Hadoop MapReduce Project

Stripe - Pair - Hybrie

Sujan Maddumage Don
Index: 985333
Ms in Computer Science
Maharishi University of Management

10/20/2016

Table of content

Solution	Hadoop map reduce for aggregate the big data		
Description			
Client	*unknown*	**Estimated Effort**	
Author	Sujan Duminda	**Drafted Date**	10/19/2016
Reviewed		**Reviewed Date**	
Version	0.1	**Last Modified Date**	10/19/2016
Status	Draft	**CR/SRS Ref**	

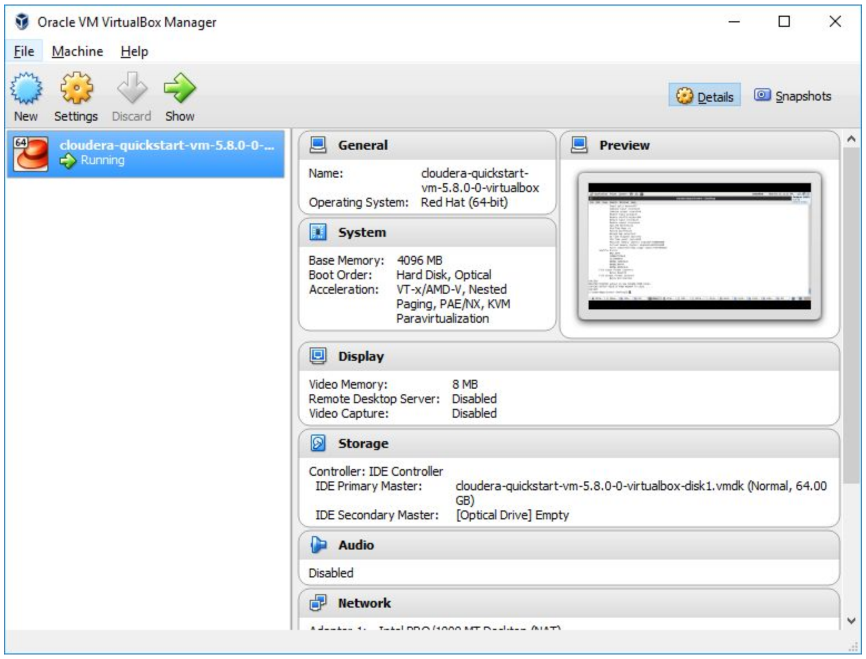
Name	**Date**	**Reason For Changes**	**Ver**
Sujan Duminda	10/19/2016	Initial Version	0.1

Overview

Hadoop MapReduce is a software framework for distributed processing of large data sets on compute clusters of commodity hardware. It is a sub project of the Apache Hadoop project. Also see Apache Hadoop and Hadoop Distributed File System (HDFS). This documentation describes three different mapreduce design pattern Stripe, Pair and hybrid approach and how to do the implementation.

Setup environment

Download cloudera virtual box version from below link
http://www.cloudera.com/downloads/manager/5-8-2.html version 5.8.2
Downloaded and Installed VirtualBox windows version from below link in order to run with linux virtual platform
https://www.virtualbox.org/wiki/Downloads version 5.1.8
Open Oracle virtualbox and drag the cloudera-quickstart-vm-5.8.0-0-virtualbox file into below window.



General
Name: cloudera-quickstart-vm-5.8.0-0-virtualbox
Operating System: Red Hat (64-bit)

System
Base Memory: 4096 MB
Boot Order: Hard Disk, Optical
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization

Display
Video Memory: 8 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage
Controller: IDE Controller
IDE Primary Master: cloudera-quickstart-vm-5.8.0-0-virtualbox-disk1.vmdk (Normal, 64.00 GB)
IDE Secondary Master: [Optical Drive] Empty

Audio
Disabled

Network

Figure 1 oracle virtual box

Run cloudera by clicking show button and virtual cloudera machine will be opened as below.

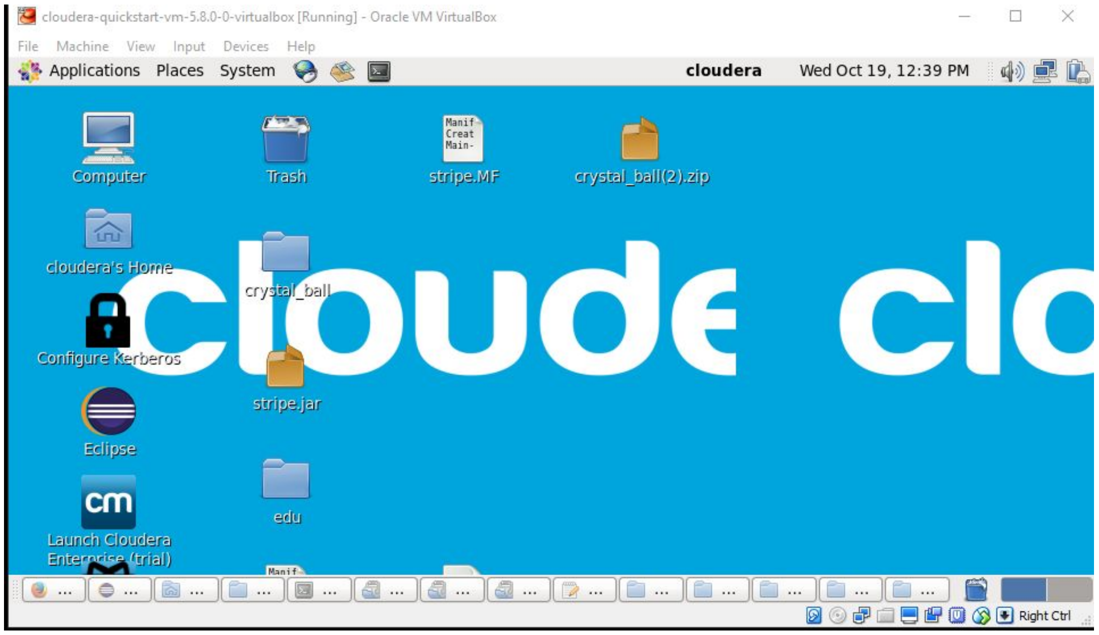

cloudera-quickstart-vm-5.8.0-0-virtualbox [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places System
cloudera
Wed Oct 19, 12:39 PM
Computer
Trash
stripe.MF
crystal_ball(2).zip
cloudera's Home
crystal_ball
Configure Kerberos
stripe.jar
Eclipse
edu
Launch Cloudera Enterprise (trial)
Right Ctrl

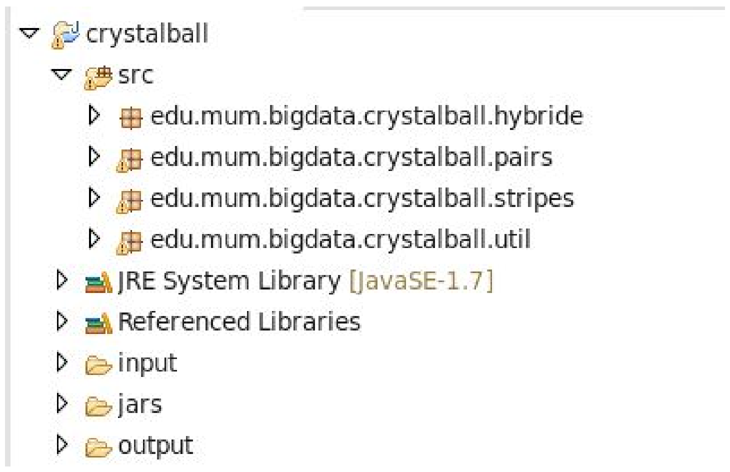
Figure 2 cloudera virtual platform

Create new project

Open eclipse in cloudera and add hadoop libraries to the class path.
Link all jar files from below paths.:

- File system usr/lib/hadoop/
- File system usr/lib/hadoop/lib/
- File system usr/lib/hadoop/client-0.20

Project name : crystalball
Project structure as below :



```
crystalball
  src
    edu.mum.bigdata.crystalball.hybride
    edu.mum.bigdata.crystalball.pairs
    edu.mum.bigdata.crystalball.stripes
    edu.mum.bigdata.crystalball.util
  JRE System Library [JavaSE-1.7]
  Referenced Libraries
  input
  jars
  output
```

Figure 3 package structure of the project

Crystalball project included three mapreduce designs patterns (stripe,pair and hybride) with separate packages. Common object like Pair,Stripemap and Util classes are available on util package.

Problem : calculate relative frequency of each bought product and products bought with the same first product. Eg : customers who bought product id 40 also have bought product 30 and product 38.
Note : neighbourhood of X, N(X) be the set of all term after X and before the next X.

Solutions

Algorithm 1 - Pair

Pair algorithm , implemented with this project can be mentioned as below.
Mapper class algorithm ,used with this project is IN MAPPER COMBINER.

```
Class MAPPER

method INITIALIZE
H<- new AssociativeArray

method Map(docid a, doc d)
    For all term x E doc d do
        For all term y NEIGHBOURS (x) do
            H{pair(x,y)} <- H{pair(x,y)} +1
            H{pair(x,#)} <- H{pair(x,#)} +1
```

```
Method Close()
    For all pair p E H do
        Emit(p, H(p))
```

```
Class REDUCER

method INITIALIZE
marginal =0;

method REDUCE (Pair(w,u) p, counts [c1,c2,c3…])

     If (u== # ) then do
        marginal =0;
         For all c E counts [c1,c2,c3…] do
        marginal = marginal + c;
    Else
      s <- 0
           For all c E counts [c1,c2,c3…]
                   s = s + c
    emit (pair(w,u), s /marginal)
```

Java Implementation

ASSUMPTION : all product ids are integers

Pair object is defined under util package including IntWritable type of key and value. For the serialization purpose , each parameters are writable and readable.
Note : Pair object must be overridden by comparedTo method in order to get the marginal value of each X value of pair. As special charactor used in this is “-1”, map sorting function with sort all pair keys according comparedTo method in Pair object. Therefore we can get first marginal value of each X in the Reducer function.

Mapper output : Pair , IntWritable

Reducer output = Pair , DoubleWritable

Partitioner

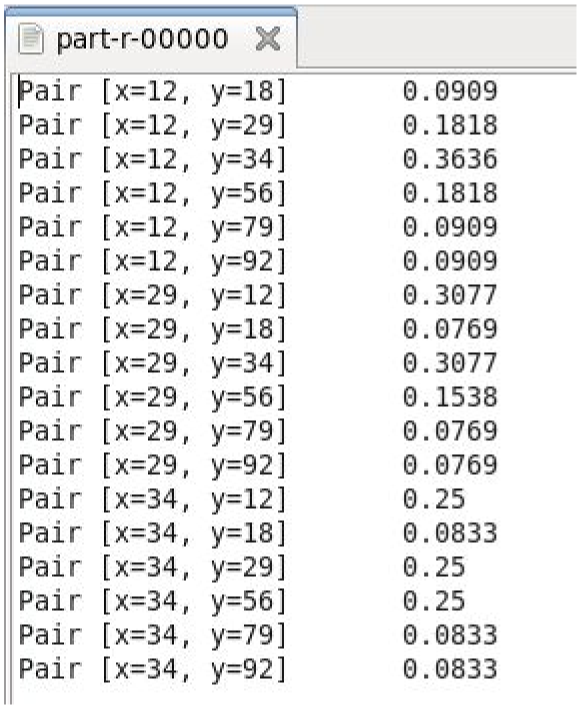
As Pair object has two values , here the partitioner check only first value (X) of the pair and only consider first digit of the product id.

Pair approach has been implemented with 3 partitions as below.

numberOfReducer = 3

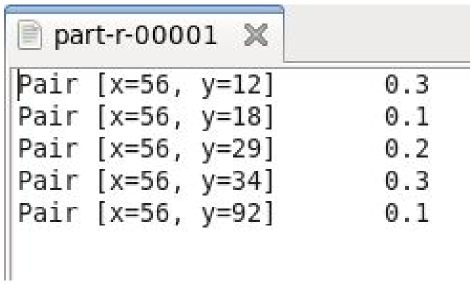
```
If ( numberOfReducer == 0 ) then return 0
Else if (pair.X.charAt(0) < '4' ) return 0
Else if (pair.X.charAt(0) < '7' ) return 1 % numberOfReducer
Else return 2 % numberOfReducer
```

This Partitioner class is named as PairPartitioner.
Sample output result has three files as there are three partitioners as below. These three has been created according to PairPartitioner logic as mentioned above. .



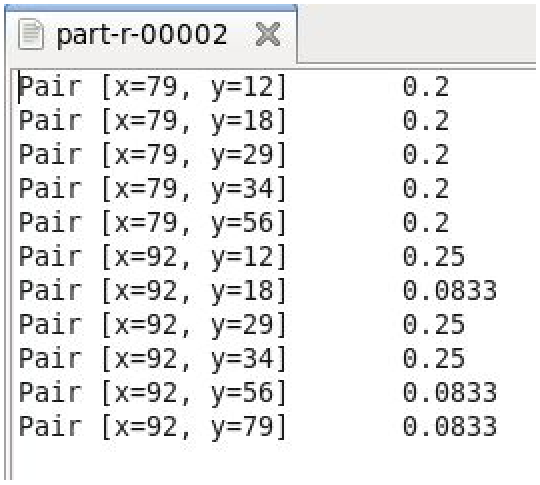
```
Pair [x=12, y=18]	0.0909
Pair [x=12, y=29]	0.1818
Pair [x=12, y=34]	0.3636
Pair [x=12, y=56]	0.1818
Pair [x=12, y=79]	0.0909
Pair [x=12, y=92]	0.0909
Pair [x=29, y=12]	0.3077
Pair [x=29, y=18]	0.0769
Pair [x=29, y=34]	0.3077
Pair [x=29, y=56]	0.1538
Pair [x=29, y=79]	0.0769
Pair [x=29, y=92]	0.0769
Pair [x=34, y=12]	0.25
Pair [x=34, y=18]	0.0833
Pair [x=34, y=29]	0.25
Pair [x=34, y=56]	0.25
Pair [x=34, y=79]	0.0833
Pair [x=34, y=92]	0.0833
```

Figure 4 pair output file 1



```
part-r-00001
Pair [x=56, y=12]       0.3
Pair [x=56, y=18]       0.1
Pair [x=56, y=29]       0.2
Pair [x=56, y=34]       0.3
Pair [x=56, y=92]       0.1
```

Figure 5 pair output file 2

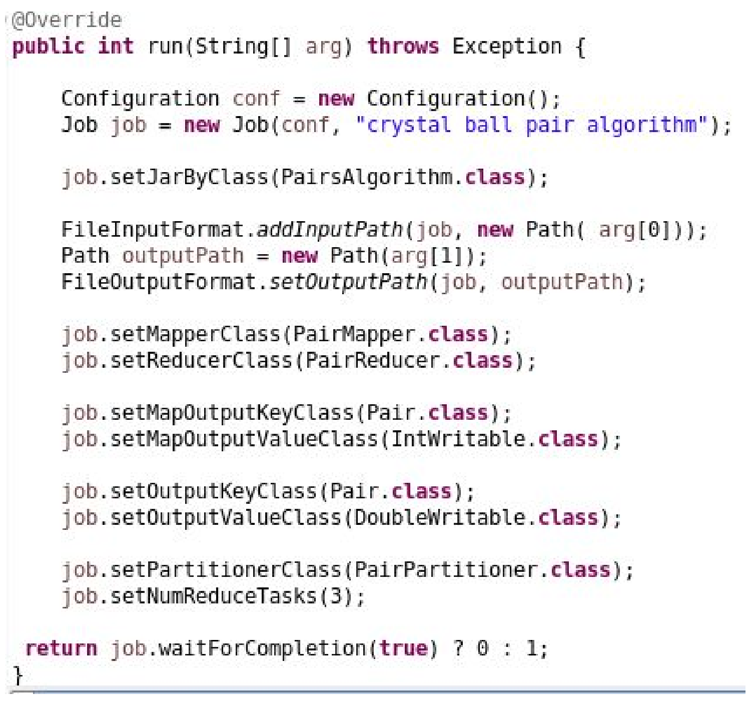


```
part-r-00002
Pair [x=79, y=12]       0.2
Pair [x=79, y=18]       0.2
Pair [x=79, y=29]       0.2
Pair [x=79, y=34]       0.2
Pair [x=79, y=56]       0.2
Pair [x=92, y=12]       0.25
Pair [x=92, y=18]       0.0833
Pair [x=92, y=29]       0.25
Pair [x=92, y=34]       0.25
Pair [x=92, y=56]       0.0833
Pair [x=92, y=79]       0.0833
```

Figure 6 pair output file 3

Job configuration

Each design pattern has main method as that main method call to run method overrriden by hadoop Tool interface. Run method get two arguments mentioned as Run configuration. In the jar file running command , it take first two arguments user enter at terminal.
First argument is Input folder which should create in the hadoop cluster and second argument is output folder which should be created on hadoop cluster. Sample job function can be mentioned as below.



```java
@Override
public int run(String[] arg) throws Exception {

    Configuration conf = new Configuration();
    Job job = new Job(conf, "crystal ball pair algorithm");

    job.setJarByClass(PairsAlgorithm.class);

    FileInputFormat.addInputPath(job, new Path( arg[0]));
    Path outputPath = new Path(arg[1]);
    FileOutputFormat.setOutputPath(job, outputPath);

    job.setMapperClass(PairMapper.class);
    job.setReducerClass(PairReducer.class);

    job.setMapOutputKeyClass(Pair.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setOutputKeyClass(Pair.class);
    job.setOutputValueClass(DoubleWritable.class);

    job.setPartitionerClass(PairPartitioner.class);
    job.setNumReduceTasks(3);

 return job.waitForCompletion(true) ? 0 : 1;
}
```

Figure 7 sample job configuration function - pair

Algorithm 2 - Stripe

Stripe approach has been implemented by IN MAPPER COMBINER mapper class and reducer class as below algorithm

```
Class MAPPER

method INITIALIZE
H<- new AssociativeArray

method Map(docid a, doc d)
    For all term x E doc d do
    If ( H{x} == null) then do
    H{x} <- new AssociativeArray
     NEIGHBOUR :
        For all term y NEIGHBOURS (x) do
            H{x)} <- (H{x)) {u}+1

Method Close()
    For all Term key E H do
        Emit(Term key, H(key))
```

```
Class REDUCER
method REDUCE (Term w, strips [H1,H2,H3…])
Hf = new AssociativeArray();
marginal =0;

    For all H E strips [H1,H2,H3…]do
        For all Itinerary t E H do                  // each itinerary of H looping
            Hf{t.key} = Hf{t.key}+ t.value     // add values of t with same key
            marginal = marginal + t.value    // all values of t of each H is the
        marginal

            For all key k E Hf do
            Hf{k} = Hf{k} /marginal

emit(term w,strip Hf)
```

Java Implementation .

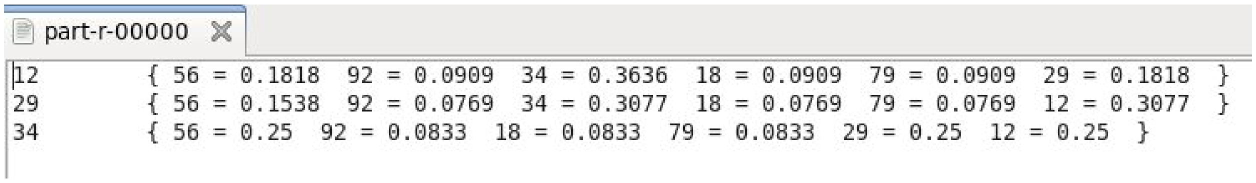
StripeMap is the mapWritable object which has used for mapper output value. This object has two writable values. Both Mapper and Reducer output value are StripeMap object but different key values.
Mapper output = Text , StripeMap(IntWritable,IntWritable)
Reducer output = Text , StripeMap(IntWritable,DoubleWritable)

Partitioner

There are two partitions for stripe.

numberOfReducer = 2
If (numberOfReducer == 0) then return 0
Else if (pair.X.charAt(0) < ‘4’) return 0
Else return 1 % numberOfReducer

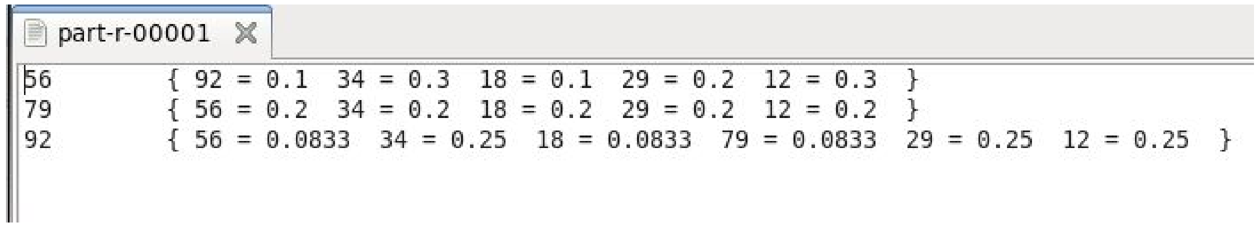
Sample output results as below. As there is two partitions output also have two files.



```
part-r-00000

12	{ 56 = 0.1818  92 = 0.0909  34 = 0.3636  18 = 0.0909  79 = 0.0909  29 = 0.1818  }
29	{ 56 = 0.1538  92 = 0.0769  34 = 0.3077  18 = 0.0769  79 = 0.0769  12 = 0.3077  }
34	{ 56 = 0.25  92 = 0.0833  18 = 0.0833  79 = 0.0833  29 = 0.25  12 = 0.25  }
```

Figure 8 stripe output file 1



```
56        { 92 = 0.1  34 = 0.3  18 = 0.1  29 = 0.2  12 = 0.3  }
79        { 56 = 0.2  34 = 0.2  18 = 0.2  29 = 0.2  12 = 0.2  }
92        { 56 = 0.0833  34 = 0.25  18 = 0.0833  79 = 0.0833  29 = 0.25  12 = 0.25  }
```

Figure 9 stripe output file 2

Algorithm 3 - Hybrid

Hybrid approach has been implemented with IN MAPPER COMBINER with Pairs and Stripe Reducer.
This Mapper class is same as Pair approche mapper algorithm except special character adding part. This approach does not require to count margin from each mapper level since reducer reponsible for that task.

Algorithm;

```
Class MAPPER

method INITIALIZE
H<- new AssociativeArray

method Map(docid a, doc d)
    For all term x E doc d do
        For all term y NEIGHBOURS (x) do
            H{pair(x,y)} <- H{pair(x,y)} +1


Method Close()
    For all pair(x,y) E H do
        Emit(pair(x,y), H(pair(x,y)
Class REDUCER

method INITIALIZER
        currentTerm <- 0;
        H <- new AssociativeArray();
        marginal <- 0;


method REDUCE (Pair(w,u) p, counts [c1,c2,c3…])

If currebtTerm==null do
                currentTerm = w;
Else if currentTerm != w do
                For all term u in H do
                H{u} <- H{u} / marginal
         emit (Term w, Strip H)

         H <- new AssociativeArraay()
        marginal <- 0
        currentTerm <- pair.w

For all c in counts [c1,c2,c3…] do
H{u} <- H{u} + c
marginal = marginal + c


method CLOSE()
        For all term u in H do
        H{u} <- H{u} / marginal
emit (Term w, Strip H)
```

Java implementation

Map output = Pair , IntWritable
Reduce output = Text , StripeMap(Text, DoubleWritable)

Here key of StripeMap is Text and vealue is DoubleWritable.
Partitioner
This has three partitions as below

If (numberOfReducer == 0) then return 0
Else if (pair.X.charAt(0) < '3') return 0
Else if (pair.X.charAt(0) < '7') return 1 % numberOfReducer
Else return 2 % numberOfReducer

As three partitioners are setup on hybrid approach there are only three output file as below.

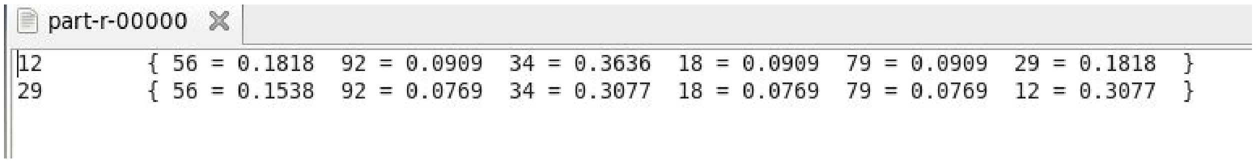

```
part-r-00000
12        { 56 = 0.1818  92 = 0.0909  34 = 0.3636  18 = 0.0909  79 = 0.0909  29 = 0.1818  }
29        { 56 = 0.1538  92 = 0.0769  34 = 0.3077  18 = 0.0769  79 = 0.0769  12 = 0.3077  }
```

Figure 10 hybrid output file 1

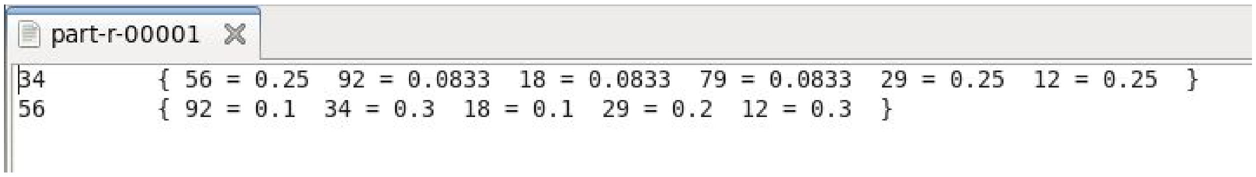

```
part-r-00001
34        { 56 = 0.25  92 = 0.0833  18 = 0.0833  79 = 0.0833  29 = 0.25  12 = 0.25  }
56        { 92 = 0.1  34 = 0.3  18 = 0.1  29 = 0.2  12 = 0.3  }
```

Figure 11 hybrid output file 2

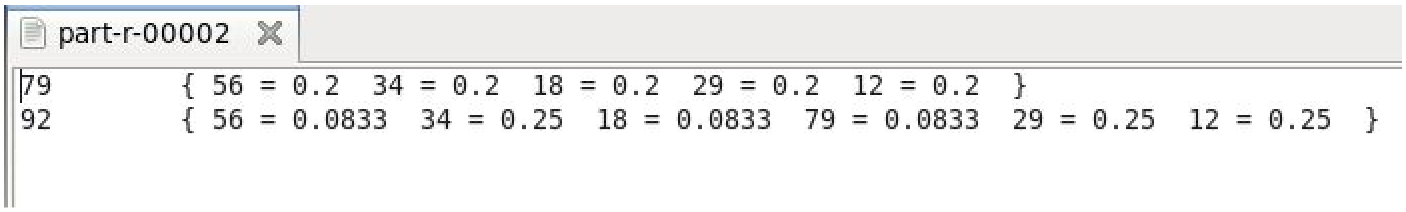

```
part-r-00002
79        { 56 = 0.2  34 = 0.2  18 = 0.2  29 = 0.2  12 = 0.2  }
92        { 56 = 0.0833  34 = 0.25  18 = 0.0833  79 = 0.0833  29 = 0.25  12 = 0.25  }
```

Figure 11 hybrid output file 3

Run the project on Hadoop cluster

In order to run this project on hadoop cluster , I have created a batch file to auto run the tasks as below.
Note : These information also available on Readme file in the Project folder.
Following steps explains how to use this batch file.

Batch file name = crystalball.sh

put Project.zip at any location
unzip the zip file
go inside   the Project folder using Terminal
Eg : [Cloudera@quickstart Project] $
run crystalball.sh file as below;

sh ./crystalball.sh <arg1> <arg2> <arg3>
example :   sh ./crystalball.sh **input output_hybrid hybrid**

above three arguments are as below;

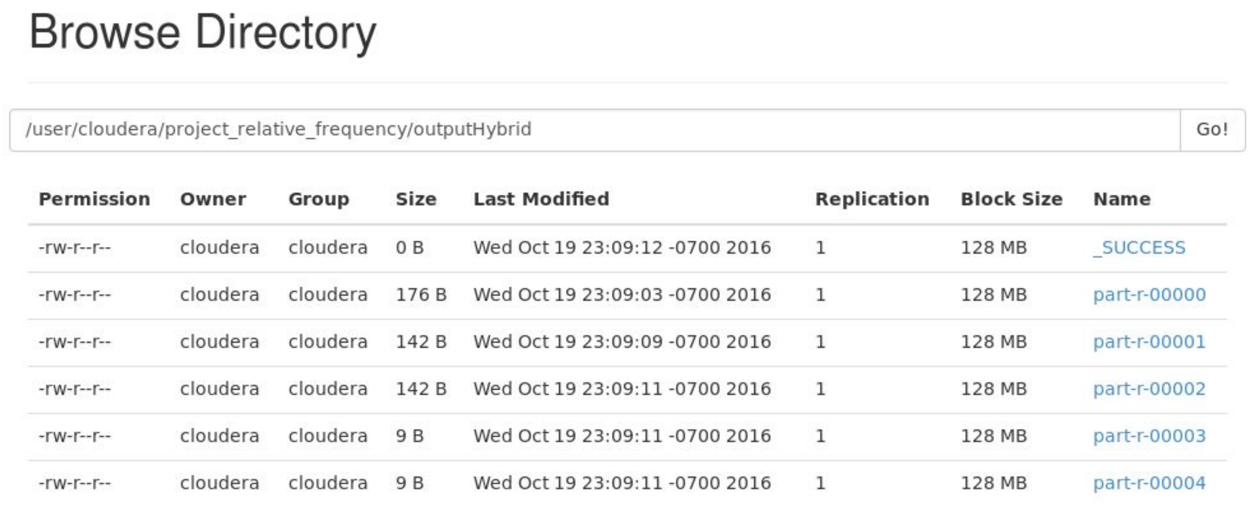
arg1 = hadoop input folder name (any folder name) eg : **input**
arg2 = hadoop output folder name (any folder name) eg : **output_pair**
arg3 = job name. For arg3 user has to add one of below three values.
Possible values (**stripe** , **pair** , **hybrid**).
eg : **stripe**

Note : When run this command terminal is should be in the project folder and project folder should have both batch file and crystalball folder.

Above batch file will be run as following order.

1. remove old jars in local folder created by user
2. create jar file according arg3 (job name).
3. remove project folder from hadoop
4. create new project folder from hadoop according to arg1 (input folder)
5. copy input file into hadoop input folder as arg1 (input folder).
6. run jar file entered as arg3. (stripe or pair or hybrid). here input and output folder path set as arguments.
7. remove output folder from local.
8. copy hadoop output results folder into local output folder.

After completed the job , Hadoop file system will be updated. Below shows sample output results screenshot.



Browse Directory

/user/cloudera/project_relative_frequency/outputHybrid

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	cloudera	cloudera	0 B	Wed Oct 19 23:09:12 -0700 2016	1	128 MB	_SUCCESS
-rw-r--r--	cloudera	cloudera	176 B	Wed Oct 19 23:09:03 -0700 2016	1	128 MB	part-r-00000
-rw-r--r--	cloudera	cloudera	142 B	Wed Oct 19 23:09:09 -0700 2016	1	128 MB	part-r-00001
-rw-r--r--	cloudera	cloudera	142 B	Wed Oct 19 23:09:11 -0700 2016	1	128 MB	part-r-00002
-rw-r--r--	cloudera	cloudera	9 B	Wed Oct 19 23:09:11 -0700 2016	1	128 MB	part-r-00003
-rw-r--r--	cloudera	cloudera	9 B	Wed Oct 19 23:09:11 -0700 2016	1	128 MB	part-r-00004

Figure 12 hadoop output result file structure - hybrid

Technical Details

Required technology

Linux platform cloudera 5.8.2
Oracle virtual box 5.1.8
Java 7
Eclipse platform
Hadoop Mapreduce libraries

Review Comments