Big Data CS522

Professor: Prem Nair
Student: Bakhodir

1. **Java code inMapperWordCount**

```
public class WordCountInMapper {

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

 private String word;
 private HashMap<String, Integer> inMapper;

     @Override
      protected void setup(Mapper<LongWritable, Text, Text,
IntWritable>.Context context)throws Exception {
          inMapper = new HashMap<>();
      }

 @Override
 public void map(LongWritable key, Text value, Context context)
throws Exception {
     String line = value.toString();

     StringTokenizer tokenizer = new StringTokenizer(line);
     while (tokenizer.hasMoreTokens()) {
       int count = 1;
       word = tokenizer.nextToken();
       if(inMapper.containsKey(word)) count = inMapper.get(word) + 1;
       inMapper.put(word, count);
 }
}

@Override
protected void cleanup(Mapper<LongWritable, Text, Text,
IntWritable>.Context context)throws Exception {
     for (String key : inMapper.keySet()) {
          int count = inMapper.get(key);
          context.write(new Text(key), new IntWritable(count));
      }
}
public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> {

public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws Exception {
       int sum = 0;
       for (IntWritable val : values)  sum += val.get();
       context.write(key, new IntWritable(sum));
   }
}

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();

   @SuppressWarnings("deprecation")
   Job job = new Job(conf, "averageCA");

   job.setJarByClass(WordCountInMapper.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);

   job.setMapperClass(Map.class);
   job.setReducerClass(Reduce.class);

   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(TextOutputFormat.class);

   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   job.waitForCompletion(true);
  }
}
```

2. Java code Average

```
AverageCA.java


public class AverageCA {

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

 private static int NUM_FIELDS = 7;
 private static String logEntryPattern = "^([a-zA-Z0-9.-_]+) (\\S+)
(\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.+?)\" (\\d{3}) (\\d+)";

 @Override
 public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
     String line = value.toString();

     Pattern p = Pattern.compile(logEntryPattern);
     Matcher matcher = p.matcher(line);

      if (!matcher.matches() || NUM_FIELDS != matcher.groupCount()) {
                System.err.println("error" + line);
                return;
           }

      String ipAddress = matcher.group(1);
      String quantity = matcher.group(7);
      context.write(new Text(ipAddress), new Pair(quantity, "1"));
}

public static class Reduce extends Reducer<Text, Pair, Text,
DoubleWritable> {

 public void reduce(Text key, Iterable<Pair> values, Context context)
     throws IOException, InterruptedException {
        int sum = 0;
        int count = 0;
        for (Pair p : values) {
            sum += Integer.parseInt(p.getKey());
            count += Integer.parseInt(p.getValue());
        }
        context.write(key, new DoubleWritable(sum/count));
    }
}

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();

   @SuppressWarnings("deprecation")
   Job job = new Job(conf, "averageCA");
   job.setJarByClass(AverageCA.class);

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

   job.setMapperClass(Map.class);
   job.setReducerClass(Reduce.class);

   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(TextOutputFormat.class);

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

   job.waitForCompletion(true);
  }
 }
}
```

Pair.java

```
public class Pair implements WritableComparable<Pair>{

	private String key;
	private String value;

	public Pair() {
	}

	public Pair(String key, String value) {
		this.key = key;
		this.value = value;
	}

	public String getKey() {
		return key;
	}

	public void setKey(String key) {
		this.key = key;
	}

	public String getValue() {
		return value;
	}

	public void setValue(String value) {
		this.value = value;
	}

	@Override
	public String toString() {
		return "(" + this.key + ", " + this.value + ")";
	}
```

```
    @Override
    public void readFields(DataInput in) throws IOException {
          key = in.readUTF();
          value = in.readUTF();

    }

    @Override
    public void write(DataOutput out) throws IOException {
          out.writeUTF(key);
          out.writeUTF(value);
    }

    @Override
    public int compareTo(Pair o) {
    return ((Pair) o).key.compareTo(this.key) != 0 ? ((Pair) o).key
                    .compareTo(this.key) * -1 : ((Pair) o).value
                    .compareTo(this.value) * -1;
    }

    @Override
    public int hashCode() {
     final int prime = 31;
     int result = 1;
     result = prime * result + ((key == null) ? 0 :key.hashCode());
     result = prime * result + ((value == null) ? 0 :
value.hashCode());
          return result;
    }

    @Override
    public boolean equals(Object obj) {
          if (this == obj) return true;
          if (obj == null)  return false;
          if (getClass() != obj.getClass())  return false;
          Pair other = (Pair) obj;
          if (key == null) {
             if (other.key != null) return false;
```

```
        }
        else if (!key.equals(other.key)) return false;
        if (value == null) {
           if (other.value != null) return false;
        }
        else if (!value.equals(other.value))  return false;
        return true;
    }
}
```

3. Output

```
10.0.0.153 6383.0
12.22.207.235    7368.0
128.227.88.79    6815.0
142.27.64.35     4488.0
145.253.208.9    4349.0
1513.cps.virtua.com.br      309.0
194.151.73.43    10879.0
195.11.231.210   6032.0
195.230.181.122  2300.0
195.246.13.119   5128.0
200.160.249.68.bmf.com.br   6634.0
200.222.33.33    2300.0
203.147.138.233  2164.0
207.195.59.160   5404.0
208.247.148.12   3067.0
212.21.228.26    2869.0
212.92.37.62     5212.0
213.181.81.4     7649.0
216.139.185.45   6051.0
219.95.17.51     3169.0
3_343_lt_someone 6277.0
4.37.97.186      2446.0
61.165.64.6      3056.0
61.9.4.61  2645.0
64.242.88.10     12710.0
64.246.94.141    0.0
64.246.94.152    0.0
66.213.206.2     3169.0
67.131.107.5     5488.0
```

4. Java code InMapperAverage

```
AverageCAInMapper.java

public class AverageCAInMapper {

public static class Map extends Mapper<LongWritable, Text, Text,
Pair> {
 private static int NUM_FIELDS = 7;
 private static String logEntryPattern = "^([a-zA-Z0-9.-_]+) (\\S+)
(\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.+?)\" (\\d{3}) (\\d+)";
 private HashMap<String, Pair> inMapper;

@Override
protected void setup(Mapper<LongWritable, Text, Text, Pair>.Context
context) throws IOException, InterruptedException {
      inMapper = new HashMap<>();
}

@Override
 public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
     String line = value.toString();

     Pattern p = Pattern.compile(logEntryPattern);
      Matcher matcher = p.matcher(line);

     if (!matcher.matches() || NUM_FIELDS != matcher.groupCount()) {
                System.err.println("error" + line);
                return;
           }

      String ipAddress = matcher.group(1);
      String quantity = matcher.group(7);

      if(!inMapper.containsKey(ipAddress)){
          inMapper.put(ipAddress, new Pair(quantity, "1"));
       }
      else {
         Pair pair = inMapper.get(ipAddress);
         pair.setKey((Long.parseLong((pair.getKey())) +
Long.parseLong(quantity)) + "");
        pair.setValue(((Long.parseLong(pair.getValue()) + 1)) + "");
      }
}
@Override
protected void cleanup(Mapper<LongWritable, Text, Text, Pair>.Context
context) throws IOException, InterruptedException {
      for (String key : inMapper.keySet()) {
          context.write(new Text(key), inMapper.get(key));
      }
  }
}

public static class Reduce extends Reducer<Text, Pair, Text,
DoubleWritable> {

 public void reduce(Text key, Iterable<Pair> values, Context context)
throws IOException, InterruptedException {
       long sum = 0;
        long count = 0;

       for (Pair p : values) {
           sum += Long.parseLong(p.getKey());
           count += Long.parseLong(p.getValue());
       }
       context.write(key, new DoubleWritable(sum/count));
   }
}

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();

   @SuppressWarnings("deprecation")
   Job job = new Job(conf, "averageCAInMapper");
   job.setJarByClass(AverageCAInMapper.class);
```

```
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Pair.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

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

    job.waitForCompletion(true);
  }
 }
```

Pair.java

```java
public class Pair implements WritableComparable<Pair>{
      private String key;
      private String value;

      public Pair() {
      }

      public Pair(String key, String value) {
            this.key = key;
            this.value = value;
      }

      public String getKey() {
            return key;
      }

      public void setKey(String key) {
            this.key = key;
      }

      public String getValue() {
            return value;
      }

      public void setValue(String value) {
            this.value = value;
      }

      @Override
      public String toString() {
            return "(" + this.key + ", " + this.value + ")";
      }

      @Override
      public void readFields(DataInput in) throws IOException {
            key = in.readUTF();
            value = in.readUTF();

      }

      @Override
      public void write(DataOutput out) throws IOException {
            out.writeUTF(key);
            out.writeUTF(value);
      }

     @Override
     public int compareTo(Pair o) {
      return ((Pair) o).key.compareTo(this.key) != 0 ? ((Pair) o).key
                        .compareTo(this.key) * -1 : ((Pair) o).value
                        .compareTo(this.value) * -1;
      }

      @Override
      public int hashCode() {
       final int prime = 31;
       int result = 1;
       result = prime * result + ((key == null) ? 0 :
key.hashCode());
       result = prime * result + ((value == null) ? 0 :
value.hashCode());
            return result;
      }

      @Override
      public boolean equals(Object obj) {
            if (this == obj)  return true;
            if (obj == null)  return false;
            if (getClass() != obj.getClass()) return false;
            Pair other = (Pair) obj;
            if (key == null) {
                  if (other.key != null)  return false;
            }
            else if (!key.equals(other.key))  return false;
             if (value == null) {
                    if (other.value != null)    return false;
             }
            else if (!value.equals(other.value))  return false;
           return true;
      }
}
```

5. Output

```
10.0.0.153 6383.0
12.22.207.235    7368.0
128.227.88.79    6815.0
142.27.64.35     4488.0
145.253.208.9    4349.0
1513.cps.virtua.com.br      309.0
194.151.73.43    10879.0
195.11.231.210   6032.0
195.230.181.122  2300.0
195.246.13.119   5128.0
200.160.249.68.bmf.com.br   6634.0
200.222.33.33    2300.0
203.147.138.233  2164.0
207.195.59.160   5404.0
208.247.148.12   3067.0
212.21.228.26    2869.0
212.92.37.62     5212.0
213.181.81.4     7649.0
216.139.185.45   6051.0
219.95.17.51     3169.0
3_343_lt_someone 6277.0
4.37.97.186      2446.0
61.165.64.6      3056.0
61.9.4.61  2645.0
64.242.88.10     12710.0
64.246.94.141    0.0
64.246.94.152    0.0
66.213.206.2     3169.0
67.131.107.5     5488.0
```

6. Pseudo code for PAIR approach

```
class Mapper

     method Initialize
         H <- new AssociativeArray

    method Map(docid a, doc d)
       for all term w in doc d do
         for all term u in Neighbors(w) do
              H{pair(w, u)} = H{pair(w,u)} + 1;
              H{pair(w, '*') = H{pair(w, '*')} + 1

     method Close
          for all term w in H do
               Emit(pair p, count H{p})

class Reducer

     method Initialize
          starSum = 1

     method Reduce(pair p, counts[H1, H2, H3,....]
          if pair.secondValue == '*' then
               if p.firstValue is new then
                    starSum = 1;
               starSum += sum(counts)
          else then
               Emit(pair p, sum(counts) / startSum)
```

7. Java code for Pair approach

```
Pair.java


public class Pair implements WritableComparable<Pair>{

	private String key;
	private String value;

	public Pair() {
	}

	public Pair(String key, String value) {
		this.key = key;
		this.value = value;
	}

	public String getKey() {
		return key;
	}

	public void setKey(String key) {
		this.key = key;
	}

	public String getValue() {
		return value;
	}

	public void setValue(String value) {
		this.value = value;
	}

	@Override
	public String toString() {
		return "(" + this.key + ", " + this.value + ")";
	}
	@Override
	public void readFields(DataInput in) throws IOException {
		key = in.readUTF();
		value = in.readUTF();

	}

	@Override
	public void write(DataOutput out) throws IOException {
		out.writeUTF(key);
		out.writeUTF(value);
	}

	@Override
	public int compareTo(Pair o) {
	 return ((Pair) o).key.compareTo(this.key) != 0 ? ((Pair) o).key
				.compareTo(this.key) * -1 : ((Pair) o).value
				.compareTo(this.value) * -1;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((key == null) ? 0 :
key.hashCode());
		result = prime * result + ((value == null) ? 0 :
value.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)	return true;
		if (obj == null)	return false;
		if (getClass() != obj.getClass())	return false;
		Pair other = (Pair) obj;
```

```
            if (key == null) {
                 if (other.key != null)     return false;
            }
            else if (!key.equals(other.key)) return false;
            if (value == null) {
                 if (other.value != null)   return false;
            }
            else if (!value.equals(other.value))  return false;
            return true;
       }
  }
```

PairsRelativeFrequence.java

```
public class PairsRelativeFrequence {

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

      private HashMap<Pair, Double> mapper;

      @Override
      protected void setup(Mapper<LongWritable, Text, Pair,
DoubleWritable>.Context context) throws Exception{
            mapper = new HashMap<>();
      }

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

          String line = value.toString();
          List<String> record = new ArrayList<>();
          StringTokenizer tokenizer = new StringTokenizer(line);

          while (tokenizer.hasMoreTokens()) {
              String s = tokenizer.nextToken();
              record.add(s);
          }

         for(int i = 0; i < record.size(); i++){
          double starCount = 0;
          Pair starPair = new Pair(String.valueOf(record.get(i)), "*");

          if(mapper.containsKey(starPair)) starCount =
mapper.get(starPair);

          List<String> neighbors = getNeighborhood(i + 1,
record.get(i),  record);
         for(String word : neighbors){
           starCount++;
```

```
            Pair pair = new Pair(String.valueOf(record.get(i)), word);
            double pairCount = 0;
            if(mapper.containsKey(pair))  pairCount = mapper.get(pair);
                    mapper.put(pair, ++pairCount);
               }
            mapper.put(starPair, starCount);
            }
        }

      @Override
       protected void cleanup(Mapper<LongWritable, Text, Pair,
DoubleWritable>.Context context) throws Exception {
              for(Pair p : mapper.keySet()){
                   context.write(p, new DoubleWritable(mapper.get(p)));
              }
          }
   }

   private static List<String> getNeighborhood(int index, String word,
List<String> record){
              List<String> neighbors = new ArrayList<>();
              while(index < record.size()){
                   if(record.get(index).equals(word)) return neighbors;
                   neighbors.add(record.get(index));
                   index++;
              }
              return neighbors;
       }

   public static class Reduce extends Reducer<Pair, DoubleWritable,
Pair, DoubleWritable> {

      private HashMap<Pair, Double> reducer;
      private double starSum = 0;
       public void reduce(Pair pair, Iterable<DoubleWritable> values,
Context context) throws IOException, InterruptedException {

       double sum = 0;
       for(DoubleWritable val: values)    sum += val.get();

       if(pair.getValue().equals("*"))  starSum = sum;
       else context.write(pair, new DoubleWritable(sum / starSum));
         }
     }

      public static void main(String[] args) throws Exception {
         Configuration conf = new Configuration();

         @SuppressWarnings("deprecation")
          Job job = new Job(conf, "pairsrelativefrequence");
         job.setJarByClass(PairsRelativeFrequence.class);
         job.setOutputKeyClass(Pair.class);
         job.setOutputValueClass(DoubleWritable.class);
         job.setMapperClass(Map.class);
         job.setReducerClass(Reduce.class);
         job.setInputFormatClass(TextInputFormat.class);
         job.setOutputFormatClass(TextOutputFormat.class);
         FileInputFormat.addInputPath(job, new Path(args[0]));
         FileOutputFormat.setOutputPath(job, new Path(args[1]));
         job.waitForCompletion(true);
      }
     }
```

7. Result of PAIR approach

```
(A10, B12) 0.5
(A10, D76) 0.5
(A12, A10) 0.08333333333333333
(A12, B12) 0.16666666666666666
(A12, B76) 0.08333333333333333
(A12, C31) 0.3333333333333333
(A12, D76) 0.3333333333333333
(B12, A10) 0.0625
(B12, A12) 0.25
(B12, B76) 0.0625
(B12, C31) 0.3125
(B12, D76) 0.3125
(B76, A10) 0.16666666666666666
(B76, B12) 0.16666666666666666
(B76, C31) 0.3333333333333333
(B76, D76) 0.3333333333333333
(C31, A10) 0.08333333333333333
(C31, A12) 0.25
(C31, B12) 0.16666666666666666
(C31, B76) 0.08333333333333333
(C31, D76) 0.4166666666666667
(D76, A10) 0.07692307692307693
(D76, A12) 0.3076923076923077
(D76, B12) 0.23076923076923078
(D76, B76) 0.07692307692307693
(D76, C31) 0.3076923076923077
```

8. Pseudo code for STRIPE approach

```
class Mapper

 method Initialize
           H <- new AssociativeArray

      method Map(docid a, doc d)
          for all term w in doc d do
             if H{w} == null then
                H{w} <- new AssociativeArray
             for all term u in Neighbors(w) do
                H{w}{u} = H{w}{u} + 1

      method Close
             for all term w in H do
                   Emit(term w, count H{w})

 class Reducer

      method Reduce(term w, stripes[H1, H2, H3,....]

             sum = 0
             Hf <- new AssociativeArray

             for all stripe H in stripes[H1, H2, H3, ....]
                 for all term t in H do
                     sum += H{t}
                     Hf{t} = Hf{t} + H{t}

           for all term t in Hf do
               Emit(term w, H{t})
```

10. Java code for STRIPE approach

Pair.java

```
public class Pair implements WritableComparable<Pair>{

	private String key;
	private String value;

	public Pair() {
	}

	public Pair(String key, String value) {
		this.key = key;
		this.value = value;
	}

	public String getKey() {
		return key;
	}

	public void setKey(String key) {
		this.key = key;
	}

	public String getValue() {
		return value;
	}

	public void setValue(String value) {
		this.value = value;
	}

	@Override
	public String toString() {
		return "(" + this.key + ", " + this.value + ")";
	}
```

```
        @Override
        public void readFields(DataInput in) throws IOException {
              key = in.readUTF();
              value = in.readUTF();

        }

        @Override
        public void write(DataOutput out) throws IOException {
              out.writeUTF(key);
              out.writeUTF(value);
        }

        @Override
        public int compareTo(Pair o) {
         return ((Pair) o).key.compareTo(this.key) != 0 ? ((Pair) o).key
                          .compareTo(this.key) * -1 : ((Pair) o).value
                          .compareTo(this.value) * -1;
        }

      @Override
      public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((key == null) ? 0 : key.hashCode());
        result = prime * result + ((value == null) ? 0 :
value.hashCode());
        return result;
       }

        @Override
        public boolean equals(Object obj) {
              if (this == obj)      return true;
              if (obj == null)      return false;
              if (getClass() != obj.getClass())     return false;
              Pair other = (Pair) obj;
```

```
            if (key == null) {
                 if (other.key != null)      return false;
            }
            else if (!key.equals(other.key))  return false;
            if (value == null) {
                 if (other.value != null)     return false;
            }
            else if (!value.equals(other.value))  return false;
            return true;
       }
  }
```

CustomMapWritable.java

```
public class CustomMapWritable extends MapWritable{

      @Override
      public String toString() {
       StringBuilder b = new StringBuilder();
       for(Writable key: this.keySet()){
            b.append("(").append(key).append(":").append(this.get(key)
+ "), ");
        }
       return b.toString();
      }
}
```

StripeRelativeFrequence.java

```
public class StripeRelativeFrequence {
     public static class Map extends Mapper<LongWritable, Text,
Text, CustomMapWritable> {

     private HashMap<String, HashMap<String, Double>> mapper;

     @Override
     protected void setup(Mapper<LongWritable, Text, Text,
CustomMapWritable>.Context context) throws Exception {
                mapper = new HashMap<>();
           }

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

          String line = value.toString();
          List<String> record = new ArrayList<>();
          StringTokenizer tokenizer = new StringTokenizer(line);

          while (tokenizer.hasMoreTokens()) {
               String s = tokenizer.nextToken();
               record.add(s);
              }

          for(int i = 0; i < record.size(); i++){
           if(!mapper.containsKey(record.get(i)))  {
             mapper.put(record.get(i), new HashMap<String,
Double>());
            }
            HashMap<String, Double> map = mapper.get(record.get(i));

            List<String> neighbors = getNeighborhood(i + 1,
record.get(i),  record);

            for(String word : neighbors){
                 double count = 0;
                 if(map.containsKey(word)) count = map.get(word);
                      map.put(word, ++count);
                 }
                 mapper.put(record.get(i), map);
              }
          }

         @Override
         protected void cleanup(Mapper<LongWritable, Text, Text,
CustomMapWritable>.Context context) throws Exception {

          for(String key : mapper.keySet()){
           CustomMapWritable mWritable = new CustomMapWritable();
                HashMap<String, Double> map = mapper.get(key);
                for(String s: map.keySet()){
                     mWritable.put(new Text(s) , new
DoubleWritable(map.get(s)));
                }
                context.write(new Text(key), mWritable);
                }
             }
       }

       private static List<String> getNeighborhood(int index, String
word, List<String> record){
           List<String> neighbors = new ArrayList<>();
           while(index < record.size()){
                if(record.get(index).equals(word)) return neighbors;
                     neighbors.add(record.get(index));
                     index++;
                }
                return neighbors;
           }
```

```
       public static class Reduce extends Reducer<Text,
CustomMapWritable, Text, CustomMapWritable> {

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

        double sum = 0;

       Iterator<CustomMapWritable> it = values.iterator();
        CustomMapWritable res = new CustomMapWritable();

        while(it.hasNext()){
          MapWritable mWritable = it.next();
         for(Entry<Writable, Writable> entry : mWritable.entrySet()){
           Text k = (Text)entry.getKey();
           DoubleWritable v = ((DoubleWritable) entry.getValue());
           sum += v.get();

           if(!res.containsKey(k))  res.put(k, v);
           else {
             DoubleWritable d = (DoubleWritable) res.get(k);
              res.put(key,  new DoubleWritable(v.get() + d.get()));
            }
           }
         }

         for(Entry<Writable, Writable> entry : res.entrySet()){
             Text k = (Text)entry.getKey();
           double v = ((DoubleWritable) entry.getValue()).get();
           res.put(k, new DoubleWritable(v / sum));
          }
            context.write(key, res);
        }
      }
```

```
        public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();

             @SuppressWarnings("deprecation")
             Job job = new Job(conf, "striperelativefrequence");
             job.setJarByClass(StripeRelativeFrequence.class);
             job.setOutputKeyClass(Text.class);
             job.setOutputValueClass(CustomMapWritable.class);

             job.setMapperClass(Map.class);
             job.setReducerClass(Reduce.class);

             job.setInputFormatClass(TextInputFormat.class);
             job.setOutputFormatClass(TextOutputFormat.class);

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

             job.waitForCompletion(true);
          }
}
```

10. Result of STRIPE approach

```
A10   (B12:0.5), (D76:0.5)

A12   (B76:0.08333333333333333), (A10:0.08333333333333333),
(B12:0.16666666666666666), (D76:0.3333333333333333),
(C31:0.3333333333333333)

B12   (B76:0.0625), (A12:0.25), (A10:0.0625), (D76:0.3125),
(C31:0.3125)

B76   (A10:0.16666666666666666), (B12:0.16666666666666666),
(D76:0.3333333333333333), (C31:0.3333333333333333)

C31   (B76:0.08333333333333333), (A12:0.25),
(A10:0.08333333333333333), (B12:0.16666666666666666),
(D76:0.4166666666666667)

D76   (B76:0.07692307692307693), (A12:0.3076923076923077),
(A10:0.07692307692307693), (B12:0.23076923076923078),
(C31:0.3076923076923077)
```

12. Pseudo code for HYBRID approach

```
class Mapper

      method Initialize
          H <- new AssociativeArray

     method Map(docid a, doc d)
         for all term w in doc d do
            for all term u in Neighbors(w) do
                 H{pair(w, u)} = H{pair(w,u)} + 1;

      method Close
            for all term w in H do
                 Emit(pair p, count H{p})


 class Reducer

      method Initialize
            Wprev <- 0
             H = new AssociativeArray

      method reduce(pair(w,u), [C1, C2....])
             if(w != Wprev && Wprev != 0) then
                    total = total(H)
                     H = H / total
                    Emit( Wprev, H)
                    H = new AssociatiaveArray
              sum = 0
              for all c in [C1, C2 ...] do
                    sum += c
              H{u} = sum
              Wprev = w

      method Close
        total = total(H)
        H = H / total
        Emit(Wprev, H)
```

13. Java code Hybrid approach

```
Pair.java

public class Pair implements WritableComparable<Pair> {

      private String key;
      private String value;

      public Pair() {
      }

      public Pair(String key, String value) {
            this.key = key;
            this.value = value;
      }

      public String getKey() {
            return key;
      }

      public void setKey(String key) {
            this.key = key;
      }

      public String getValue() {
            return value;
      }

      public void setValue(String value) {
            this.value = value;
      }

      @Override
      public String toString() {
            return "(" + this.key + ", " + this.value + ")";
      }
```

```
	@Override
	public void readFields(DataInput in) throws IOException {
		key = in.readUTF();
		value = in.readUTF();

	}

	@Override
	public void write(DataOutput out) throws IOException {
		out.writeUTF(key);
		out.writeUTF(value);
	}

    @Override
    public int compareTo(Pair o) {
	 return ((Pair) o).key.compareTo(this.key) != 0 ? ((Pair) o).key
				.compareTo(this.key) * -1 : ((Pair) o).value
				.compareTo(this.value) * -1;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((key == null) ? 0 :
key.hashCode());
		result = prime * result + ((value == null) ? 0 :
value.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)	return true;
		if (obj == null)	return false;
		if (getClass() != obj.getClass())	return false;
		Pair other = (Pair) obj;
```

```
            if (key == null) {
                 if (other.key != null)     return false;
            }
            else if (!key.equals(other.key))   return false;
            if (value == null) {
                 if (other.value != null)   return false;
            }
            else if (!value.equals(other.value))  return false;
            return true;
       }
  }
```

Hybrid.java

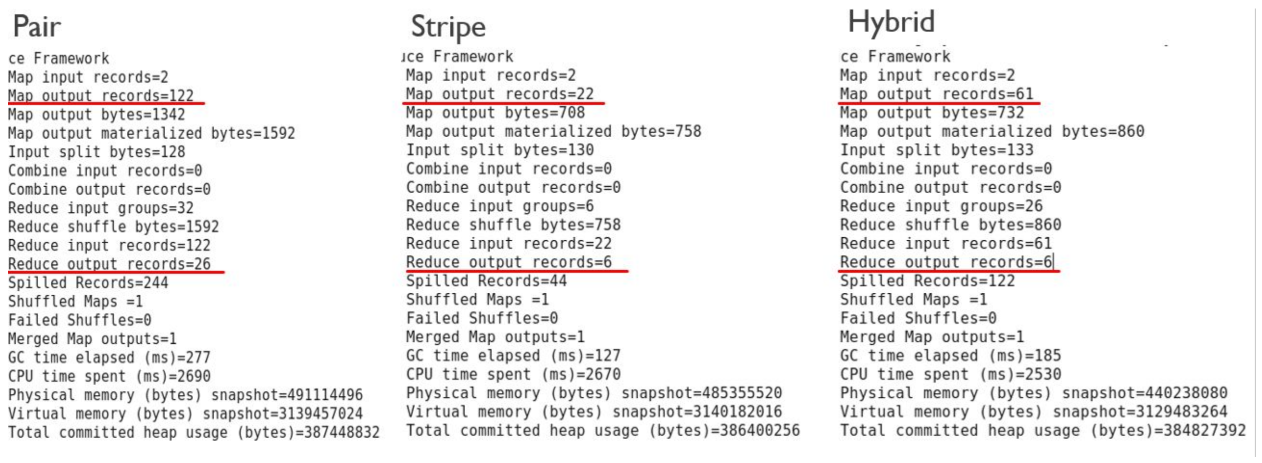
```
public class Hybrid {

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

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

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

      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      List<String> record = new ArrayList<String>();

      while (tokenizer.hasMoreTokens()) {
           record.add(tokenizer.nextToken());
      }

      int index = 0;
      for (int i = 0; i < record.size(); i++) {
           if (index == record.size())
           continue;
           List<String> neighbors = getNeighborhood(index + 1,
record.get(i), record);

           for (String k : neighbors) {
                Pair pair = new Pair(record.get(i), k);
                context.write(pair, one);
           }
        }
       }
      }

      private static List<String> getNeighborhood(int index, String
word, List<String> record) {
           List<String> neighbors = new ArrayList<>();
```

```
            while(index < record.size()){
                  if(record.get(index).equals(word)) return neighbors;
                  neighbors.add(record.get(index));
                  index++;
            }
            return neighbors;
      }

      public static class Reduce extends Reducer<Pair,
DoubleWritable, Text, Text> {
            public static String previous;
            public static MapWritable h;

            @Override
            public void setup(Context context) throws Exception {
                  h = new MapWritable();
                  previous = null;
            }

            public void reduce(Pair key, Iterable<DoubleWritable>
values, Context context) throws IOException, InterruptedException {
              double total = 0;
              double sum = 0;
              if (!key.getKey().equals(previous) && previous != null){
                      total = total(h);
                       h = frequence(h, total);
                      context.write(new Text(previous), new
Text(toString(h)));
                       h = new MapWritable();
                  }

                for(DoubleWritable value : values)
                  sum+=value.get();

                h.put(new Text(key.getValue()), new
DoubleWritable(sum));
                previous = key.getKey();
            }

            @Override
            public void cleanup(Context context) throws Exception {

                  double total = total(h);
                  h = frequence(h, total);
                  context.write(new Text(previous), new
Text(toString(h)));
            }


            public String toString(MapWritable h) {
                  StringBuilder sb = new StringBuilder();
                  for(java.util.Map.Entry<Writable, Writable> item:
h.entrySet()){
                       double val =
((DoubleWritable)item.getValue()).get();
                       sb.append("(" + item.getKey().toString() + " :
" + (val) + ")");
                  }
                  return sb.toString();
            }

            public double total(MapWritable h) {
                 double result = 0;
                 for (Entry<Writable, Writable> hItem : h.entrySet())
{
                      double i = ((DoubleWritable)
hItem.getValue()).get();
                      result += i;
                 }
                 return result;
            }

            public MapWritable frequence(MapWritable h, double total){
               for (Entry<Writable, Writable> hItem : h.entrySet()) {
                      double i = ((DoubleWritable)
hItem.getValue()).get();
                      h.put(hItem.getKey(), new
DoubleWritable(i/total));
                }
                return h;
           }
     }

     public static void main(String[] args) throws Exception {
           Configuration conf = new Configuration();

           Job job = new Job(conf, "hybrid");
           job.setJarByClass(Hybrid.class);
           job.setOutputKeyClass(Text.class);
           job.setOutputValueClass(Text.class);

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

           job.setMapperClass(Map.class);
           job.setReducerClass(Reduce.class);

           job.setInputFormatClass(TextInputFormat.class);
           job.setOutputFormatClass(TextOutputFormat.class);

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

           job.waitForCompletion(true);
     }
}
```

14. Result of HYBRID approach

```
A10   (B76 : 0.14285714285714285)(A12 : 0.14285714285714285)(D76 :
0.2857142857142857)(C31 : 0.42857142857142855)

A12   (B12 : 0.375)(D76 : 0.5)(C31 : 0.125)

B12   (B76 : 0.1111111111111111)(A12 : 0.1111111111111111)(A10 :
0.1111111111111111)(D76 : 0.3333333333333333)(C31 :
0.3333333333333333)

B76   (A12 : 0.3333333333333333)(D76 : 0.3333333333333333)(C31 :
0.3333333333333333)

C31   (A12 : 0.3333333333333333)(B12 : 0.3333333333333333)(D76 :
0.3333333333333333)

D76   (C31 : 1.0)
```

15. Time and Resource comparison: Pair-Stripe-Hybrid



Pair

```
ce Framework
Map input records=2
Map output records=122
Map output bytes=1342
Map output materialized bytes=1592
Input split bytes=128
Combine input records=0
Combine output records=0
Reduce input groups=32
Reduce shuffle bytes=1592
Reduce input records=122
Reduce output records=26
Spilled Records=244
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=277
CPU time spent (ms)=2690
Physical memory (bytes) snapshot=491114496
Virtual memory (bytes) snapshot=3139457024
Total committed heap usage (bytes)=387448832
```

Stripe

```
ıce Framework
Map input records=2
Map output records=22
Map output bytes=708
Map output materialized bytes=758
Input split bytes=130
Combine input records=0
Combine output records=0
Reduce input groups=6
Reduce shuffle bytes=758
Reduce input records=22
Reduce output records=6
Spilled Records=44
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=127
CPU time spent (ms)=2670
Physical memory (bytes) snapshot=485355520
Virtual memory (bytes) snapshot=3140182016
Total committed heap usage (bytes)=386400256
```

Hybrid

```
ce Framework
Map input records=2
Map output records=61
Map output bytes=732
Map output materialized bytes=860
Input split bytes=133
Combine input records=0
Combine output records=0
Reduce input groups=26
Reduce shuffle bytes=860
Reduce input records=61
Reduce output records=6
Spilled Records=122
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=185
CPU time spent (ms)=2530
Physical memory (bytes) snapshot=440238080
Virtual memory (bytes) snapshot=3129483264
Total committed heap usage (bytes)=384827392
```

16. Problem statement (including link to dataset, categorical variable chosen and numeric variable chosen)

File name: Cane

Description
The cane data frame has 180 rows and 5 columns. The data frame represents a randomized block design with 45 varieties of sugar-cane and 4 blocks.

Format

This data frame contains the following columns:
n - The total number of shoots in each plot.
r - The number of diseased shoots.
x - The number of pieces of the stems, out of 50, planted in each plot.
var - A factor indicating the variety of sugar-cane in each plot.
block - A factor for the blocks.

Details

The aim of the experiment was to classify the varieties into resistant, intermediate and susceptible to a disease called "coal of sugar-cane" (carvao da cana-de-acucar). This is a disease that is common in sugar-cane plantations in certain areas of Brazil.
For each plot, fifty pieces of sugar-cane stem were put in a solution containing the disease agent and then some were planted in the plot. After a fixed period of time, the total number of shoots and the number of diseased shoots were recorded.

Dataset link: https://vincentarelbundock.github.io/Rdatasets/doc/boot/cane.html

Categorical variable chosen: A, B, C, D

Numeric variable chosen: n - The total number of shoots in each plot.

17. Scala Code

RowNumAndCategory.scala

```
case class RowNumAndCategory(c: String, v: Double) {
  var category: String = this.c
  var value: Double = this.v
}
```

SparkObject.scala

```
object SparkObject {

  def main(args: Array[String]): Unit = {
    // spark config
    val sparkConf = new SparkConf().setAppName("Spark Object")
      .setMaster("local[2]").set("spark.executor.memory", "1g")
    val sc = new SparkContext(sparkConf)

    // Step 2

    val cane = sc.textFile("cane.csv") // creating RDD
    // a file is to write the result into
    val writer = new PrintWriter(new File("result.txt"))

    val rows = cane.map(line => line.split(",")) // making an array
    val header = rows.first() // first row in the array

    // Step 3

    // get list of all category and convert to RowNumAndCategory that
has category name and value
    val population = rows.filter(line => !(line sameElements header))
      .map(line => RowNumAndCategory(line(5).substring(1,
line(5).length - 1), line(1).toDouble))

    // Step 4

    // get list of values in category A
    val categoryA = population.filter(x =>
x.category.equals("A")).map(x => x.value)
    // get list of values in category B
    val categoryB = population.filter(x =>
x.category.equals("B")).map(x => x.value)
    // get list of values in category C
    val categoryC = population.filter(x =>
x.category.equals("C")).map(x => x.value)
```

```
    // get list of values in category D
    val categoryD = population.filter(x =>
x.category.equals("D")).map(x => x.value)

    val meanA: Double = categoryA.sum() // get mean of A
    val meanB = categoryB.sum() // get mean of B
    val meanC = categoryC.sum() // get mean of C
    val meanD = categoryD.sum() // get mean of D

    val sumA = categoryA.map(x => Math.pow(x - meanA, 2)).sum() //
sum of numbers in A
    val countA = categoryA.map(x => Math.pow(x - meanA, 2)).count()
// count of numbers in A
    val varA: Double = sumA / countA // variance of category A

    val sumB = categoryB.map(x => Math.pow(x - meanB, 2)).sum() //
sum of numbers in B
    val countB = categoryB.map(x => Math.pow(x - meanB, 2)).count()
// count of numbers in A
    var varB: Double = sumB / countB // variance of category B

    val sumC = categoryC.map(x => Math.pow(x - meanC, 2)).sum() //
sum of numbers in C
    val countC = categoryC.map(x => Math.pow(x - meanC, 2)).count()
// count of numbers in C
    val varC: Double = sumC / countC // variance of category C

    val sumD = categoryD.map(x => Math.pow(x - meanD, 2)).sum() //
sum of numbers in D
    val countD = categoryD.map(x => Math.pow(x - meanD, 2)).count()
// count of numbers in D
    val varD: Double = sumD / countD // variance of category D

    // write to the file
    writer.write("Category    Mean    Variance \n")
    writer.write("A          " + "%.2f".format(meanA) + "
%.2f".format(varA) + "\n")
    writer.write("B          " + "%.2f".format(meanB) + "
%.2f".format(varB) + "\n")
    writer.write("C          " + "%.2f".format(meanC) + "
%.2f".format(varC) + "\n")
    writer.write("D          " + "%.2f".format(meanD) + "
%.2f".format(varD) + "\n")
    writer.write("\n")

    // Step 4

    // get sample of the data with 0.25, no replacement
    val samplePopulation = population.sample(withReplacement = false,
0.25)

    // Step 5

    var mean1000A: Double = 0
    var var1000A: Double = 0

    var mean1000B: Double = 0
    var var1000B: Double = 0

    var mean1000C: Double = 0
    var var1000C: Double = 0

    var mean1000D: Double = 0
    var var1000D: Double = 0

    for (i <- 1 to 10) {

      // Step 5a

      // get sample of the data with 1, with replacement
      val resampleData = samplePopulation.sample(withReplacement =
true, 1)

      // Step 5b, 5c

      // get list of values in category A
      val categoryA = resampleData.filter(x =>
x.category.equals("A")).map(x => x.value)
      // get list of values in category B
      val categoryB = resampleData.filter(x =>
x.category.equals("B")).map(x => x.value)
      // get list of values in category C
      val categoryC = resampleData.filter(x =>
x.category.equals("C")).map(x => x.value)
      // get list of values in category D
      val categoryD = resampleData.filter(x =>
x.category.equals("D")).map(x => x.value)

      val meanA = categoryA.sum() // get mean of A
      val meanB = categoryB.sum() // get mean of B
      val meanC = categoryC.sum() // get mean of C
      val meanD = categoryD.sum() // get mean of D

      val sumA = categoryA.map(x => Math.pow(x - meanA, 2)).sum()
      val countA = categoryA.map(x => Math.pow(x - meanA, 2)).count()
      val varA = sumA / countA
      var1000A += varA
      mean1000A += meanA

      val sumB = categoryB.map(x => Math.pow(x - meanA, 2)).sum()
      val countB = categoryB.map(x => Math.pow(x - meanA, 2)).count()
      val varB = sumB / countB
      var1000B += varB
      mean1000B += meanB

      val sumC = categoryC.map(x => Math.pow(x - meanC, 2)).sum()
      val countC = categoryC.map(x => Math.pow(x - meanC, 2)).count()
      val varC = sumC / countC
      var1000C += varC
      mean1000C += meanC

      val sumD = categoryD.map(x => Math.pow(x - meanD, 2)).sum()
      val countD = categoryD.map(x => Math.pow(x - meanD, 2)).count()
      val varD = sumD / countD
```

```
        var1000D += varD
        mean1000D += meanD
      }

      // Step 6
      // write to the file
      writer.write("Category    Mean    Variance \n")
      writer.write("A          " + "%.2f".format(mean1000A / 10) + "
%.2f".format(var1000A / 10) + "\n")
      writer.write("B          " + "%.2f".format(mean1000B / 10) + "
%.2f".format(var1000B / 10) + "\n")
      writer.write("C          " + "%.2f".format(mean1000C / 10) + "
%.2f".format(var1000C / 10) + "\n")
      writer.write("D          " + "%.2f".format(mean1000D / 10) + "
%.2f".format(var1000D / 10) + "\n")
      writer.close()
    }
  }
```

18. Output

```
Category     Mean     Variance
A           5165.00  25506899.96
B           5629.00  30294912.98
C           5095.00  24819936.98
D           5377.00  27643330.13

Category     Mean     Variance
A           2025.50  3837507.58
B           1344.20  3822068.16
C           1789.70  2914760.96
D           1554.10  2186000.99
```