1 \* (listingCount.compareTo(o.listingCount));

}

@Override

public String toString() {

return hostId + "\t" + listingCount;

}

}

package tophosts;

import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class TophostWithMostListingsMapper2 extends Mapper<LongWritable, Text, TopHostCustomWritable, IntWritable>{

protected void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException {

Text hostId = new Text();

FloatWritable listingCount = new FloatWritable();

if (values.toString().length() > 0) {

try {

String fields[] = values.toString().split("\t");

hostId.set(fields[0]);

listingCount.set(Float.parseFloat(fields[1]));

TopHostCustomWritable data = new TopHostCustomWritable(hostId.toString(), listingCount.get());

context.write(data, new IntWritable(1));

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

}

package tophosts;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.mapreduce.Reducer;

public class TophostWithMostListingsReducer2

extends Reducer<TopHostCustomWritable, IntWritable, TopHostCustomWritable, NullWritable> {

public static int count = 1;

@Override

protected void reduce(TopHostCustomWritable key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

for (IntWritable value : values) {

if (count < 26) {

context.write(key, NullWritable.get());

count++;

} else {

break;

}

}

}

}

1. **Partitioning Pattern and Numerical Summarization Pattern:**

package org.adbms.finalproject.listingperpropertytype;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.NullWritable;

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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

if (args.length != 3) {

System.err.println("Usage: Number of Room type Group by Property Type <input> <intermediate> <output>");

System.exit(2);

}

Path inputPath = new Path(args[0]);

Path intermediate = new Path(args[1]);

Path outputDir = new Path(args[2]);

Job job = new Job(conf, "NumOfRoomTypeGroupByPropertyType");

job.setJarByClass(PartitionPatternMapper.class);

job.setMapperClass(PartitionPatternMapper.class);

job.setPartitionerClass(GroupbyPropertyTypePartitioner.class);

GroupbyPropertyTypePartitioner.setPropertyType(job, 1);

job.setReducerClass(PartitionPatternReducer.class);

job.setNumReduceTasks(17);

job.setMapOutputKeyClass(IntWritable.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(NullWritable.class);

// Input

FileInputFormat.addInputPath(job, inputPath);

job.setInputFormatClass(TextInputFormat.class);

// Output

FileOutputFormat.setOutputPath(job, intermediate);

// Delete output if exists

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(intermediate))

hdfs.delete(intermediate, true);

// Execute job

boolean complete = job.waitForCompletion(true);

Configuration conf2 = new Configuration(true);

Job job2 = new Job(conf, "NumOfRoomTypeGroupByPropertyType");

if (complete) {

job2.setJarByClass(RoomTypeMapper.class);

// Setup MapReduce

job2.setMapperClass(RoomTypeMapper.class);

job2.setReducerClass(RoomTypeReducer.class);

job2.setNumReduceTasks(1);

// Specify key / value

job2.setMapOutputKeyClass(Text.class);

job2.setMapOutputValueClass(IntWritable.class);

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(IntWritable.class);

// Input

FileInputFormat.addInputPath(job2, intermediate);

job2.setInputFormatClass(TextInputFormat.class);

// Output

FileOutputFormat.setOutputPath(job2, outputDir);

// Delete output if exists

if (hdfs.exists(outputDir))

hdfs.delete(outputDir, true);

// Execute job

int code = job2.waitForCompletion(true) ? 0 : 1;

System.exit(code);

}

}

}

package org.adbms.finalproject.listingperpropertytype;

import org.apache.hadoop.conf.Configurable;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Partitioner;

public class GroupbyPropertyTypePartitioner extends Partitioner<IntWritable, Text> implements Configurable {

private static final String PROPERTY\_TYPE = "propertytype";

private Configuration conf = null;

private int propertytype = 0;

public int getPartition(IntWritable key, Text value, int numPartitions) {

return key.get() - propertytype;

}

public Configuration getConf() {

return conf;

}

public void setConf(Configuration conf) {

this.conf = conf;

propertytype = conf.getInt(PROPERTY\_TYPE, 0);

}

public static void setPropertyType(Job job, int type) {

job.getConfiguration().setInt(PROPERTY\_TYPE, type);

}

}

package org.adbms.finalproject.listingperpropertytype;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class PartitionPatternMapper extends Mapper<Object, Text, IntWritable, Text>{

private IntWritable outkey = new IntWritable();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

String[] row = value.toString().split(",");

String propertyType = row[1];

if(!propertyType.equals("property\_type")) {

if (propertyType.trim().equals("Apartment")) {

outkey.set(1);

}

else if (propertyType.trim().equals("Bed & Breakfast")) {

outkey.set(2);

}

else if (propertyType.trim().equals("Boat")) {

outkey.set(3);

}

else if (propertyType.trim().equals("Boutique hotel")) {

outkey.set(4);

}

else if (propertyType.trim().equals("Condominium")) {

outkey.set(5);

}

else if (propertyType.trim().equals("Dorm")) {

outkey.set(6);

}

else if (propertyType.trim().equals("Guest suite")) {

outkey.set(7);

}

else if (propertyType.trim().equals("Guesthouse")) {

outkey.set(8);

}

else if (propertyType.trim().equals("Hostel")) {

outkey.set(9);

}

else if (propertyType.trim().equals("House")) {

outkey.set(10);

}

else if (propertyType.trim().equals("In-law")) {

outkey.set(11);

}

else if (propertyType.trim().equals("Loft")) {

outkey.set(12);

}

else if (propertyType.trim().equals("Other")) {

outkey.set(13);

}

else if (propertyType.trim().equals("Serviced apartment")) {

outkey.set(14);

}

else if (propertyType.trim().equals("Timeshare")) {

outkey.set(15);

}

else if (propertyType.trim().equals("Townhouse")) {

outkey.set(16);

}

else if (propertyType.trim().equals("Villa")) {

outkey.set(17);

}

context.write(outkey, value);

}

}

}

package org.adbms.finalproject.listingperpropertytype;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class PartitionPatternReducer extends Reducer<IntWritable, Text, Text, NullWritable> {

protected void reduce(IntWritable key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

for (Text value : values) {

context.write(value, NullWritable.get());

}

}

}

package org.adbms.finalproject.listingperpropertytype;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class RoomTypeMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String[] fields = value.toString().split(",");

Text roomType = new Text(fields[1] + " : " + fields[2]);

context.write(roomType, one);

}

}

package org.adbms.finalproject.listingperpropertytype;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class RoomTypeReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable total = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

total.set(sum);

context.write(key, total);

}

}

1. **Binning Pattern:**

package org.adbms.finalproject.cancellationpolicybinning;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Driver {

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

Configuration conf = new Configuration();

if (args.length != 2) {

System.err.println("Usage: Binning <input> <output>");

System.exit(2);

}

Path inputPath = new Path(args[0]);

Path outputDir = new Path(args[1]);

Job job = new Job(conf, "Binning");

job.setJarByClass(BinningMapper.class);

job.setMapperClass(BinningMapper.class);

job.setNumReduceTasks(0);

TextInputFormat.setInputPaths(job, inputPath);

FileOutputFormat.setOutputPath(job, outputDir);

MultipleOutputs.addNamedOutput(job, "bins", TextOutputFormat.class, Text.class, NullWritable.class);

MultipleOutputs.setCountersEnabled(job, true);

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(outputDir))

hdfs.delete(outputDir, true);

// Execute job

int code = job.waitForCompletion(true) ? 0 : 1;

System.exit(code);

}

}

package org.adbms.finalproject.cancellationpolicybinning;

import java.io.IOException;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;

public class BinningMapper extends Mapper<Object, Text, Text, NullWritable> {

private MultipleOutputs<Text, NullWritable> mos = null;

protected void setup(Context context) {

mos = new MultipleOutputs(context);

}

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] row = value.toString().split(",");

String cancellationPolicy = row[1];

if(cancellationPolicy.equals("cancellation\_policy")) {

return;

}

if (cancellationPolicy.equalsIgnoreCase("moderate")) {

mos.write("bins", value, NullWritable.get(), "moderate");

}

if (cancellationPolicy.equalsIgnoreCase("flexible")) {

mos.write("bins", value, NullWritable.get(), "flexible");

}

if (cancellationPolicy.equalsIgnoreCase("strict")) {

mos.write("bins", value, NullWritable.get(), "strict");

}

}

protected void cleanup(Context context) throws IOException, InterruptedException {

mos.close();

}

}

1. **Data Organization Pattern:**

package org.adbms.finalproject.listingandreviewshierarchy;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;

public class Driver {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

if (otherArgs.length != 3) {

System.err.println("Usage: ListingReviewHierarchy <movie> <tag> <outdir>");

System.exit(1);

}

Path outputDir = new Path(args[2]);

Job job = new Job(conf, "ListingReviewHierarchy");

job.setJarByClass(Driver.class);

MultipleInputs.addInputPath(job, new Path(otherArgs[0]), TextInputFormat.class, ListingMapper.class);

MultipleInputs.addInputPath(job, new Path(otherArgs[1]), TextInputFormat.class, ReviewMapper.class);

job.setReducerClass(ListingReviewReducer.class);

job.setOutputFormatClass(TextOutputFormat.class);

TextOutputFormat.setOutputPath(job, outputDir);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

// Delete output if exists

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(outputDir))

hdfs.delete(outputDir, true);

System.exit(job.waitForCompletion(true) ? 0 : 2);

}

}

package org.adbms.finalproject.listingandreviewshierarchy;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class ListingMapper extends Mapper<Object,Text,Text,Text> {

private Text outKey=new Text();

private Text outValue=new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] input=value.toString().split(",");

String listingId=input[0];

if(listingId.equals("id"))

return;

outKey.set(listingId);

outValue.set("L"+listingId);

context.write(outKey,outValue);

}

}

package org.adbms.finalproject.listingandreviewshierarchy;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class ReviewMapper extends Mapper<Object, Text, Text, Text> {

private Text outKey = new Text();

private Text outValue = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] input = value.toString().split(",");

if(input.length != 6) {

return;

}

String listingId = input[0];

String review = input[5];

outKey.set(listingId);

outValue.set("R" + review);

context.write(outKey, outValue);

}

}

package org.adbms.finalproject.listingandreviewshierarchy;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.Reducer.Context;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.xml.sax.SAXException;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.\*;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import java.io.IOException;

import java.io.StringWriter;

import java.util.ArrayList;

import java.util.List;

public class ListingReviewReducer extends Reducer<Text,Text,Text,NullWritable>

{

private ArrayList<String> reviews=new ArrayList<String>();

private DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

private String listing = null;

@Override

public void reduce(Text key, Iterable<Text> value, Context context) throws IOException, InterruptedException {

listing = null;

reviews.clear();

for (Text text : value) {

if (text.charAt(0) == 'L') {

listing = text.toString().substring(1, text.toString().length()).trim();

} else {

reviews.add(text.toString().substring(1, text.toString().length()).trim());

}

}

if (listing != null) {

try {

String moviesWithTags = nestedElements(listing, reviews);

context.write(new Text(moviesWithTags), NullWritable.get());

} catch (ParserConfigurationException e) {

e.printStackTrace();

} catch (SAXException e) {

e.printStackTrace();

} catch (TransformerException e) {

e.printStackTrace();

}

}

}

private String nestedElements(String listingid, List<String> reviews)

throws ParserConfigurationException, IOException, SAXException, TransformerException {

DocumentBuilder db = dbf.newDocumentBuilder();

Document doc = db.newDocument();

Element rootElement = doc.createElement("listing");

rootElement.setAttribute("listingid", listingid);

doc.appendChild(rootElement);

for (String tagsXml : reviews) {

Element tag = doc.createElement("review");

tag.appendChild(doc.createTextNode(tagsXml));

rootElement.appendChild(tag);

}

return transforDocumentToString(doc);

}

private String transforDocumentToString(Document doc) throws TransformerException {

TransformerFactory tf = TransformerFactory.newInstance();

Transformer transformer = tf.newTransformer();

transformer.setOutputProperty(OutputKeys.OMIT\_XML\_DECLARATION, "yes");

StringWriter writer = new StringWriter();

transformer.transform(new DOMSource(doc), new StreamResult(writer));

return writer.getBuffer().toString().replaceAll("\n|\r", "");

}

}

1. **Join Pattern:**

package org.adbms.finalproject.joinpattern;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Driver {

public static void main(String args[]) throws Exception {

if (args.length != 3) {

System.err.println(

"Usage: Join <input\_listings> <input\_calendar> <output>");

System.exit(2);

}

Path listingInput = new Path(args[0]);

Path calenderInput = new Path(args[1]);

Path outputDir = new Path(args[2]);

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Join ReduceSide");

job.setJarByClass(Driver.class);

MultipleInputs.addInputPath(job, listingInput, TextInputFormat.class, ListingMapper.class);

MultipleInputs.addInputPath(job, calenderInput, TextInputFormat.class, CalendarMapper.class);

job.getConfiguration().set("join.type", "inner");

job.setReducerClass(ListingCalendarReducer.class);

job.setOutputFormatClass(TextOutputFormat.class);

TextOutputFormat.setOutputPath(job, outputDir);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(outputDir))

hdfs.delete(outputDir, true);

System.exit(job.waitForCompletion(true) ? 0 : 2);

}

}

package org.adbms.finalproject.joinpattern;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class ListingMapper extends Mapper<Object, Text, Text, Text> {

private Text outkey = new Text();

private Text outvalue = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] values = value.toString().split(",");

String listingId = values[0];

if (listingId.equals("id")) {

return;

}

outkey.set(listingId);

outvalue.set("L|" + value);

context.write(outkey, outvalue);

}

}

package org.adbms.finalproject.joinpattern;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class CalendarMapper extends Mapper<Object, Text, Text, Text> {

private Text outkey = new Text();

private Text outvalue = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] rows = value.toString().split(",");

String listingId = rows[0];

if (listingId.equals("listing\_id")) {

return;

}

outkey.set(listingId);

outvalue.set("C|" + value);

context.write(outkey, outvalue);

}

}

package org.adbms.finalproject.joinpattern;

import java.io.IOException;

import java.util.ArrayList;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class ListingCalendarReducer extends Reducer<Text, Text, Text, Text> {

private static final Text EMPTY\_TEXT = new Text("");

private Text tmp = new Text();

private ArrayList<Text> listA = new ArrayList<Text>();

private ArrayList<Text> listB = new ArrayList<Text>();

private String joinType = null;

public void setup(Context context) {

joinType = context.getConfiguration().get("join.type");

}

public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

listA.clear();

listB.clear();

while (values.iterator().hasNext()) {

tmp = values.iterator().next();

if ((Character.toString((char) tmp.charAt(0)).equals("L"))

&& (Character.toString((char) tmp.charAt(1)).equals("|"))) {

listA.add(new Text(tmp.toString().substring(2)));

}

if ((Character.toString((char) tmp.charAt(0)).equals("C"))

&& (Character.toString((char) tmp.charAt(1)).equals("|"))) {

listB.add(new Text(tmp.toString().substring(2)));

}

}

executeJoinLogic(context);

}

private void executeJoinLogic(Context context) throws IOException, InterruptedException {

if (joinType.equalsIgnoreCase("inner")) {

if (!listA.isEmpty() && !listB.isEmpty()) {

for (Text A : listA) {

for (Text B : listB) {

context.write(A, B);

}

}

}

} else if (joinType.equalsIgnoreCase("leftouter")) {

for (Text A : listA) {

if (!listB.isEmpty()) {

for (Text B : listB) {

context.write(A, B);

}

} else {

context.write(A, EMPTY\_TEXT);

}

}

} else if (joinType.equalsIgnoreCase("rightouter")) {

for (Text B : listB) {

if (!listA.isEmpty()) {

for (Text A : listA) {

context.write(A, B);

}

} else {

context.write(EMPTY\_TEXT, B);

}

}

} else if (joinType.equalsIgnoreCase("fullouter")) {

if (!listA.isEmpty()) {

for (Text A : listA) {

if (!listB.isEmpty()) {

for (Text B : listB) {

context.write(A, B);

}

} else {

context.write(A, EMPTY\_TEXT);

}

}

} else {

for (Text B : listB) {

context.write(EMPTY\_TEXT, B);

}

}

} else if (joinType.equalsIgnoreCase("anti")) {

if (listA.isEmpty() ^ listB.isEmpty()) {

for (Text A : listA) {

context.write(A, EMPTY\_TEXT);

}

for (Text B : listB) {

context.write(EMPTY\_TEXT, B);

}

}

}

}

}

**Project also uploaded on Github:**

<https://github.com/DhanashreeC/ADBMSMapReduceAirbnbDataSet>