**BDA Assignment**

|  |  |
| --- | --- |
| Team Members | Name Registration No. |
|  |  |
|  |  |

*Assignment 1:*

1. *Select a public data set (with min. of 5 attributes and 500 tuples – refer data repository links), typically a csv file,*
2. *import to Mongodb, do the necessary data operations using Mongodb query language and export from Mongodb.*
3. *This file shouldbe subsequently stored in HDFS to derive some useful information, in the context of problem domain, using mapreduce in Java.*

*1.* Selected data set: Dow Jones Index Data Set

Abstract: This dataset contains weekly data for the Dow Jones Industrial Index. It has been used in computational investing research.

Number of Instances: 750

Number of Attributes: 16

Attribute Information: Quarter, Stock, Date, Open, High, Low, Close, Volume, Percent\_Change\_Price, Percent\_Change\_Volume\_Over\_Last\_Wk, Previous\_Weeks\_Volume, Next\_Weeks\_Open, Next\_Weeks\_Close, Percent\_Change\_Next\_Weeks\_Price, Days\_To\_Next\_Dividend, Percent\_Return\_Next\_Dividend.

*2.* Exported from MongoDB with new fields:

“quarter", "volumePrice", "closingPrice", "stockAndDate", "dailyAvg", "weekAvg", "twoWeekAvg".

*3.* File is given as input using:

$ hdfs dfs -cat /dowjones\_MongoDB.txt

The program sorts and groups by total volume price.

The command to run:

$ hadoop jar ./test.jar dj.dowjones..GroupVolume /dowjones\_MongoDB.txt /test\_output

To get the output:/dowjones\_MongoDB.txt

$ hdfs dfs -cat /test\_output/part-r-00000

**Explain what fields are present in csv file used to import data to mongodb:**

Quarter: the quarter when the stock was measured.

Stock: the name used to identify the stock.

Date: the date when the stock was measured.

Open: the measured when the stock exchange opened.

High: the highest value during the financial day of the stock.

Low: the lowest value during the financial day of the stock.

Close: the measured when the stock exchange closed.

Volume: the number of stock of the company present in stock exchange.

Percent\_Change\_Price: percentile change in stock.

Percent\_Change\_Volume\_Over\_Last\_Wk: percentile change in stock over last week.

Previous\_Weeks\_Volume: percentile change in stock previous week.

Next\_Weeks\_Open: the measured when the stock exchange opened next week.

Next\_Weeks\_Close: the measured when the stock exchange closed next week.

Percent\_Change\_Next\_Weeks\_Price: percentile change in stock over next week.

**Command used to import data to mongodb:**

mongoimport

This command used at command prompt imports CSV (Comma Separated Values) or TSV (Tab Separated Values) files or JSON (Javascript Object Notation) documents into MongoDB.

***mongoimport -d mydb1 -c djis --type csv --file E:/djindex.csv –headerline***

fields present

-d, --db=<database-name> database to use.

-c, --collection=<collection-name> collection to use.

--type=<type> input format to import: json, csv, or tsv (defaults to 'json').

--headerline use first line in input source as the field list (CSV and TSV only).

--file=<filename> file to import from; if not specified, stdin is used.

mongodb query for data operations:

MongoDB query language

Create, Read, Update, and Delete operation in MongoDB

Create Creation of data is done using insert() or update() or save() method.

Read Reading the data is peformed using the find() method.

Update Update to data is accomplished using the update() method with

UPSERT set to false.

Delete A document is deleted using the remove() method.

**MongoDB query:**

**Map function:**

var map = function(){

var key = this.stock+"-"+this.quarter;

var value = {

date1: this.date,

quarter1: this.quarter,

open1: this.open,

high1: this.high,

low1: this.low,

close1: this.close,

volume1: this.volume,

next\_weeks\_open1: this.next\_weeks\_open,

next\_weeks\_close1: this.next\_weeks\_close

};

emit(key,value);

}

**Reduce function:**

var reduce = function(key,values){

var final\_val = {

quarter: 0,

volumePrice: 0,

closingPrice: 0,

stockAndDate: key,

dailyAvg: 0,

weekAvg: 0,

twoWeekAvg: 0

}

for (var i = 1; i < values.length; i++) {

final\_val.quarter += values[i].quarter1;

final\_val.volumePrice += (values[i].close1 \* values[i].volume1);

final\_val.closingPrice += values[i].close1;

final\_val.stockAndDate += "-"+values[i].date1;

final\_val.dailyAvg += ( values[i].open1 + values[i].close1 ) / 2;

final\_val.weekAvg += ( values[i].next\_weeks\_open1 + values[i].next\_weeks\_close1 ) / 2;

final\_val.twoWeekAvg += ( values[i].open1 + values[i].close1 + values[i].next\_weeks\_open1 + values[i].next\_weeks\_close1 ) / 4;

}

final\_val.quarter = values.length-1;

final\_val.volumePrice /= values.length-1;

final\_val.closingPrice /= values.length-1;

final\_val.dailyAvg /= values.length-1;

final\_val.weekAvg /= values.length-1;

final\_val.twoWeekAvg /= values.length-1;

return final\_val;

}

**MapReduce function call:**

db.djis.mapReduce(

map,

reduce,

{ out: "djis\_filtered" }

)

**Map Reduce stats:**

{

"result" : "djis\_filtered",

"timeMillis" : 502,

"counts" : {

"input" : 2250,

"emit" : 2250,

"reduce" : 194,

"output" : 60

},

"ok" : 1

}

**Command used to export data from MongoDB:**

***mongoexport --db mydb1 --collection djis\_filtered --out e:\djis\_filtered.json***

**Explain what problem is solved using MapReduce:**

In the map function the system gathered all required attributes and assigned them to an array. The key was then generated by concatenating stock name and quarter attributes.

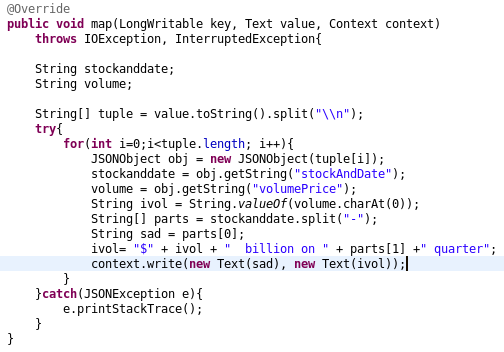
**Definition:**

The MapReduce model simplifies parallel processing by abstracting away the complexities involved in working with distributed systems, such as computational parallelization, work distribution, and dealing with unreliable hardware and software.

With this abstraction, MapReduce allows the programmer to focus on addressing business needs, rather than getting tangled up in distributed system complications.

MapReduce decomposes work submitted by a client into small parallelized map and reduce workers.

**Briefly explain what is done in mapper:**



Separates the name of the stock from the key. which is attached to the quarter. In the following format: [STOCK\_NAME]-[QUARTER].

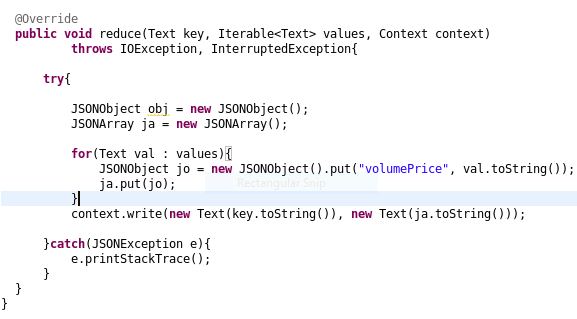
Collects parameters, concatenates currency (dollar $) units and quarter to the volume.

**Definition:**

The mapper splits the line into distinct words, and outputs each word (the key) along with the word's originating filename (the value).

Your reducers need to be able to generate a line for each word in your input, so your map output key should be each word in the input files so that MapReduce can join them all together. The value for each key will be the containing filename, which is your document ID.

**Briefly explain what is done in reducer:**



Reducer receives input as pair of volume and stock name. in which the stock name is the key. and gives final output with stock name and price of total share of the company for both the quarter. Thus reducer performs aggregate operation.

**Definition:**

The reducer collects all the filenames for each key, and outputs a single record, with the key and a comma-separated list of filenames.

The goal of your reducer is to create an output line for each word, and a list of the document ID s in which the word appears. The MapReduce framework will take care of calling your reducer once per unique key outputted by the mappers, along with a list of document ID s. All you need to do in your reducer is combine all the document ID s together and output them once in the reducer.