AdventureWorks DataWareHouse Analysis

**Northeastern University**

**12/12/2016**

Vijayashree Uppili

This document includes details of various MapReduce patterns implemented to do analysis on AdventureWorks data warehouse and get hands on various scenarios like product sales based on region, promotion types etc. Also this document discuss about improvement in few techniques used in Mapreduce pattern.

Contents

[1. Problem Statement 3](#_Toc469341457)

[2. Dataset 3](#_Toc469341458)

[3. Analysis 6](#_Toc469341459)

[A) Top 10 Products by TotalSales (Join, Summarization, Filtering Patterns) 6](#_Toc469341460)

[B) Total Sales of Product by Geography (Join, Summarization, Binning Pattern) 7](#_Toc469341461)

[C) Counting Number of Customer based in Geo (Distinct and Counting with Counter pattern) 8](#_Toc469341462)

[D) Predictive yearly Sales Analysis (Summarization pattern) 9](#_Toc469341463)

[E) Analyzing Products promoted based on Specific reason (Inverted Index) 10](#_Toc469341464)

[F) Performance Improvement 11](#_Toc469341465)

[i) Sending Less Data over network 11](#_Toc469341466)

[ii) Use of Partitioner and Combiner 12](#_Toc469341467)

[iii) Input/Output file Compression 13](#_Toc469341468)

[iv) Multiple Joins using Caching 14](#_Toc469341469)

[v) GenericWritable vs Writable 16](#_Toc469341470)

[G) Visual Representation 16](#_Toc469341471)

[H) HBase 18](#_Toc469341472)

[Appendix 19](#_Toc469341473)

[i) Code for Top 10 Products by TotalSales (Join, Summarization, Filtering Patterns) 19](#_Toc469341474)

[ii) Code for Total Sales of Product by Geography (Join, Summarization, Binning Pattern) 29](#_Toc469341475)

[iii) Code for Counting Number of Customer based in Geo (Distinct and Counting with Counter pattern) 37](#_Toc469341476)

[iv) Code for Predictive yearly Sales Analysis (Summarization pattern) 40](#_Toc469341477)

[v) Code for Analyzing Products promoted based on Specific reason (Inverted Index) 44](#_Toc469341478)

[vi) Code for Performance Improvement 47](#_Toc469341479)

**Table of Figure**

[Figure 1: SQL Query for Top10 6](#_Toc469340352)

[Figure 2: Counting with Counter 7](#_Toc469340353)

[Figure 3: Output of Counters 8](#_Toc469340354)

[Figure 4: Code Snippet for Inverted Index 9](#_Toc469340355)

[Figure 5: Performance Comparison of various analysis 9](#_Toc469340356)

[Figure 6: Original Map output 10](#_Toc469340357)

[Figure 7: Improved Map Output 10](#_Toc469340358)

[Figure 8: Code Snippet Partitioner 11](#_Toc469340359)

[Figure 9: Cod Snippet of Configuring Combiner and Partitioner 11](#_Toc469340360)

[Figure 10: Compression Format supported by Hadoop 12](#_Toc469340361)

[Figure 11: Configuring file Compression 12](#_Toc469340362)

[Figure 12: Cache Property in job 13](#_Toc469340363)

[Figure 13: Reading cached file in setup phase of map 13](#_Toc469340364)

[Figure 14: Generic Writable Class 14](#_Toc469340365)

[Figure 15: Dashboard of Input data analysis 14](#_Toc469340366)

[Figure 16:TotalSales and Profit of United States 15](#_Toc469340367)

[Figure 17: Dashboard for output Analysis 15](#_Toc469340368)

# Problem Statement

Adventureworks is a sample data warehouse dataset which gives students a hands on various business scenario which includes Manufacturing, Sales, Purchasing, Product Management etc of a fictitious company. This dataset will help one to understand the basic scenarios of marketing and help one to analyze various ups and down in marketing which in turn is affected by region, product type, age group etc. Following analysis are covered in this document

1. Analysing TotalSales and TotalProfit for each product using reduce side join and map reduce chaining for calculating total sales and total profit of products
2. Top 10 products by total sales using reduce side joining and top 10 filtering pattern
3. Analysing TotalSales and TotalProfit of Products sold segregated by countryregion using reduce side join for joining 3 files and seggerating data based on country by using binnig pattern
4. Total number of distinct customer in each country by using distinct and counter pattern
5. Forecasting the sales for next year based on StandardDeviation.
6. Creating Inverted index on all product which were promoted for specific promotion reason category.
7. Performance tuning by using compression of files while performing Mapreduce on larger file
8. Performance tuning by using cache for multiple file join.
9. Performance tuning by using partitioner and combiner for summarization pattern

# Dataset

Link: <https://msftdbprodsamples.codeplex.com/releases>

Official Documentation: https://technet.microsoft.com/en-us/library/ms124438(v=sql.100).aspx

Dataset description

**FactInternetSales.csv**

|  |  |
| --- | --- |
| **Field** | Description |
| **ProductKey** | Primary key for **Product** rows. |
| **OrderDateKey** | YYYYMMDD Order date |
| **DueDateKey** | YYYYMMDD Due date |
| **ShipDateKey** | YYYYMMDD Ship date |
| **CustomerKey** | Foreign Key to Customer data |
| **PromotionKey** | Foreign key to promotion data |
| **CurrencyKey** | Foreign key to Currency data |
| **SalesOrderNumber** | Unique sales order identification number. |
| **SalesOrderLineNumber** | Customer purchase order number reference. |
| **RevisionNumber** | Incremental number to track changes to the sales order over time. |
| **OrderQuantity** | Quantity ordered per product. |
| **ExtendedAmount** |  |
| **UnitPriceDiscountPct** | Discount amount per unit |
| **DiscountAmount** | Discount amount |
| **TotalProductCost** | Total Standard cost of the product. |
| **SalesAmount** | Total sales price of product |
| **TaxAmt** | Tax amount. |
| **Freight** | Shipping cost. |
| **OrderDate** | Actual order date |
| **DueDate** | Actual Due Date |
| **ShipDate** | Actual Ship Date |

**Customer.csv**

|  |  |
| --- | --- |
| **Fields** | **Description** |
| **CustomerKey** | Primary key of Customer data |
| **GeographyKey** | Foreign key to Geography data |
| **CustomerAlternateKey** | Alternate key of customer table |
| **Title** | A courtesy title. For example, Mr. or Ms. |
| **CustomerFullName** | Firstname and last name of customer |
| **FirstName** | First name of the person. |
| **MiddleName** | Middle name or middle initial of the person. |
| **LastName** | Last name of the person. |
| **BirthDate** | Birthdate of customer |
| **MaritalStatus** | Married status M,S |
| **Suffix** | Surname suffix. For example, Sr. or Jr. |
| **Gender** | F,M |
| **EmailAddress** | Email address of person |
| **YearlyIncome** | Income of customer |
| **TotalChildren** | Total children of customer |
| **NumberChildrenAtHome** | Number of children at home |
| **Education** | Education of customer |
| **Occupation** | Current occupation of customer |
| **HouseOwnerFlag** | Customer is owner of house |
| **NumberCarsOwned** | Number of cars owned by customer |
| **Phone** | Contact detail of customer |
| **DateFirstPurchase** | Date at which customer bought first product |
| **CommuteDistance** | Distance of commutation |

**ProductHierarchy.csv**

|  |  |
| --- | --- |
| **Fields** | **Description** |
| **ProductKey** | Primary key for **Product** rows. |
| **ProductSubcategoryKey** | Foreign Key to ProductSubcategory |
| **ProductSubcategoryAlternateKey** | Alternate key for ProductSubcategory |
| **ProductCategoryKey** | Foreign Key to ProductCategory |
| **ProductCategoryAlternateKey** | Alternate key for ProductCategory |
| **ProductName** | Name of the product. |
| **ProductDescription** | Description of product |
| **ProductSubcategory** | Product Subcategory Name |
| **ProductCategory** | Product Category Name |
| **ProductModel** | Product Model Name |
| **ProductLine** | R = Road  M = Mountain  T = Touring  S = Standard |
| **WeightUnitMeasureCode** | Unit of measure for Weight column. |
| **SizeUnitMeasureCode** | Unit of measure for **Size** column. |
| **StandardCost** | Standard cost of the product. |
| **FinishedGoodsFlag** | 0 = Product is not a salable item.  1 = Product is salable. |
| **Color** | Product color. |
| **SafetyStockLevel** | Minimum inventory quantity. |
| **ReorderPoint** | Inventory level that triggers a purchase order or work order. |
| **ListPrice** | Selling price. |
| **Size** | Product size. |
| **SizeRange** |  |
| **Weight** | Product weight. |
| **DaysToManufacture** | Number of days required to manufacture the product. |
| **DealerPrice** |  |
| **Class** | H = High  M = Medium  L = Low |
| **Style** | W = Women's  M = Men's  U = Universal |
| **ProductStartDate** | Date the product was available for sale. |
| **ProductEndDate** | Date the product was no longer available for sale. |
| **ProductStatus** |  |

**GeoHierarchy.csv**

|  |  |
| --- | --- |
| **Fields** | **Description** |
| **GeographyKey** | Primary key |
| **City** | city |
| **StateProvinceCode** | State or province code. |
| **StateProvince** | State or province description. |
| **CountryRegionCode** | Standard code identifying countries and regions |
| **CountryRegion** | Country or region name. |
| **PostalCode** | Zipcode of given location |
| **SalesTerritoryKey** | Foreign key to Salesterritory |

# Analysis

## Top 10 Products by TotalSales (Join, Summarization, Filtering Patterns)

Adventure works provides 2 files one consisting of all online sales records containing dates on which sale was made which product was purchased and from whom. The second file is product file which gives the product details and description. The two files are connected via productkey.

Steps

1. First perform reduce side inner join the productkey from fact file is joined with productkey pf product file. The join type is taken from the user as a command line input.
2. Product tuple and SalesProfitTuple class are custome writable class created to hold data from the file.
3. Use a reducer to execute join logic by using inmemory list to store data coming from mapper and then with the complexity of O(n2) performing the inner join between the records.
4. Once the intermediate file is obtained than the data is sent to other mapper through mapreduce chaining job to perform summation of Totalsales and Total profit of individual product using summarization pattern.
5. The output form second reducer is further sent to TopTenMapper to extract top 10 products with highest sales. This uses Top 10 filtering pattern.
6. TopTen reducer to find the top 10 from all the records hence the number of reducer need to be 1 to perform analysis on whole data.

SQL query for the same analysis

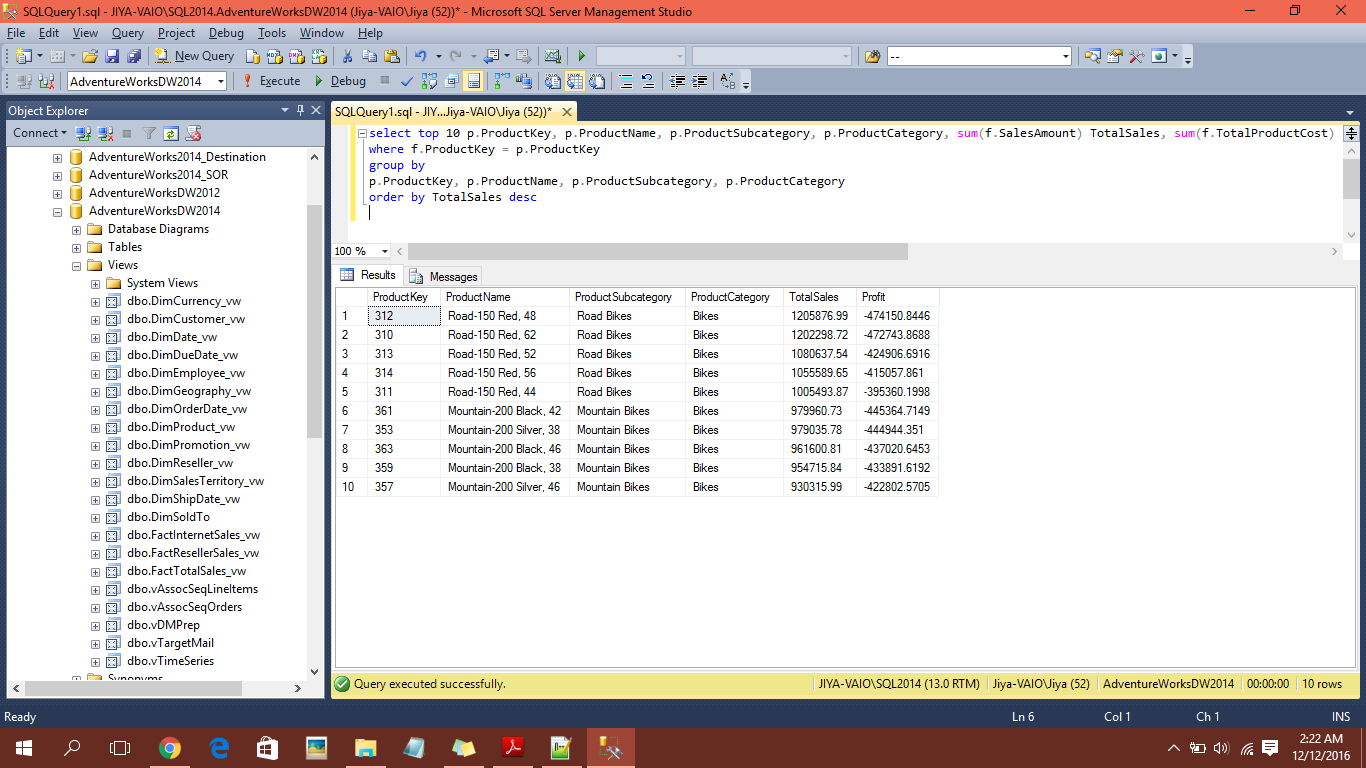


Figure : SQL Query for Top10

**Command To Run Jar**

hadoop jar /home/jiya/Desktop/examples/Project/1\_TopTenProduct/AdventureWorks.jar /jiya/examples/Project/Fact /jiya/examples/Project/Product "inner" /jiya/examples/Project/Output/ttOut1 /jiya/examples/Project/Output/ttout2 /jiya/examples/Project/Output/finalttout

## Total Sales of Product by Geography (Join, Summarization, Binning Pattern)

Adventure works 3 files are used for this analysis. The analysis joins Fact and Customer on customerkey and the resulting file with Geography file on geography key. Hence 3 joins are used to get the resulting data. The further analysis includes summarization pattern to calculate total sales and profit of respective customer. Once the computation is done than further the file is divided into bins based on different Country present. This dividing of files into bins helps to analyze data at deeper level of city for particular country and state.

Steps

1. First perform reduce side inner join the Customerkey from fact file is joined with Customerkey of Customer file. The join type is taken from the user as a command line input.
2. Second the output from step 1 is joined with Geography file in order to get sales based on country, state and city.
3. Use a reducer to execute join logic by using in memory list to store data coming from mapper and then with the complexity of O(n2) performing the inner join between the records.
4. Once the intermediate file is obtained from step 3 than the data is sent to other mapper through mapreduce chaining job to perform summation of Totalsales and Total profit of individual product using summarization pattern.
5. Once the totalSales is calculated the files are divided in bins based on country obtained from Geography file. Hence depending on number of bins defined for country that many files are generated from Binning pattern.

**Command To Run Jar**

hadoop jar /home/jiya/Desktop/examples/Project/2\_MultipleJoinWithBinning/testTuple.jar /jiya/examples/Project/Fact /jiya/examples/Project/Customer "inner" /jiya/examples/Project/Output/OriBinOut1 /jiya/examples/Project/Geo /jiya/examples/Project/Output/OriBinOut2 /jiya/examples/Project/Output/OriBinOut3 /jiya/examples/Project/Output/finalOriBinOut

# Counting Number of Customer based in Geo (Distinct and Counting with Counter pattern)

For this analysis since we already have the join file from above analysis the join between fact, customer and geo file. This file includes customer details and respective product purchased. Hence one can simply use inbuilt counter in MapReduce framework to count number of customer. Here in this analysis number of distinct customer who have placed order based on geography is counted.

Steps

1. First the file is read using mapper. Since here we need distinct customer from each geography hence we need to implement distinct pattern.
2. In the reducer the logic of counting with counter is implemented which counts the users based on geography.

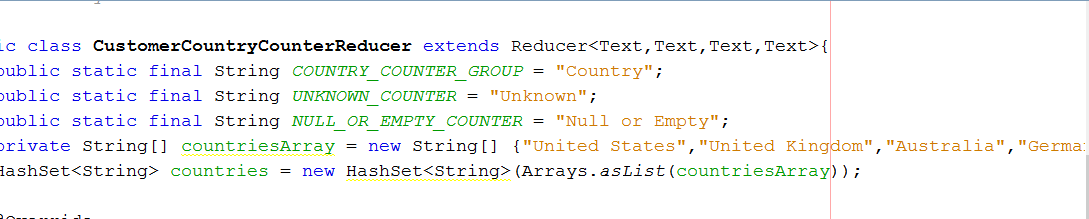


Figure : Counting with Counter

Note: here we are not generating any output file instead the counter value can be seen in MapReduce run statistics

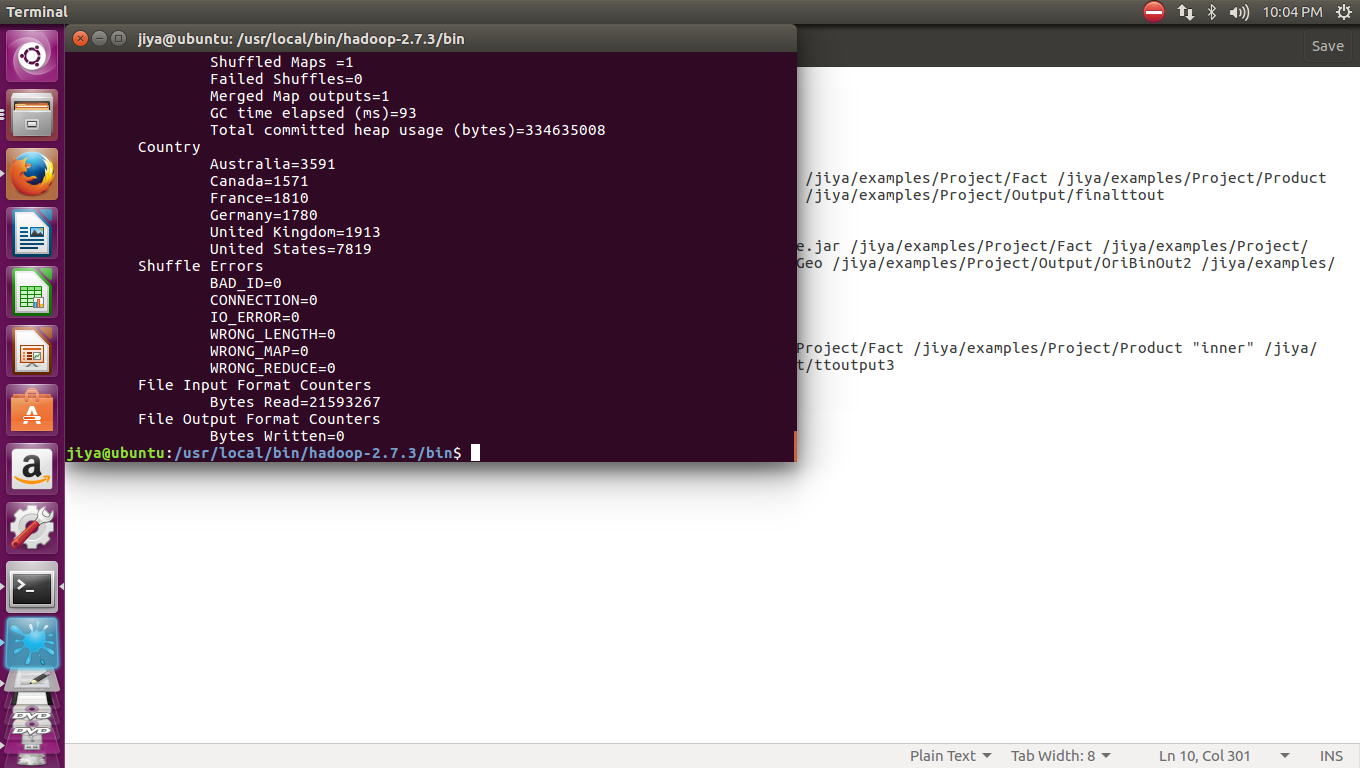


Figure : Output of Counters

**Command To Run Jar**

hadoop jar /home/jiya/Desktop/examples/Project/3\_CountingCustomerWithCounter/CustomerCounterExample.jar /jiya/examples/Project/Output/OriBinOut2 /jiya/examples/Project/Output/finalCounter

# Predictive yearly Sales Analysis (Summarization pattern)

For this analysis I have used summarization pattern to implement standard deviation and used this standard deviation to calculate the Yearly sales for next year. Since standard deviation gives us the value of how much the sales got deviated from its original value this is a good calculation technique to predict the approximate value of next year sales.

Steps

1. First the fact file is parsed and year and total sales are extracted.
2. Second the data extracted by the mapper is passed to the reducer to calculate Total sales in each year.
3. This file generated in second step is send to other mapper using mapreduce chaining for further predictive analysis
4. In the second job only one reducer is used since complete data need to be present in order to calculate the standard deviation of all records
5. Once the standard deviation is calculated it is added to the last sales and displayed as predictive sales for next year.

**Command To Run Jar**

hadoop jar /home/jiya/Desktop/examples/Project/4\_PredictiveYearlySales/PredictiveYearlySales.jar /jiya/examples/Project/Fact /jiya/examples/Project/Output/predictOut /jiya/examples/Project/Output/finalPredictOut

# Analyzing Products promoted based on Specific reason (Inverted Index)

For this analysis inverted index a type of summarization pattern is used. I idea behind creating inverted index on productkeys based on promotion reason is the user will always wants to search which are product were rejected based on which reason and for which reason what all products got rejected. Hence creating an index on list of product keys would help to increase the search time for the user.

Steps

1. First the promotion type and product keys are extracted from fact file.
2. In the reduce phase all the productkeys found for particular promotion reasons are concatenated.
3. Here I have made sure no duplicate value for product keys should be appended hence HashSet are used to ensure the same.

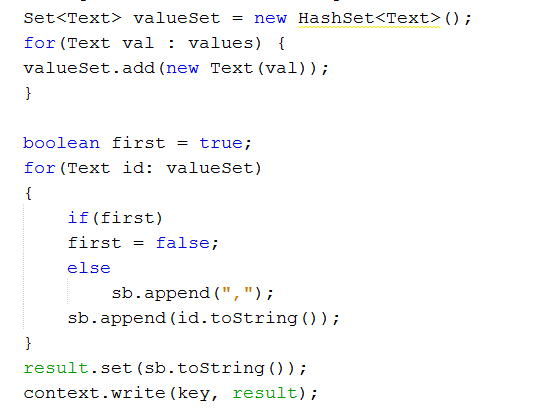


Figure : Code Snippet for Inverted Index

**Command To Run Jar**

hadoop jar /home/jiya/Desktop/examples/Project/5\_InvertedIndex/ProductPromotionInvertedIndex.jar /jiya/examples/Project/Fact /jiya/examples/Project/Output/finalInvOut

# Performance Improvement

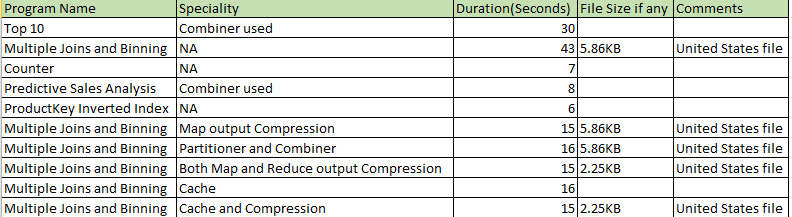


Figure : Performance Comparison of various analysis

## Sending Less Data over network

It has been observed when it comes to joining files the whole record of the file read from the mapper is sent over the network instead of only necessary attribute. This in turn creates a huge overhead and makes the work of reducer tougher to parse through whole line.

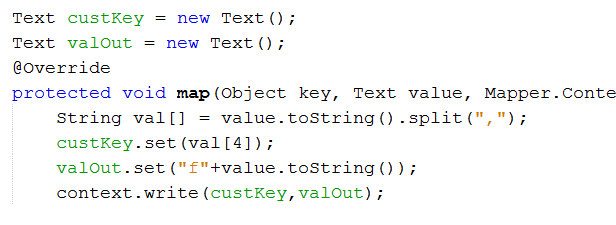


Figure : Original Map output

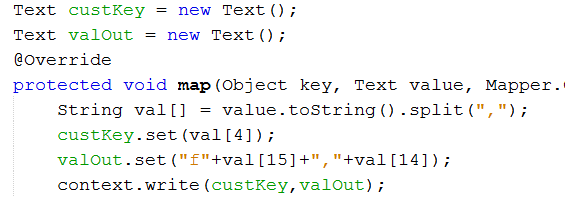


Figure : Improved Map Output

Definitely the second will improve the performance in terms of parsing the whole values vs parsing only few fields in reducer.

## Use of Partitioner and Combiner

MapReduce can be really made effective if the logic of partitioner and combiner used judiciously. Partitioner helps the data to equally distribute among the reducer which in turn helps in reducing the load on internal shuffler of identifying the key distribution to particular reducer. Combiner helps in reducing the calculation load on reducer. The map output goes to the combiner who generate intermediate sum and reduce the load on reducer to traverse to the keys again and use this intermediate sum to calculate total sum. Hence combiners are effective when it comes to computations which are associative and commutative.

The Analysis in previous map reduce program of calculating total sales and profit is enhanced using proper use of partitioner and combiner. Here the partitioner is used to send all the keys which are Country name here to the same reducer. Hence reducing the load on reducer to aggregate based on country. Combiner is used here to calculate intermediate sales and profit to avoid load on reducer.

The beauty behind using this is MapReduce provide easy configuration for the same.

**Command To Run Jar**

hadoop jar /home/jiya/Desktop/examples/Project/6\_OnlyPartitionerCombiner/ImprovedMultipleJoins.jar /jiya/examples/Project/Fact /jiya/examples/Project/Customer "inner" /jiya/examples/Project/Output/PartBinOut1 /jiya/examples/Project/Geo /jiya/examples/Project/Output/PartBinOut2 /jiya/examples/Project/Output/PartBinOut3 /jiya/examples/Project/Output/finalPartBinOut

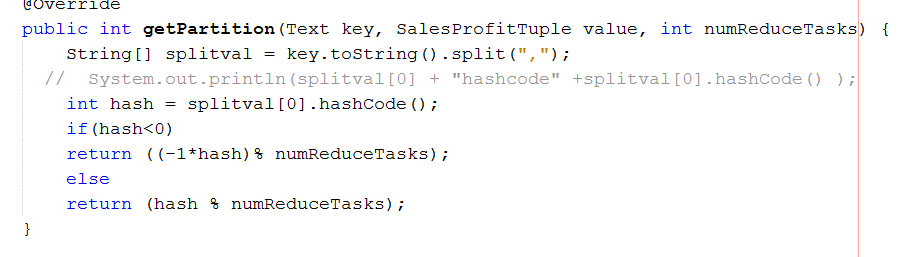


Figure : Code Snippet Partitioner

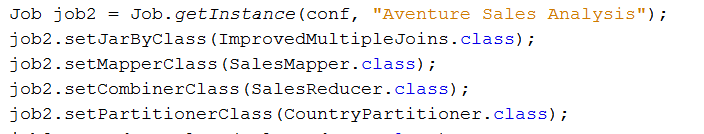


Figure : Cod Snippet of Configuring Combiner and Partitioner

## Input/Output file Compression

MapReduce provides another great configuration feature of compressing input output file. One can either compress the input file read by map or compress the output file produced at each intermediate and reducer phase produced by reducer in order to store compressed file in HDFS resulting in better storage in HDFS. Also this is easy to configure. One can use various compression techniques to improve the performance further but by default MapReduce provides codec compression.

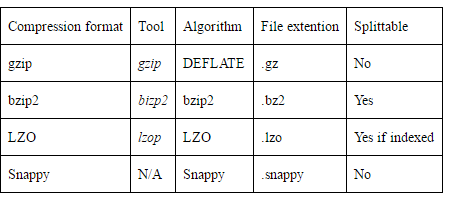


Figure : Compression Format supported by Hadoop

The gzip format uses DEFLATE to store the compressed data, and DEFLATE stores data as a series of compressed blocks. It creates a sequence file hence prohibiting user to split the file hence it is accessed only by synchronization. This uses Huffman coding algorithm to compress data. **Huffman coding** is a compression technique used to reduce the number of bits needed to send or store a message. It's based on the idea that frequently-appearing letters should have shorter bit representations and less common letters should have longer representations

User can easily implement this by setting few parameter in job configuration.

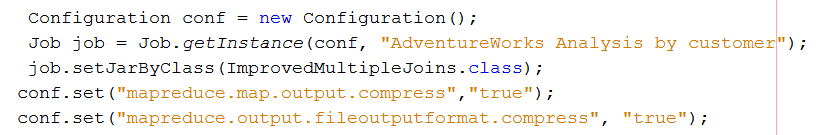


Figure : Configuring file Compression

By setting the map output and file output to true will ensure the files are compressed before and stored in HDFS. This improves the performance of MR jobs and also resolves the storage issues on HDFS

**Command To Run Jar**

MapCompression

hadoop jar /home/jiya/Desktop/examples/Project/7\_MapCompression/ImprovedMultipleJoins.jar /jiya/examples/Project/Fact /jiya/examples/Project/Customer "inner" /jiya/examples/Project/Output/MapCompBinOut1 /jiya/examples/Project/Geo /jiya/examples/Project/Output/MapCompBinOut2 /jiya/examples/Project/Output/MapCompBinOut3 /jiya/examples/Project/Output/finalMapCompBinOut

BothMapAndReduceCompression

hadoop jar /home/jiya/Desktop/examples/Project/8\_BothCompression/ImprovedMultipleJoins.jar /jiya/examples/Project/Fact /jiya/examples/Project/Customer "inner" /jiya/examples/Project/Output/BothCompBinOut1 /jiya/examples/Project/Geo /jiya/examples/Project/Output/BothCompBinOut2 /jiya/examples/Project/Output/BothCompBinOut3 /jiya/examples/Project/Output/finalBothCompBinOut

## Multiple Joins using Caching

Mapreduce provides one of the best features of distributed cache. Here in my case this cache is useful while joining multiple files. If one knows the keys on which multiple files will be joined together than one can simply join the smaller file in cache and use the cache for further join with large file. In the example of multiple join I have configured cache property in job for storing geo file and while reading the product file in mapper in setup phase I am retrieving only those product records which has geography key present in the cache if not present simply that product is removed and not sent forward. This improves number of records sent over the network and hence improving the performance.

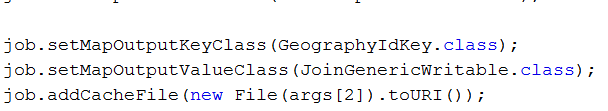


Figure : Cache Property in job

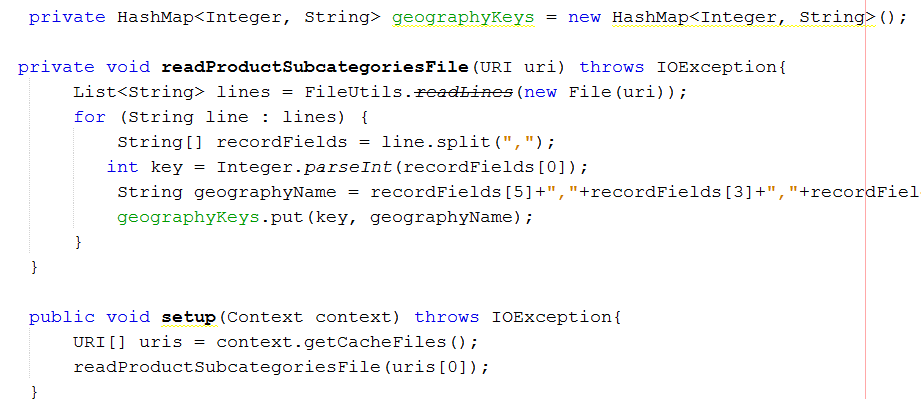


Figure : Reading cached file in setup phase of map

One can check whether geography is present in the hashmap with the complexity of O(n). Hence performance is improved when it comes to multiple join. Only disadvantage of this is one cannot cache file with larger size.

**Command To Run Jar**

hadoop jar /home/jiya/Desktop/examples/Project/9\_MultipleJoinWithCache/MultipleJoinUsingCache.jar /jiya/examples/Project/Fact /jiya/examples/Project/Customer /home/jiya/Desktop/examples/Project/Geo.csv /jiya/examples/Project/Output/CacheBinOut1 /jiya/examples/Project/Output/CacheBinOut2 /jiya/examples/Project/Output/finalCacheBinOut

## GenericWritable vs Writable

A wrapper for Writable instances.  
When two sequence files, which have same Key type but different Value types, are mapped out to reduce, multiple Value types is not allowed. In this case, this class can help you wrap instances with different types.  
Compared with ObjectWritable, this class is much more effective, because ObjectWritable will append the class declaration as a String into the output file in every Key-Value pair.  
Generic Writable implements Configurable interface, so that it will be configured by the framework. The configuration is passed to the wrapped objects implementing Configurable interface before deserialization.

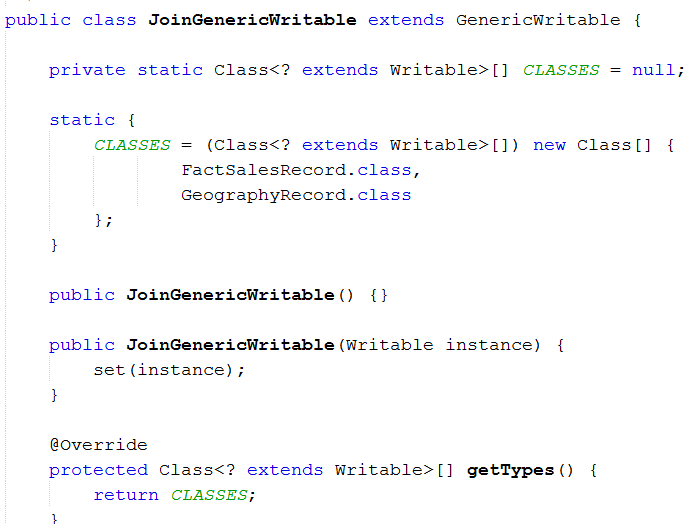


Figure : Generic Writable Class

Using such writable class avoids the use of multiple mapper reducers and mapreduce chaining for joining data of different type also avoids the unnecessary typecasting of value in Text format to maintain the compatibility.

# Visual Representation

1. Input Data Analysis

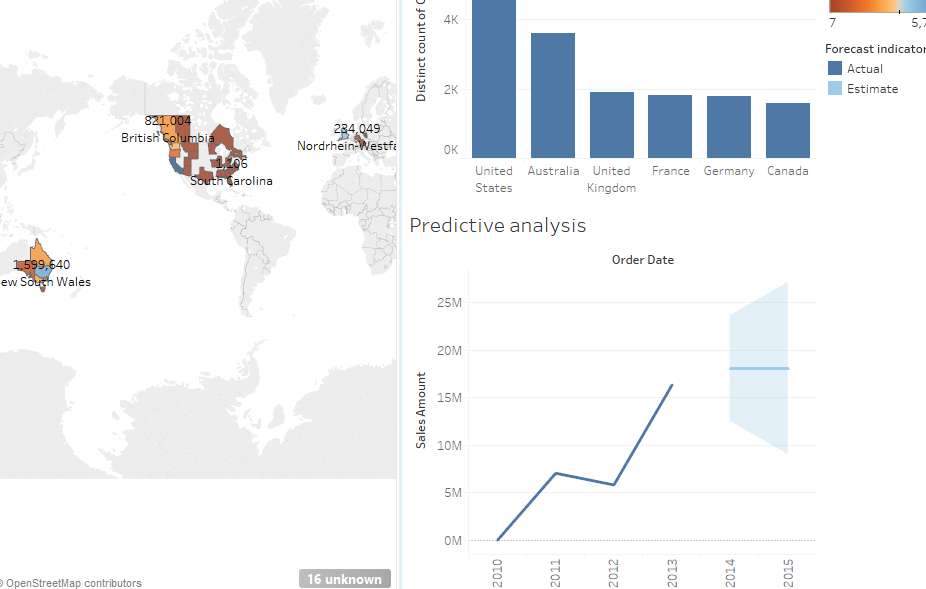


Figure : Dashboard of Input data analysis

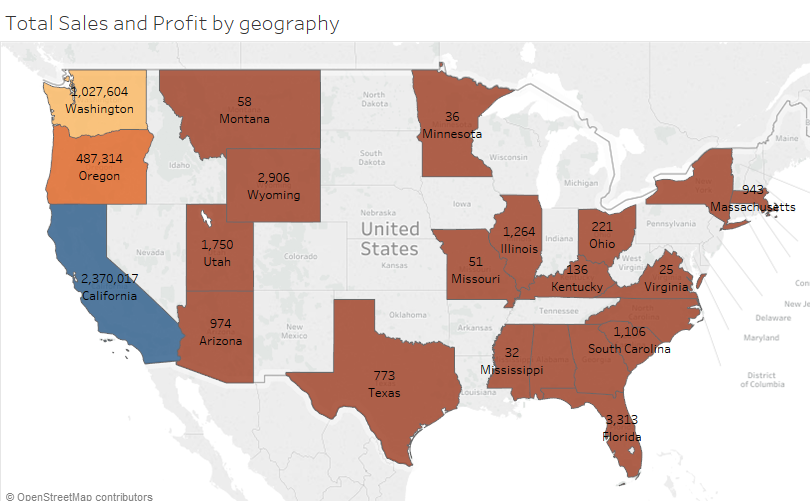


Figure :TotalSales and Profit of United States

1. Output file Analysis

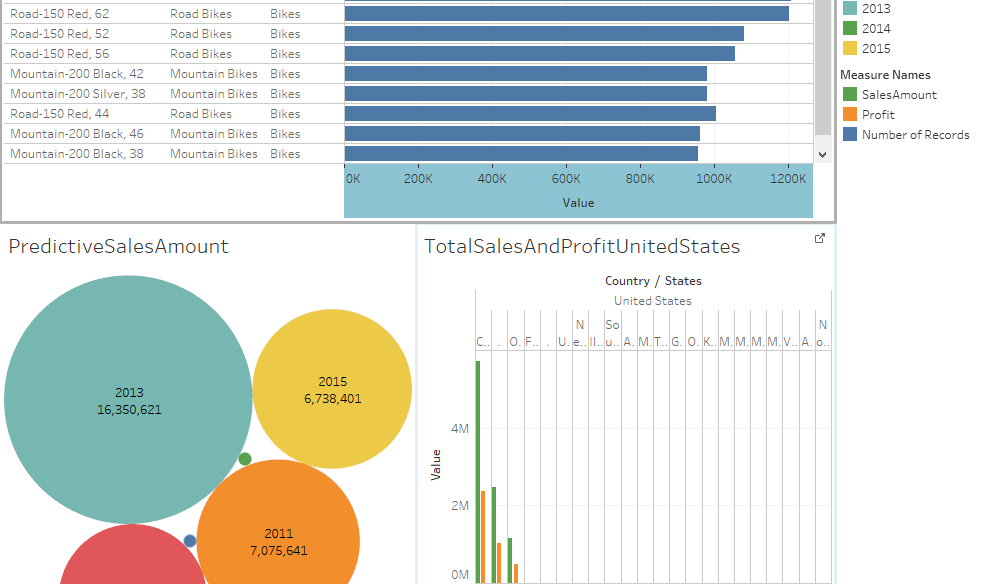


Figure : Dashboard for output Analysis

# HBase

HBase provides a better storage for HDFS file. Also One can implement indexing on certain column to make retrieval of record faster. Since HBase stores everything in column family it is easier to retrieve any particular value of column.

Table creation

create 'invtab2','productkey'

Importing value from csv to hbase

hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=,  -Dimporttsv.columns="HBASE\_ROW\_KEY,productkey" invtab2 hdfs://localhost:9000/jiya/examples/Project/invtest.csv  
  
scanning a column value

scan 'invtab2', { COLUMNS => 'productkey', LIMIT => 10, FILTER => "ValueFilter( =, 'regexstring:480' )" }

# Appendix

## Code for Top 10 Products by TotalSales (Join, Summarization, Filtering Patterns)

1. FactMapper

package adventureworks;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

Text productKey = new Text();

Text valOut = new Text();

@Override

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

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

productKey.set(val[0]);

valOut.set("f"+value.toString());

context.write(productKey,valOut);

}

}

1. ProductMapper

package adventureworks;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

Text productKey = new Text();

Text valOut = new Text();

@Override

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

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

productKey.set(val[0]);

valOut.set("p"+value.toString());

context.write(productKey,valOut);

}

}

1. FactProduct Join Reducer

package adventureworks;

import java.io.IOException;

import java.util.ArrayList;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

private static int count = 0;

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

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

private String joinType = null;

private Text tmp = new Text();

public void setup(Context context)

{

/\* Get the type of join from our configuration \*/

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

}

@Override

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

{

listFact.clear();

listProduct.clear();

/\* iterate through all our values, binning each record based on what

it was tagged with. Make sure to remove the tag! \*/

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

{

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

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

{

listFact.add(new Text(tmp.toString().substring(1)));

}

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

{

listProduct.add(new Text(tmp.toString().substring(1)));

}

}

// Execute our join logic now that the lists are filled

executeJoinLogic(context);

}

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

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

// If both lists are not empty, join A with B

if (!listFact.isEmpty() && !listProduct.isEmpty()) {

for (Text fact : listFact) {

for (Text prod : listProduct) {

context.write(fact,new Text(","+prod));

}

}

}

}

}

}

1. SalesMapper

package adventureworks;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class SalesMapper extends Mapper<Object,Text,Text,SalesProfitTuple> {

Text p = new Text();

@Override

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

String otherThanQuote = " [^\"] ";

String quotedString = String.format(" \" %s\* \" ", otherThanQuote);

String regex = String.format("(?x) "+ // enable comments, ignore white spaces

", "+ // match a comma

"(?= "+ // start positive look ahead

" ( "+ // start group 1

" %s\* "+ // match 'otherThanQuote' zero or more times

" %s "+ // match 'quotedString'

" )\* "+ // end group 1 and repeat it zero or more times

" %s\* "+ // match 'otherThanQuote'

" $ "+ // match the end of the string

") ", // stop positive look ahead

otherThanQuote, quotedString, otherThanQuote);

String val[] = value.toString().split(regex, -1);

ProductTuple pt = new ProductTuple();

pt.setProductID(Integer.parseInt(val[0]));

pt.setProductCategory(val[29]);

pt.setProductName(val[26]);

pt.setProductSubCategory(val[28]);

p.set(pt.toString());

float salesAmt =Float.parseFloat(val[15]);

float cost = Float.parseFloat(val[14]);

float profit = salesAmt - cost;

SalesProfitTuple spt = new SalesProfitTuple();

spt.setSalesAmt(salesAmt);

spt.setProfitMargin(profit);

context.write(p,spt);

}

}

1. SalesReducer

package adventureworks;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class SalesReducer extends Reducer<Text,SalesProfitTuple,Text,SalesProfitTuple>{

SalesProfitTuple result = new SalesProfitTuple();

@Override

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

float sum = 0;

float profit = 0;

for(SalesProfitTuple s : values)

{

sum += s.getSalesAmt();

profit += s.getProfitMargin();

}

result.setProfitMargin(profit);

result.setSalesAmt(sum);

context.write(key, result); }}

1. Top10 Mapper

package adventureworks;

import java.io.IOException;

import java.util.TreeMap;

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 TopTenMapper extends Mapper<LongWritable, Text,NullWritable,Text> {

public TreeMap<Float, Text> topValue=new TreeMap<Float, Text>();

@Override

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

String[] splitVal = value.toString().split("\t");

String salesAmount = splitVal[4];

topValue.put(Float.parseFloat(salesAmount),new Text(value.toString()));

if(topValue.size()>10)

{

topValue.remove(topValue.firstKey());

}

}

@Override

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

for(Text value:topValue.values())

{

context.write(NullWritable.get(),new Text(value.toString()));

}

}

}

1. Top10 Reducer

package adventureworks;

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.Reducer;

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

public TreeMap<Float, Text> topValue=new TreeMap<Float, Text>();

@Override

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

for(Text value:values)

{

String[] splitVal=value.toString().split("\t");

String salesAmount = splitVal[4];

topValue.put(Float.parseFloat(salesAmount), new Text(value.toString()));

if(topValue.size()>10)

{

topValue.remove(topValue.firstKey());

}

}

for(Text value:topValue.values())

{

context.write(NullWritable.get(),new Text(value.toString()));

}

}

}

1. ProductTuple

package adventureworks;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class ProductTuple implements Writable {

int productID;

String productName;

String productSubCategory;

String productCategory;

public int getProductID() {

return productID;

}

public void setProductID(int productID) {

this.productID = productID;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public String getProductSubCategory() {

return productSubCategory;

}

public void setProductSubCategory(String productSubCategory) {

this.productSubCategory = productSubCategory;

}

public String getProductCategory() {

return productCategory;

}

public void setProductCategory(String productCategory) {

this.productCategory = productCategory;

}

@Override

public void write(DataOutput d) throws IOException {

d.write(productID);

d.writeChars(productName);

d.writeChars(productSubCategory);

d.writeChars(productCategory);

}

@Override

public void readFields(DataInput di) throws IOException {

productID = di.readInt();

productName = di.readLine();

productSubCategory = di.readLine();

productCategory = di.readLine();

}

@Override

public String toString()

{

return productID +"\t"+productName + "\t" + productSubCategory +"\t" + productCategory;

}

}

1. SalesProfitTuple

package adventureworks;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class SalesProfitTuple implements Writable {

private float salesAmt;

private float profitMargin;

public float getSalesAmt() {

return salesAmt;

}

public void setSalesAmt(float salesAmt) {

this.salesAmt = salesAmt;

}

public float getProfitMargin() {

return profitMargin;

}

public void setProfitMargin(float profitMargin) {

this.profitMargin = profitMargin;

}

@Override

public void write(DataOutput d) throws IOException {

d.writeFloat(salesAmt);

d.writeFloat(profitMargin);

}

@Override

public void readFields(DataInput di) throws IOException {

salesAmt = di.readFloat();

profitMargin = di.readFloat();

}

@Override

public String toString()

{

return salesAmt +"\t" + profitMargin;

}

}

1. DriverCode

package adventureworks;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

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.input.MultipleInputs;

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

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

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

public class AdventureWorks {

public static void main(String[] args) {

// TODO code application logic here

try

{

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "AdventureWorks Analysis");

job.setJarByClass(AdventureWorks.class);

MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, AdventureMapper.class);

MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, ProductMapper.class);

job.getConfiguration().set("join.type", args[2]);

//job.setNumReduceTasks(0);

job.setReducerClass(ProductReducer.class);

job.setOutputFormatClass(TextOutputFormat.class);

TextOutputFormat.setOutputPath(job, new Path(args[3]));

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.waitForCompletion(true);

/\*Job2\*/

Job job2 = Job.getInstance(conf, "Aventure Sales Analysis");

job2.setJarByClass(AdventureWorks.class);

job2.setMapperClass(SalesMapper.class);

job2.setCombinerClass(SalesReducer.class);

job2.setReducerClass(SalesReducer.class);

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(SalesProfitTuple.class);

FileInputFormat.addInputPath(job2, new Path(args[3]));

FileOutputFormat.setOutputPath(job2, new Path(args[4]));

job2.waitForCompletion(true);

Job job3 = Job.getInstance(conf, "Top 10 Aventure Sales Analysis");

job3.setJarByClass(AdventureWorks.class);

job3.setMapperClass(TopTenMapper.class);

job3.setReducerClass(TopTenReducer.class);

job3.setNumReduceTasks(1);

job3.setOutputKeyClass(NullWritable.class);

job3.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job3, new Path(args[4]));

FileOutputFormat.setOutputPath(job3, new Path(args[5]));

System.exit(job3.waitForCompletion(true) ? 0 : 1);

}

catch(IOException e)

{

System.out.println("Exception in IO main "+e.getMessage());

}

catch(InterruptedException e)

{

System.out.println("Exception in interrupted main "+e.getMessage());

}

catch(ClassNotFoundException e)

{

System.out.println("Exception in class not found main "+e.getMessage());

}

}

}

## Code for Total Sales of Product by Geography (Join, Summarization, Binning Pattern)

1. FactCustomerMapper

package testtuple;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

Text custKey = new Text();

Text valOut = new Text();

@Override

protected void map(Object key, Text value, Mapper.Context context) throws IOException, InterruptedException {

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

custKey.set(val[4]);

valOut.set("f"+value.toString());

context.write(custKey,valOut);

}

}

1. CustomerMapper

package testtuple;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

Text custKey = new Text();

Text valOut = new Text();

@Override

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

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

custKey.set(val[0]);

valOut.set("c"+value.toString());

context.write(custKey,valOut);

}

}

1. CustomerReducer

package testtuple;

import java.io.IOException;

import java.util.ArrayList;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

private static int count = 0;

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

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

private String joinType = null;

private Text tmp = new Text();

public void setup(Reducer.Context context)

{

// Get the type of join from our configuration

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

}

@Override

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

{

listFact.clear();

listCust.clear();

// iterate through all our values, binning each record based on what

// it was tagged with. Make sure to remove the tag!

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

{

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

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

{

listFact.add(new Text(tmp.toString().substring(1)));

}

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

{

listCust.add(new Text(tmp.toString().substring(1)));

}

}

// Execute our join logic now that the lists are filled

executeJoinLogic(context);

}

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

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

// If both lists are not empty, join A with B

if (!listFact.isEmpty() && !listCust.isEmpty()) {

for (Text fact : listFact) {

for (Text cust : listCust) {

context.write(fact,","+cust);

}

}

}

}

}

}

1. CountryMapper

package testtuple;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

Text geoKey = new Text();

Text valOut = new Text();

@Override

protected void map(Object key, Text value, Mapper.Context context) throws IOException, InterruptedException {

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

geoKey.set(val[0]);

valOut.set("c"+value.toString());

context.write(geoKey,valOut);

}

}

1. Intermeadiate File mapper

package testtuple;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

Text geoKey = new Text();

Text valOut = new Text();

@Override

protected void map(Object key, Text value, Mapper.Context context) throws IOException, InterruptedException {

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

geoKey.set(val[22]);

valOut.set("f"+value.toString());

context.write(geoKey,valOut);

}

}

1. CountryReducer

package testtuple;

import java.io.IOException;

import java.util.ArrayList;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

private static int count = 0;

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

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

private String joinType = null;

private Text tmp = new Text();

public void setup(Context context)

{

// Get the type of join from our configuration

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

}

@Override

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

{

listFactProd.clear();

listGeo.clear();

// iterate through all our values, binning each record based on what

// it was tagged with. Make sure to remove the tag!

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

{

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

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

{

listFactProd.add(new Text(tmp.toString().substring(1)));

}

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

{

listGeo.add(new Text(tmp.toString().substring(1)));

}

}

// Execute our join logic now that the lists are filled

executeJoinLogic(context);

//System.out.println("Count:"+count);

}

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

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

// If both lists are not empty, join A with B

//System.out.println("here3");

if (!listFactProd.isEmpty() && !listGeo.isEmpty()) {

for (Text fact : listFactProd) {

//System.out.println("here1");

for (Text geo : listGeo) {

context.write(fact,new Text(","+geo));

}

}

}

}

}

}

1. SalesMapper, SalesReducer and SalesProfitTuple

Same as Top10 code

1. CountryBinMapper

package testtuple;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

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 CountryBinMapper extends Mapper<Object,Text,Text,NullWritable>{

private MultipleOutputs<Text, NullWritable> mos = null;

@Override

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

mos = new MultipleOutputs(context);

}

@Override

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

String splitVal = value.toString();

if(splitVal.contains("United States"))

{

mos.write("Country", new Text(splitVal), NullWritable.get(),"United States");

}

else if(splitVal.contains("United Kingdom"))

{

mos.write("Country", new Text(splitVal), NullWritable.get(),"United Kingdom");

}

else if(splitVal.contains("Australia"))

{

mos.write("Country", new Text(splitVal), NullWritable.get(),"Australia");

}

else if(splitVal.contains("Canada"))

{

mos.write("Country", new Text(splitVal), NullWritable.get(),"Canada");

}

else if(splitVal.contains("France"))

{

mos.write("Country", new Text(splitVal), NullWritable.get(),"France");

}

else if(splitVal.contains("Germany"))

{

mos.write("Country", new Text(splitVal), NullWritable.get(),"Germany");

}

}

@Override

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

mos.close();

}

}

1. DriverCode

package testtuple;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

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.input.MultipleInputs;

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 TestTuple {

public static void main(String[] args) {

try

{

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "AdventureWorks Analysis by customer");

job.setJarByClass(TestTuple.class);

MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, FactProdMapper.class);

MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, CustomerMapper.class);

job.getConfiguration().set("join.type", args[2]);

//job.setNumReduceTasks(0);

job.setReducerClass(CustomerReducer.class);

job.setOutputFormatClass(TextOutputFormat.class);

TextOutputFormat.setOutputPath(job, new Path(args[3]));

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.waitForCompletion(true);

/\*Job 2\*/

Job job1 = Job.getInstance(conf, "AdventureWorks Analysis by Country");

job1.setJarByClass(TestTuple.class);

MultipleInputs.addInputPath(job1, new Path(args[3]), TextInputFormat.class, IntermediateMapper.class);

MultipleInputs.addInputPath(job1, new Path(args[4]), TextInputFormat.class, CountryMapper.class);

job1.getConfiguration().set("join.type",args[2]);

//job.setNumReduceTasks(0);

job1.setReducerClass(CustomerReducer.class);

job1.setOutputFormatClass(TextOutputFormat.class);

TextOutputFormat.setOutputPath(job1, new Path(args[5]));

job1.setOutputKeyClass(Text.class);

job1.setOutputValueClass(Text.class);

job1.waitForCompletion(true);

/\*Job3\*/

Job job2 = Job.getInstance(conf, "Aventure Sales Analysis");

job2.setJarByClass(TestTuple.class);

job2.setMapperClass(SalesMapper.class);

job2.setCombinerClass(SalesReducer.class);

job2.setReducerClass(SalesReducer.class);

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(SalesProfitTuple.class);

FileInputFormat.addInputPath(job2, new Path(args[5]));

FileOutputFormat.setOutputPath(job2, new Path(args[6]));

job2.waitForCompletion(true);

/\*job4\*/

Job job3 = Job.getInstance(conf, "Aventure Sales Analysis by binning");

job3.setJarByClass(TestTuple.class);

job3.setMapperClass(CountryBinMapper.class);

job3.setNumReduceTasks(0);

MultipleOutputs.addNamedOutput(job3, "Country", TextOutputFormat.class, Text.class, NullWritable.class);

MultipleOutputs.setCountersEnabled(job3, true);

FileInputFormat.addInputPath(job3, new Path(args[6]));

FileOutputFormat.setOutputPath(job3, new Path(args[7]));

System.exit(job3.waitForCompletion(true) ? 0 : 1);

}

catch(IOException e)

{

System.out.println("Exception in IO main "+e.getMessage());

}

catch(InterruptedException e)

{

System.out.println("Exception in interrupted main "+e.getMessage());

}

catch(ClassNotFoundException e)

{

System.out.println("Exception in class not found main "+e.getMessage());

}

}

}

## Code for Counting Number of Customer based in Geo (Distinct and Counting with Counter pattern)

1. CustomerCountryCounter

package customercounterexample;

import java.io.IOException;

import java.util.Arrays;

import java.util.HashSet;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

@Override

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

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

String country = values[50];

String customerKey = values[4];

context.write(new Text(customerKey), new Text(country));

}

}

1. CustomerCountryCounterReducer

package customercounterexample;

import java.io.IOException;

import java.util.Arrays;

import java.util.HashSet;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

public static final String COUNTRY\_COUNTER\_GROUP = "Country";

public static final String UNKNOWN\_COUNTER = "Unknown";

public static final String NULL\_OR\_EMPTY\_COUNTER = "Null or Empty";

private String[] countriesArray = new String[] {"United States","United Kingdom","Australia","Germany","France","Canada"};

HashSet<String> countries = new HashSet<String>(Arrays.asList(countriesArray));

@Override

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

String country = "";

for(Text val : values)

{

country = val.toString();

break;

}

if(country != null && !country.isEmpty() )

{

boolean unknown = true;

if(countries.contains(country))

{

context.getCounter(COUNTRY\_COUNTER\_GROUP, country).increment(1);

unknown = false;

}

if(unknown)

{

context.getCounter(UNKNOWN\_COUNTER, country).increment(1);

}

}

else

{

context.getCounter(UNKNOWN\_COUNTER, country).increment(1);

}

}}

1. DriverCode

package customercounterexample;

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.Counter;

import org.apache.hadoop.mapreduce.Job;

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

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

public class CustomerCounterExample {

public static void main(String[] args) {

try

{

Configuration conf = new Configuration();

Job job = new Job(conf, "AdventureWorks Number of Customer by Country");

job.setJarByClass(CustomerCounterExample.class);

job.setMapperClass(CustomerCountryCounterMapper.class);

job.setReducerClass(CustomerCountryCounterReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

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

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

boolean success = job.waitForCompletion(true);

if (success) {

for (Counter counter : job.getCounters().getGroup(

CustomerCountryCounterReducer.COUNTRY\_COUNTER\_GROUP)) {

System.out.println(counter.getDisplayName() + "\t"

+ counter.getValue());

}

}

FileSystem.get(conf).delete(outputDir);

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

}

catch(IOException e)

{

System.out.println("Exception in IO main "+e.getMessage());

}

catch(InterruptedException e)

{

System.out.println("Exception in interrupted main "+e.getMessage());

}

catch(ClassNotFoundException e)

{

System.out.println("Exception in class not found main "+e.getMessage());

}

}

}

## Code for Predictive yearly Sales Analysis (Summarization pattern)

1. FactMapper

package predictiveyearlysales;

import java.io.IOException;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

Text medstddev = new Text();

@Override

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

String[] itr = value.toString().split("\t");

medstddev.set(itr[0]+"\t"+itr[1]);

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

}

}

1. StandardDeviationReducer

package predictiveyearlysales;

import java.io.IOException;

import java.util.Map.Entry;

import java.util.TreeMap;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

private TreeMap<String, Float> salesAmt = new TreeMap<String,Float>();

private Text result = new Text();

@Override

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

float sum = 0;

float totalCount = 0;

salesAmt.clear();

for(Text val: values)

{

String[] splitval = val.toString().split("\t");

salesAmt.put(splitval[0],Float.parseFloat(splitval[1]));

totalCount++;

sum += Float.parseFloat(splitval[1]);

}

//std deviation

float mean = sum/totalCount;

float sumOfSquares = 0.0f;

for(Entry<String,Float> entry : salesAmt.entrySet())

{

sumOfSquares += (entry.getValue() - mean)\*(entry.getValue() - mean);

}

float stdDev = (float) Math.sqrt(sumOfSquares/(totalCount-1));

Entry<String,Float> ent = salesAmt.lastEntry();

String nextYear = String.valueOf(Integer.parseInt(ent.getKey())+1);

float predictedSalesAmt = ent.getValue() + stdDev;

salesAmt.put(nextYear,predictedSalesAmt);

for(Entry<String,Float> entry : salesAmt.entrySet())

{

context.write(NullWritable.get(), new Text(entry.getKey() +"\t"+ entry.getValue()));

}

}

}

1. PredictiveSalesMapper

package predictiveyearlysales;

import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class YearlySalesMapper extends Mapper<Object,Text,Text,FloatWritable>{

Text year = new Text();

private FloatWritable salesAmt = new FloatWritable();

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

@Override

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

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

if(val[0].equalsIgnoreCase("ProductKey"))

{

}

else

{

year.set(val[1].substring(0, 4));

salesAmt.set(Float.parseFloat(val[15]));

context.write(year,salesAmt);

}

}

}

1. PredictiveSalesReducer

package predictiveyearlysales;

import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

FloatWritable totalSum = new FloatWritable();

@Override

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

float sum = 0;

for(FloatWritable val : values)

{

sum += val.get();

}

totalSum.set(sum);

context.write(new Text(key),totalSum);

}

}

1. DriverCode

package predictiveyearlysales;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.FloatWritable;

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.MultipleInputs;

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

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

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

public class PredictiveYearlySales {

public static void main(String[] args) {

try

{

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Total sales Analysis");

job.setJarByClass(PredictiveYearlySales.class);

job.setMapperClass(YearlySalesMapper.class);

job.setCombinerClass(YearlySalesReducer.class);

job.setReducerClass(YearlySalesReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(FloatWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.waitForCompletion(true);

/\*Job2\*/

Job job2 = Job.getInstance(conf, "Predictive Sales Analysis");

job2.setJarByClass(PredictiveYearlySales.class);

job2.setMapperClass(MedStdDevMapper.class);

job2.setReducerClass(MedStdDevReducer.class);

job2.setOutputKeyClass(NullWritable.class);

job2.setOutputValueClass(Text.class);

job2.setNumReduceTasks(1);

FileInputFormat.addInputPath(job2, new Path(args[1]));

FileOutputFormat.setOutputPath(job2, new Path(args[2]));

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

}

catch(IOException e)

{

System.out.println("Exception in IO main "+e.getMessage());

}

catch(InterruptedException e)

{

System.out.println("Exception in interrupted main "+e.getMessage());

}

catch(ClassNotFoundException e)

{

System.out.println("Exception in class not found main "+e.getMessage());

}

}

}

## Code for Analyzing Products promoted based on Specific reason (Inverted Index)

1. FactProductMapper

package productpromotioninvertedindex;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class FactCustomerMapper extends Mapper<LongWritable, Text, Text,Text>{

Text promotion = new Text();

Text productKey = new Text();

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

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

if(recordFields[0].equalsIgnoreCase("ProductKey"))

{

}

else

{

String promotionType = "";

int promoId = Integer.parseInt(recordFields[5]);

if(promoId == 1)

promotionType = "No Discount";

else if(promoId == 13 || promoId == 14)

promotionType = "New Product";

else if(promoId == 2)

promotionType = "Volume Discount";

promotion.set(promotionType);

productKey.set(recordFields[0]);

context.write(promotion, productKey);

}

}

}

1. InvertedIndexReducer

package productpromotioninvertedindex;

import java.io.IOException;

import java.util.HashSet;

import java.util.Set;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

Text result = new Text();

@Override

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

StringBuilder sb = new StringBuilder();

Set<Text> valueSet = new HashSet<Text>();

for(Text val : values) {

valueSet.add(new Text(val));

}

boolean first = true;

for(Text id: valueSet)

{

if(first)

first = false;

else

sb.append(",");

sb.append(id.toString());

}

result.set(sb.toString());

context.write(key, result);

}

}

1. DriverCode

package productpromotioninvertedindex;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

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.FileInputFormat;

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

public class ProductPromotionInvertedIndex {

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Aventure Sales Analysis");

job.setJarByClass(ProductPromotionInvertedIndex.class);

job.setMapperClass(FactCustomerMapper.class);

job.setCombinerClass(InvertedIndexReducer.class);

job.setReducerClass(InvertedIndexReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

}

}

## Code for Performance Improvement

1. CountryPartitioner

package improvedmultiplejoins;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Partitioner;

public class CountryPartitioner extends Partitioner<Text, SalesProfitTuple> {

@Override

public int getPartition(Text key, SalesProfitTuple value, int numReduceTasks) {

String[] splitval = key.toString().split(",");

// System.out.println(splitval[0] + "hashcode" +splitval[0].hashCode() );

int hash = splitval[0].hashCode();

if(hash<0)

return ((-1\*hash)% numReduceTasks);

else

return (hash % numReduceTasks);

}

}

1. DriverCode with compression enabled and use of Partitioner and Combiner

package improvedmultiplejoins;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

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.input.MultipleInputs;

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;

/\*

\* Performance improved by

\* 1 - Implementing Partitioner

\* 2- Implementing Combiner

\* 3- Compression of file generated and read by mapper and reducer

\* 4 - estimating number of reducer beforehand and taking appropriate number of reducer

\* 5 – sending less data from mapper to reducer

\*/

public class ImprovedMultipleJoins {

public static void main(String[] args) {

try

{

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "AdventureWorks Analysis by customer");

job.setJarByClass(ImprovedMultipleJoins.class);

conf.set("mapreduce.map.output.compress","true");

conf.set("mapreduce.output.fileoutputformat.compress", "true");

MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, FactProdMapper.class);

MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, CustomerMapper.class);

job.getConfiguration().set("join.type", args[2]);

//job.setNumReduceTasks(0);

job.setReducerClass(CustomerReducer.class);

job.setOutputFormatClass(TextOutputFormat.class);

TextOutputFormat.setOutputPath(job, new Path(args[3]));

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.waitForCompletion(true);

/\*Job 2\*/

Job job1 = Job.getInstance(conf, "AdventureWorks Analysis by Country");

job1.setJarByClass(ImprovedMultipleJoins.class);

MultipleInputs.addInputPath(job1, new Path(args[3]), TextInputFormat.class, IntermediateMapper.class);

MultipleInputs.addInputPath(job1, new Path(args[4]), TextInputFormat.class, CountryMapper.class);

job1.getConfiguration().set("join.type",args[2]);

//job.setNumReduceTasks(0);

job1.setReducerClass(CustomerReducer.class);

job1.setOutputFormatClass(TextOutputFormat.class);

TextOutputFormat.setOutputPath(job1, new Path(args[5]));

job1.setOutputKeyClass(Text.class);

job1.setOutputValueClass(Text.class);

job1.waitForCompletion(true);

/\*Job3\*/

Job job2 = Job.getInstance(conf, "Aventure Sales Analysis");

job2.setJarByClass(ImprovedMultipleJoins.class);

job2.setMapperClass(SalesMapper.class);

job2.setCombinerClass(SalesReducer.class);

job2.setPartitionerClass(CountryPartitioner.class);

job2.setReducerClass(SalesReducer.class);

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(SalesProfitTuple.class);

job2.setNumReduceTasks(6);

FileInputFormat.addInputPath(job2, new Path(args[5]));

FileOutputFormat.setOutputPath(job2, new Path(args[6]));

job2.waitForCompletion(true);

/\*job4\*/

Job job3 = Job.getInstance(conf, "Aventure Sales Analysis by binning");

job3.setJarByClass(ImprovedMultipleJoins.class);

job3.setMapperClass(CountryBinMapper.class);

job3.setNumReduceTasks(0);

MultipleOutputs.addNamedOutput(job3, "Country", TextOutputFormat.class, Text.class, NullWritable.class);

MultipleOutputs.setCountersEnabled(job3, true);

FileInputFormat.addInputPath(job3, new Path(args[6]));

FileOutputFormat.setOutputPath(job3, new Path(args[7]));

System.exit(job3.waitForCompletion(true) ? 0 : 1);

}

catch(IOException e)

{

System.out.println("Exception in IO main "+e.getMessage());

}

catch(InterruptedException e)

{

System.out.println("Exception in interrupted main "+e.getMessage());

}

catch(ClassNotFoundException e)

{

System.out.println("Exception in class not found main "+e.getMessage());

}

}

}

1. DriverCode Implementing Cache

package multiplejoinusingcache;

import java.io.File;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

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.input.MultipleInputs;

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 MultipleJoinUsingCache {

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Multi join improved way");

job.setJarByClass(MultipleJoinUsingCache.class);

conf.set("mapreduce.map.output.compress","true");

conf.set("mapreduce.output.fileoutputformat.compress", "true");

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

job.setMapOutputKeyClass(GeographyIdKey.class);

job.setMapOutputValueClass(JoinGenericWritable.class);

job.addCacheFile(new File(args[2]).toURI());

MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, FactCustomerMapper.class);

MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, CountryMapper.class);

job.setReducerClass(JoinReducer.class);

job.setSortComparatorClass(JoinSortingComparator.class);

job.setGroupingComparatorClass(JoinGroupingComparator.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

FileOutputFormat.setOutputPath(job, new Path(args[3]));

job.waitForCompletion(true);

/\*Job2\*/

Job job2 = Job.getInstance(conf, "Aventure Sales Analysis");

job2.setJarByClass(MultipleJoinUsingCache.class);

job2.setMapperClass(SalesMapper.class);

job2.setCombinerClass(SalesReducer.class);

job2.setPartitionerClass(CountryPartitioner.class);

job2.setReducerClass(SalesReducer.class);

job2.setNumReduceTasks(6);

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(SalesProfitTuple.class);

FileInputFormat.addInputPath(job2, new Path(args[3]));

FileOutputFormat.setOutputPath(job2, new Path(args[4]));

job2.waitForCompletion(true);

/\*Job3\*/

Job job3 = Job.getInstance(conf, "Aventure Sales Analysis by binning");

job3.setJarByClass(MultipleJoinUsingCache.class);

job3.setMapperClass(CountryBinMapper.class);

job3.setNumReduceTasks(0);

MultipleOutputs.addNamedOutput(job3, "Country", TextOutputFormat.class, Text.class, NullWritable.class);

MultipleOutputs.setCountersEnabled(job3, true);

FileInputFormat.addInputPath(job3, new Path(args[4]));

FileOutputFormat.setOutputPath(job3, new Path(args[5]));

System.exit(job3.waitForCompletion(true) ? 0 : 1);

}

}

1. UsingCache in One Mapper while joining multiplefiles

package multiplejoinusingcache;

import java.io.BufferedReader;

import java.io.File;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.URI;

import java.util.HashMap;

import java.util.List;

import org.apache.commons.io.FileUtils;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class CountryMapper extends Mapper<LongWritable, Text, GeographyIdKey, JoinGenericWritable> {

private HashMap<Integer, String> geographyKeys = new HashMap<Integer, String>();

private void readProductSubcategoriesFile(URI uri) throws IOException{

List<String> lines = FileUtils.readLines(new File(uri));

for (String line : lines) {

String[] recordFields = line.split(",");

int key = Integer.parseInt(recordFields[0]);

String geographyName = recordFields[5]+","+recordFields[3]+","+recordFields[1];

geographyKeys.put(key, geographyName);

}

}

public void setup(Context context) throws IOException{

URI[] uris = context.getCacheFiles();

readProductSubcategoriesFile(uris[0]);

}

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

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

int customerId = Integer.parseInt(recordFields[0]);

int geographyId = recordFields[1].length() > 0 ? Integer.parseInt(recordFields[1]) : 0;

String geographyName = geographyId > 0 ? geographyKeys.get(geographyId) : "";

String[] geoSplit = geographyName.split(",");

String country = geoSplit[0];

String state = geoSplit[1];

String city = geoSplit[2];

GeographyIdKey recordKey = new GeographyIdKey(customerId, GeographyIdKey.GEOGRAPHY\_RECORD);

GeographyRecord record = new GeographyRecord(country, state, city);

JoinGenericWritable genericRecord = new JoinGenericWritable(record);

context.write(recordKey, genericRecord);

}

}

1. GenericWritable class for joining 2 classes with different output value type

package multiplejoinusingcache;

import org.apache.hadoop.io.GenericWritable;

import org.apache.hadoop.io.Writable;

public class JoinGenericWritable extends GenericWritable {

private static Class<? extends Writable>[] CLASSES = null;

static {

CLASSES = (Class<? extends Writable>[]) new Class[] {

FactSalesRecord.class,

GeographyRecord.class

};

}

public JoinGenericWritable() {}

public JoinGenericWritable(Writable instance) {

set(instance);

}

@Override

protected Class<? extends Writable>[] getTypes() {

return CLASSES;

}

}

1. GroupingComparator

package multiplejoinusingcache;

import org.apache.hadoop.io.WritableComparable;

import org.apache.hadoop.io.WritableComparator;

public class JoinGroupingComparator extends WritableComparator {

public JoinGroupingComparator() {

super (GeographyIdKey.class, true);

}

@Override

public int compare (WritableComparable a, WritableComparable b){

GeographyIdKey first = (GeographyIdKey) a;

GeographyIdKey second = (GeographyIdKey) b;

return first.customerId.compareTo(second.customerId);

}

}

1. SortingComparator

package multiplejoinusingcache;

import org.apache.hadoop.io.WritableComparable;

import org.apache.hadoop.io.WritableComparator;

public class JoinSortingComparator extends WritableComparator {

public JoinSortingComparator()

{

super (GeographyIdKey.class, true);

}

@Override

public int compare (WritableComparable a, WritableComparable b){

GeographyIdKey first = (GeographyIdKey) a;

GeographyIdKey second = (GeographyIdKey) b;

return first.compareTo(second);

}

}

1. Joining Fact and Product File Reducer

package multiplejoinusingcache;

import java.io.IOException;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.Writable;

import org.apache.hadoop.mapreduce.Reducer;

public class JoinReducer extends Reducer<GeographyIdKey, JoinGenericWritable, NullWritable, Text>{

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

StringBuilder output = new StringBuilder();

double salesAmt = 0;

double profit = 0.0;

for (JoinGenericWritable v : values) {

Writable record = v.get();

if (key.recordType.equals(GeographyIdKey.GEOGRAPHY\_RECORD)){

GeographyRecord pRecord = (GeographyRecord)record;

output.append(pRecord.country.toString()).append(", ");

output.append(pRecord.state.toString()).append(", ");

output.append(pRecord.city.toString()).append(", ");

} else {

FactSalesRecord record2 = (FactSalesRecord)record;

salesAmt += Double.parseDouble(record2.salesAmt.toString());

profit += Double.parseDouble(record2.profit.toString());

}

}

context.write(NullWritable.get(), new Text(output.toString() + salesAmt + ", " + profit));

}

}