**Instructions**

In this project you will create a crystal ball to predict events that may happen once a certain event happened.

Example: Amazon will say people who bought “item one” have bought the following items : “item two”, “item three”, “item four”.

For the purpose of this project you can assume that historical customer data is available in the following form.

34 56 29 12 34 56 92 10 34 12      // items bought by a customer, listed in the order she bought it

18 29 12 34 79 18 56 12 34 92  // items bought by another customer, listed in the order she bought it

…

**Let the neighborhood of X, N(X) be set of all term after X and before the next X.**

**Example: Let Data block be [a b c a d e]**

**N(a) = {b, c}, N(b) = {c, a, d, e}, N(c) = {a, d, e}, N(a) ={d, e}, N(d) = {e}, N(e) = {}.**

**Part 1. [5 point]  Set up a single node cluster and optionally an eclipse development environment to create and test your programs.**

            Document all the steps and submit.

**Part 2. Implement Pairs algorithm to compute relative frequencies.**

1. [2 points] Create Java classes (.java files)
2. [1 points] Show input, output and batch file to execute your program at command line in Hadoop.

**Part 3. Implement Stripes algorithm to compute relative frequencies.**

1. [2 points] Create Java classes (.java files)
2. [1 points] Show input, output and batch file to execute your program at command line in Hadoop.

**Part 4. Implement Pairs in Mapper and Stripes in Reducer to compute relative frequencies.**

1. [2 points] Create Java classes (.java files)
2. [1 points] Show input, output and batch file to execute your program at command line in Hadoop.

YOU MUST SUBMIT ALL THE WORK  BY 10:00 AM, FRIDAY, SEPT. 18, 2015

**Using Hadoop to Solve Crystal Ball Problem**

**Pairs aproach Mapper class**

public class Mapper1 extends Mapper<LongWritable, Text, Pair, IntWritable> {  
  
 Map<Pair, Integer> map = new HashMap<>();  
  
 @Override  
 public void map(LongWritable key, Text value, Context context)  
 throws IOException, InterruptedException {  
  
 String[] items = value.toString().split(" ");  
 for (int i = 0; i < items.length; i++) {  
 // System.out.println(getN(i, items));  
 for (String neighbor : getN(i, items)) {  
 if (neighbor.equals("")) {  
 continue;  
 }  
 Pair pair = new Pair(items[i], neighbor);  
 Pair p = new Pair(items[i], "\*");  
 context.write(pair, new IntWritable(1));  
 context.write(p, new IntWritable(1));  
 }  
 }  
 }  
  
 public List<String> getN(int idx, String[] items) {  
  
 List<String> list = new ArrayList<>();  
 for (int j = idx + 1; j < items.length; j++) {  
 if (items[idx].equals(items[j])) {  
 return list;  
 } else {  
 list.add(items[j]);  
 }  
 }  
 return list;  
 }  
}

**Pairs aproach Pair class**

class Pair implements WritableComparable<Pair> {  
  
 private Text host = new Text();  
 private Text neighbor = new Text();  
  
 public Pair() {  
 }  
  
 public Pair(String host, String neighbor) {  
 this.host = new Text(host);  
 this.neighbor = new Text(neighbor);  
 }  
  
 public String getHost() {  
 return host.toString();  
 }  
  
 public void setHost(String host) {  
 this.host = new Text(host);  
 }  
  
 public String getNeighbor() {  
 return neighbor.toString();  
 }  
  
 public void setNeighbor(String neighbor) {  
 this.neighbor = new Text(neighbor);  
 }  
  
 @Override  
 public void readFields(DataInput in) throws IOException {  
 host.readFields(in);  
 neighbor.readFields(in);  
 }  
  
 @Override  
 public void write(DataOutput out) throws IOException {  
 host.write(out);  
 neighbor.write(out);  
 }  
  
 @Override  
 public int compareTo(Pair pair) {  
 int tmp = this.getHost().compareTo(pair.getHost());  
 return tmp == 0 ? this.getNeighbor().compareTo(pair.getNeighbor())  
 : tmp;  
 }  
  
 @Override  
 public String toString() {  
 return "(" + host + ", " + neighbor + ")";  
 }  
  
 @Override  
 public int hashCode() {  
 return host.hashCode() \* 163 + neighbor.hashCode();  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (o instanceof Pair) {  
 Pair tp = (Pair) o;  
 return host.equals(tp.host) && neighbor.equals(tp.neighbor);  
 }  
 return false;  
 }  
  
}

**Pairs aproach Reducer class**

public class Reducer1 extends Reducer<Pair, IntWritable, Text, Text> {  
 Map<String, Integer> map = new HashMap<>();  
 DecimalFormat decimalFormat = new DecimalFormat("0.00");  
  
 @Override  
 public void reduce(Pair key, Iterable<IntWritable> values, Context context)  
 throws IOException, InterruptedException {  
 int count = 0;  
  
 if (key.getNeighbor().equals("\*")) {  
 for (IntWritable v : values) {  
 count += v.get();  
 }  
 map.put(key.getHost(), count);  
 } else {  
 for (IntWritable v : values) {  
 count += v.get();  
 }  
 context.write(  
 new Text(key.toString()),  
 new Text(decimalFormat.format(count  
 / new Double(map.get(key.getHost())))));  
 }  
 }  
  
}

**Pairs aproach output**

**(10, 12) 0.50**

**(10, 34) 0.50**

**(12, 10) 0.09**

**(12, 18) 0.09**

**(12, 34) 0.36**

**(12, 56) 0.18**

**(12, 79) 0.09**

**(12, 92) 0.18**

**(18, 12) 0.25**

**(18, 29) 0.12**

**(18, 34) 0.25**

**(18, 56) 0.12**

**(18, 79) 0.12**

**(18, 92) 0.12**

**(29, 10) 0.07**

**(29, 12) 0.27**

**(29, 18) 0.07**

**(29, 34) 0.27**

**(29, 56) 0.13**

**(29, 79) 0.07**

**(29, 92) 0.13**

**(34, 10) 0.08**

**(34, 12) 0.25**

**(34, 18) 0.08**

**(34, 29) 0.08**

**(34, 56) 0.25**

**(34, 79) 0.08**

**(34, 92) 0.17**

**(56, 10) 0.10**

**(56, 12) 0.30**

**(56, 29) 0.10**

**(56, 34) 0.30**

**(56, 92) 0.20**

**(79, 12) 0.20**

**(79, 18) 0.20**

**(79, 34) 0.20**

**(79, 56) 0.20**

**(79, 92) 0.20**

**(92, 10) 0.33**

**(92, 12) 0.33**

**(92, 34) 0.33**

**Stripes aproach Mapper class**

public class Mapper2 extends Mapper<LongWritable, Text, Text, MapWritable> {  
  
  
 @Override  
 public void map(LongWritable key, Text value, Context context)  
 throws IOException, InterruptedException {  
  
 String[] items = value.toString().split(" ");  
 for (int i = 0; i < items.length; i++) {  
 for (String neighbor : getN(i, items)) {  
 if (neighbor.equals("")) {  
 continue;  
 }  
   
 MapWritable mapw = new MapWritable();  
 mapw.put(new Text(neighbor), new IntWritable(1));  
 context.write(new Text(items[i]), mapw);  
 }  
 }  
 }  
  
 public List<String> getN(int idx, String[] items) {  
  
 List<String> list = new ArrayList<>();  
 for (int j = idx + 1; j < items.length; j++) {  
 if (items[idx].equals(items[j])) {  
 return list;  
 } else {  
 list.add(items[j]);  
 }  
 }  
 return list;  
 }  
}

**Stripes aproach Reducer class**

public class Reducer2 extends Reducer<Text, MapWritable, Text, Text> {  
  
 Map<String, Integer> map = new HashMap<>();  
 DecimalFormat decimalFormat = new DecimalFormat("0.00");  
  
 @Override  
 public void reduce(Text key, Iterable<MapWritable> values, Context context)  
 throws IOException, InterruptedException {  
  
 Map<String, Integer> mapvs = new HashMap<>();  
 for (MapWritable v : values) {  
 for (Entry<Writable, Writable> e : v.entrySet()) {  
  
 String nKey = e.getKey().toString();  
  
 if (mapvs.containsKey(nKey)) {  
 mapvs.put(nKey, mapvs.get(nKey) + 1);  
 } else {  
 mapvs.put(nKey, 1);  
 }  
  
 String hKey = key.toString();  
  
 if (map.containsKey(hKey)) {  
 map.put(hKey, map.get(hKey) + 1);  
 } else {  
 map.put(hKey, 1);  
 }  
 }  
 }  
 StringBuilder sb = new StringBuilder("[");  
 for (Entry<String, Integer> entry : mapvs.entrySet()) {  
 String frequency = decimalFormat.format(entry.getValue()  
 / new Double(map.get(key.toString())));  
 sb.append(" (" + entry.getKey() + ", " + frequency + ") ");  
 }  
 sb.append("]");  
 context.write(new Text(key.toString()), new Text(sb.toString()));  
 }  
}

**Stripes aproach output:**

**10 [ (34, 0.50) (12, 0.50) ]**

**12 [ (79, 0.09) (10, 0.09) (92, 0.18) (56, 0.18) (18, 0.09) (34, 0.36) ]**

**18 [ (79, 0.12) (92, 0.12) (56, 0.12) (34, 0.25) (29, 0.12) (12, 0.25) ]**

**29 [ (79, 0.07) (10, 0.07) (56, 0.13) (92, 0.13) (18, 0.07) (34, 0.27) (12, 0.27) ]**

**34 [ (79, 0.08) (10, 0.08) (56, 0.25) (92, 0.17) (18, 0.08) (29, 0.08) (12, 0.25) ]**

**56 [ (10, 0.10) (92, 0.20) (34, 0.30) (29, 0.10) (12, 0.30) ]**

**79 [ (92, 0.20) (56, 0.20) (18, 0.20) (34, 0.20) (12, 0.20) ]**

**92 [ (10, 0.33) (34, 0.33) (12, 0.33) ]**

**Hybrid aproach Mapper class**

public class Mapper3 extends Mapper<LongWritable, Text, Text, Pair> {  
  
 Map<Pair, Integer> map = new HashMap<>();  
  
 @Override  
 public void map(LongWritable key, Text value, Context context)  
 throws IOException, InterruptedException {  
  
 String[] items = value.toString().split(" ");  
 for (int i = 0; i < items.length; i++) {  
 for (String neighbor : getN(i, items)) {  
 if (neighbor.equals("")) {  
 continue;  
 }  
 Pair pair = new Pair(neighbor, 1);  
 context.write(new Text(items[i]), pair);  
 }  
 }  
 }  
  
 public List<String> getN(int idx, String[] items) {  
  
 List<String> list = new ArrayList<>();  
 for (int j = idx + 1; j < items.length; j++) {  
 if (items[idx].equals(items[j])) {  
 return list;  
 } else {  
 list.add(items[j]);  
 }  
 }  
 return list;  
 }  
}

**Hybrid aproach Pair class**

class Pair implements WritableComparable<Pair> {  
  
 private Text neighbor = new Text();  
 private IntWritable count = new IntWritable();  
  
 public Pair() {  
 }  
  
 public Pair(String neighbor, int count) {  
 this.neighbor = new Text(neighbor);  
 this.count = new IntWritable(count);  
 }  
  
 public String getNeighbor() {  
 return neighbor.toString();  
 }  
  
 public void setNeighbor(String host) {  
 this.neighbor = new Text(host);  
 }  
  
 public String getCount() {  
 return count.toString();  
 }  
  
 public void setCount(int count) {  
 this.count = new IntWritable(count);  
 }  
  
 @Override  
 public void readFields(DataInput in) throws IOException {  
 neighbor.readFields(in);  
 count.readFields(in);  
 }  
  
 @Override  
 public void write(DataOutput out) throws IOException {  
 neighbor.write(out);  
 count.write(out);  
 }  
  
 @Override  
 public int compareTo(Pair pair) {  
 int tmp = this.getNeighbor().compareTo(pair.getNeighbor());  
 return tmp == 0 ? this.getCount().compareTo(pair.getCount())  
 : tmp;  
 }  
  
 @Override  
 public String toString() {  
 return "(" + neighbor + ", " + count + ")";  
 }  
  
 @Override  
 public int hashCode() {  
 return neighbor.hashCode() \* 163 + count.hashCode();  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (o instanceof Pair) {  
 Pair tp = (Pair) o;  
 return neighbor.equals(tp.neighbor) && count.equals(tp.count);  
 }  
 return false;  
 }  
  
}

**Hybrid aproach Reducer class**

public class Reducer3 extends Reducer<Text, Pair, Text, Text> {  
 Map<String, Integer> map = new HashMap<>();  
 DecimalFormat decimalFormat = new DecimalFormat("0.00");  
  
 @Override  
 public void reduce(Text key, Iterable<Pair> values, Context context)  
 throws IOException, InterruptedException {  
   
 int sum = 0;  
 for (Pair pair: values) {  
 String pKey = pair.getNeighbor();  
 if(map.containsKey(pKey)){  
 map.put(pKey, map.get(pKey) + 1);  
 }else{  
 map.put(pKey, new Integer(pair.getCount()));  
 }  
 sum += 1;  
 }  
   
 StringBuilder sb = new StringBuilder("[");  
 for(Entry<String, Integer> entry: map.entrySet()){  
 String frequency = decimalFormat.format(entry.getValue()/new Double(sum));  
 sb.append(" (" + entry.getKey() + ", " + frequency + ") ");  
 }  
 sb.append("]");  
 context.write(new Text(key.toString()), new Text(sb.toString()));  
 map.clear();  
 }  
}

**Hybrid aproach output**

**10 [ (34, 0.50) (12, 0.50) ]**

**12 [ (79, 0.09) (10, 0.09) (92, 0.18) (56, 0.18) (18, 0.09) (34, 0.36) ]**

**18 [ (79, 0.12) (92, 0.12) (56, 0.12) (34, 0.25) (29, 0.12) (12, 0.25) ]**

**29 [ (79, 0.07) (10, 0.07) (56, 0.13) (92, 0.13) (18, 0.07) (34, 0.27) (12, 0.27) ]**

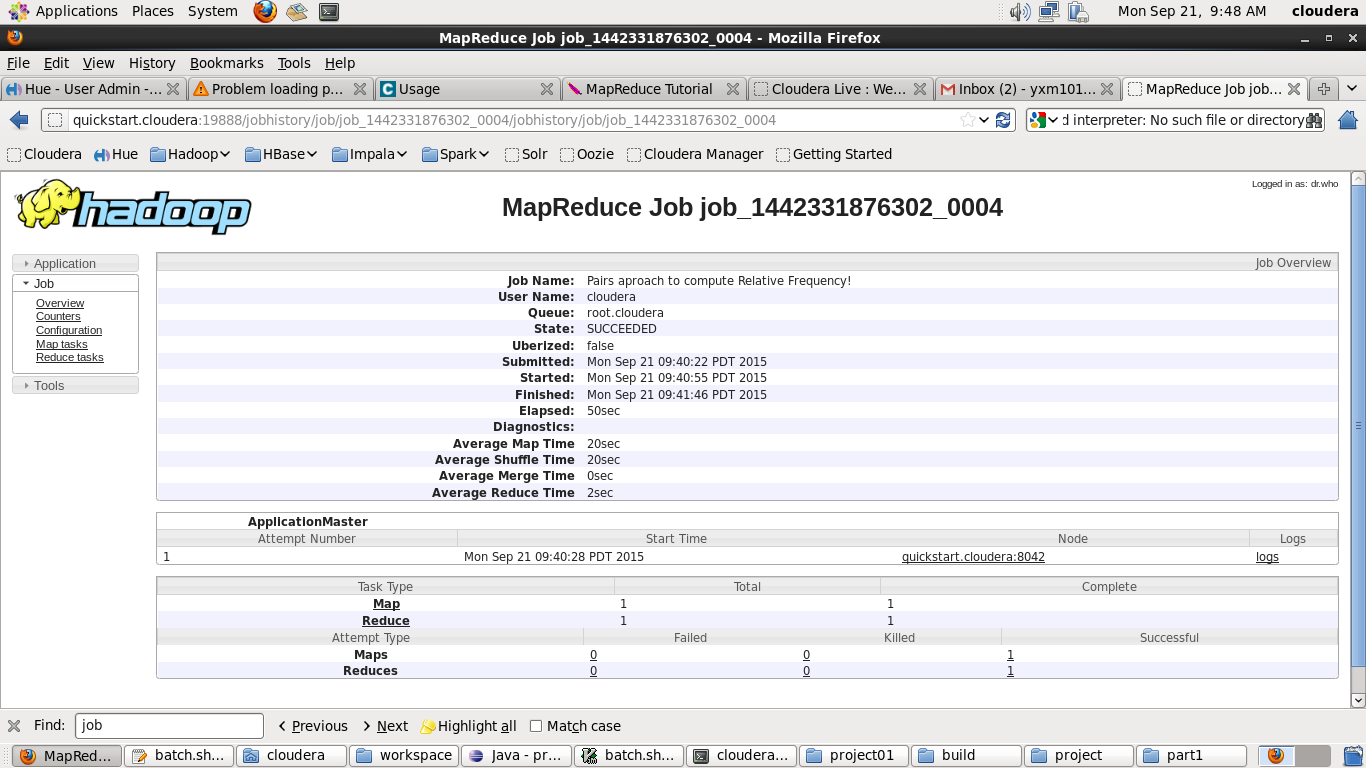
**34 [ (79, 0.08) (10, 0.08) (56, 0.25) (92, 0.17) (18, 0.08) (29, 0.08) (12, 0.25) ]**

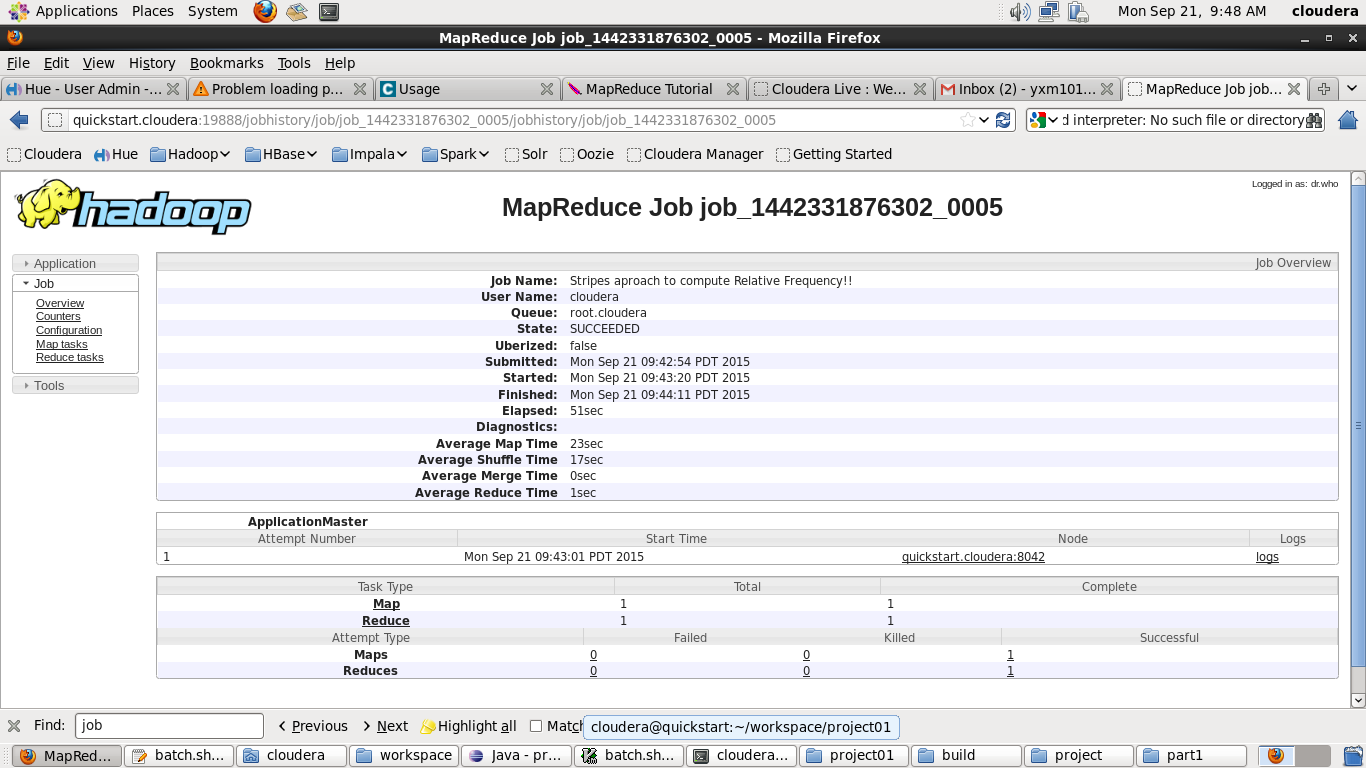
**56 [ (10, 0.10) (92, 0.20) (34, 0.30) (29, 0.10) (12, 0.30) ]**

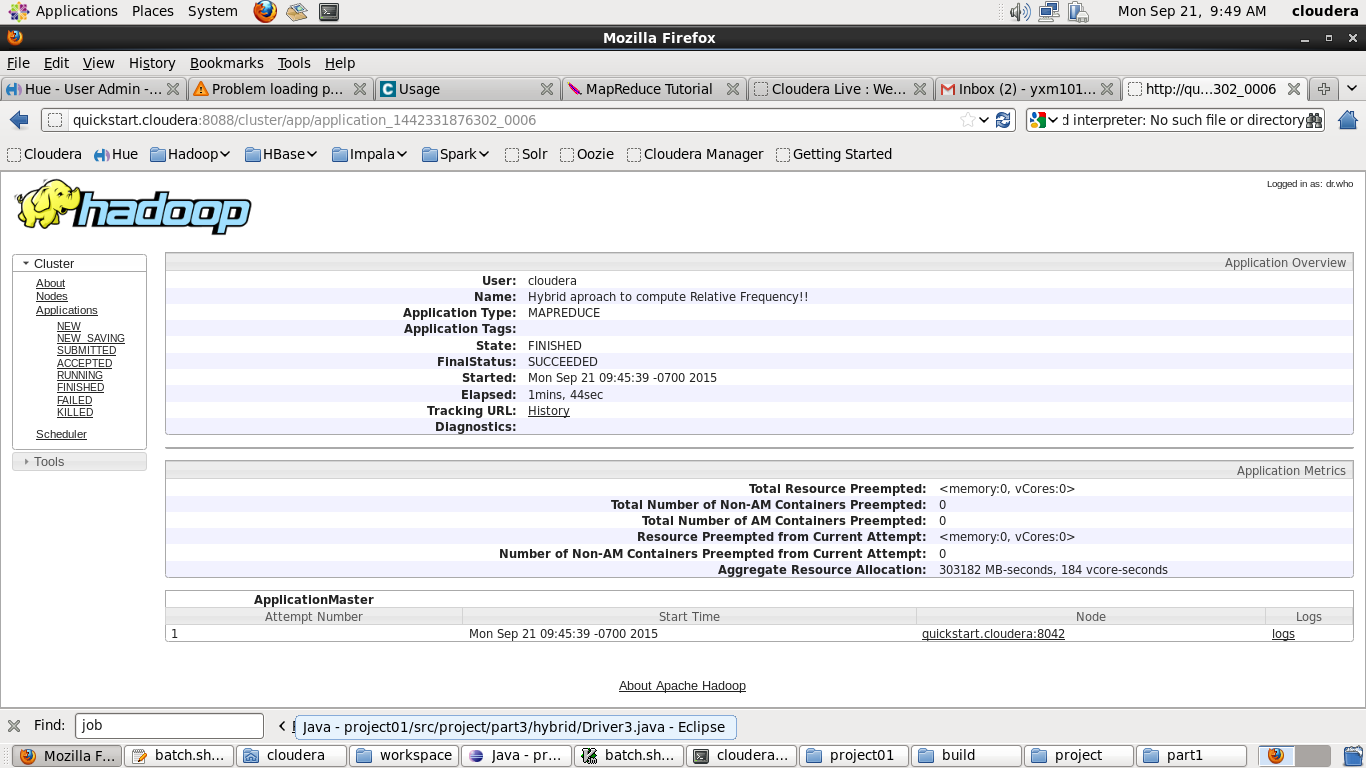
**79 [ (92, 0.20) (56, 0.20) (18, 0.20) (34, 0.20) (12, 0.20) ]**

**92 [ (10, 0.33) (34, 0.33) (12, 0.33) ]**

**Three aproaches comparison**







**Spark Application Example**

Find the top 3 words occurence in a file

**IDE: Intellij + Scala Plugin**

**1. Configuration**

spark-env.sh

#HADOOP\_CONF\_DIR=/opt/data02/hadoop-2.6.0-cdh5.4.0/etc/hadoop

JAVA\_HOME=/opt/modules/jdk1.7.0\_67

SCALA\_HOME=/opt/modules/scala-2.10.4

#######################################################

SPARK\_MASTER\_IP=quickstart.cloudera

SPARK\_MASTER\_PORT=7077

SPARK\_MASTER\_WEBUI\_PORT=8080

SPARK\_WORKER\_CORES=1

SPARK\_WORKER\_MEMORY=1000m

SPARK\_WORKER\_PORT=7078

SPARK\_WORKER\_WEBUI\_PORT=8081

SPARK\_WORKER\_INSTANCES=1

slaves

quickstart.cloudera

spark-defaults.conf

spark.master spark://quickstart.cloudera:7077

Start Spark

Start Master

sbin/start-master.sh

Start Slaves

sbin/start-slaves.sh

WEB UI

http://quickstart.cloudera:8080

**2. Prepare the Data**

touch input.txt

cat bat pat rat

bat cat cat rat

pat bat pat cat

mat cat pat pat

hdfs dfs -mkdir -p /user/hadoop/spark/wordCount

bin/hdfs dfs -put input.txt /user/hadoop/spark/wordCount

bin/hdfs dfs -text /user/hadoop/spark/wordCount/input.txt

**3. Program debugging**

scala> val rdd01 = sc.textFile("hdfs://quickstart.cloudera:8020/user/spark/data/input.txt")

scala> val rdd02 = rdd01.flatMap(\_.split(" ")).map(x=>(x,1)).reduceByKey(\_+\_)

scala> rdd02.collect

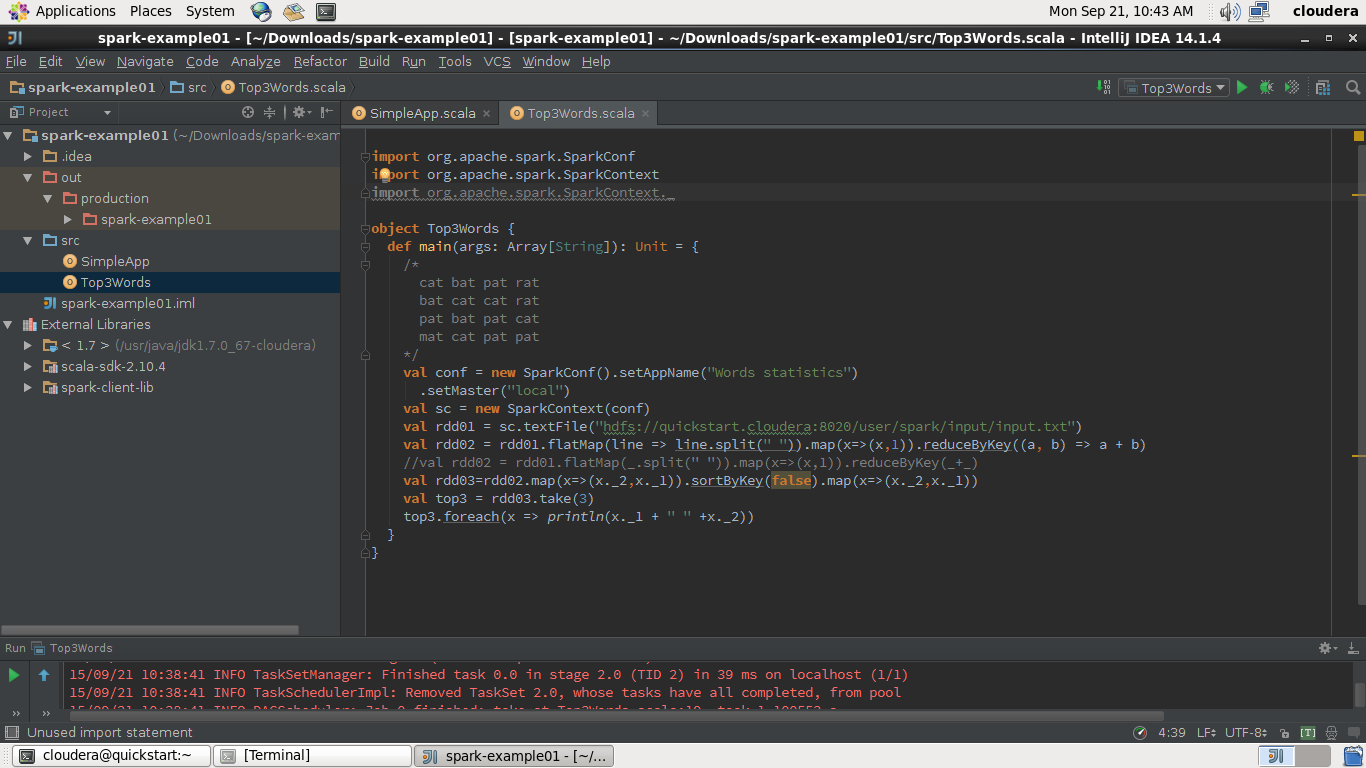
scala> val rdd03=rdd02.map(x=>(x.\_2,x.\_1)).sortByKey(false).map(x=>(x.\_2,x.\_1))

scala> rdd03.take(3)

scala> radd03.collect

scala> wordsort.take(3)

**4. Compose Scala Program**



**5. Package Application to jar**

$ find .

.

./simple.sbt

./src

./src/main

./src/main/scala

./src/main/scala/SimpleApp.scala

# Package a jar containing your application

$ sbt package

...

[info] Packaging {..}/{..}/ out/artifacts/top3occurence/ top3occurence.jar

**6. Sumbit application**

to Spark master in Local Mode

[cloudera@quickstart ~]$ spark-submit --class "Top3Words" --master local Downloads/spark-example01/out/artifacts/top3occurence/top3occurence.jar