Project

Course: Big Data [CS522]
Professor: Prem Nair
Student: Ganijon Rahimov [985565]

Objectives

1. Compute Relative frequencies
    - Hadoop MapReduce algorithms (Java)
2. Analyze apache access logs
    - Spark algorithms (Scala)

Compute Relative Frequencies

- Pairs approach algorithm
- Stripes approach algorithm
- Hybrid approach algorithm
- Comparison

Pairs approach - pseudo code

```
class Mapper

     method map(docid id, doc d)

          for all term w in doc d do

               for all term u in N(w) do

                    Emit(pair(w;u), 1)
                    Emit(pair(w;*), 1)
```

```
class Reducer

     method initialize
          total = 0;

     method reduce(pair p, count [c1, c2, ...])
          sum = sum([c1, c2, ...])
          if(p.right == '*')
               total = sum
          else
               Emit(p, sum/total)
```

Pairs approach - Mapper class

```java
public class PairsMapper extends Mapper<LongWritable, Text, WordPair, IntWritable> {
    private final IntWritable one = new IntWritable(1);
    private final WordPair pair = new WordPair();

    @Override
    public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException {
        List<String> words = Arrays.asList(values.toString().split("\\s"));
        int wordIndex = 0;
        for (String word : words) {
            pair.setWord(word);
            for (String neighbor : getNeighbors(words, wordIndex)) {
                pair.setNeighbor(neighbor);
                context.write(pair, one);
                pair.setNeighbor("*");
                context.write(pair, one);
            }
            wordIndex++;
        }
    }

    private List<String> getNeighbors(List<String> words, int wordIndex) {
        List<String> neigbors = new ArrayList<>();
        for (int i = wordIndex + 1; i < words.size(); i++) {
            if (words.get(wordIndex).equals(words.get(i)))
                break;
            neigbors.add(words.get(i));
        }
        return neigbors;
    }
}
```

Pairs approach - Reducer class

```java
public class PairsReducer extends Reducer<WordPair, IntWritable, WordPair, Text> {

    private int total;

    @Override
    protected void setup(Reducer<WordPair, IntWritable, WordPair, Text>.Context context) throws IOException, InterruptedException {
        super.setup(context);
        total = 0;
    }

    @Override
    public void reduce(WordPair pair, Iterable<IntWritable> counts, Context context) throws IOException, InterruptedException {
        int sum = sum(counts);
        if (pair.getNeighbor().equals(new Text("*"))) {
            total = sum;
        } else {
            String relativeFrequency = sum + "/" + total;
            context.write(pair, new Text(relativeFrequency));
        }
    }

    private int sum(Iterable<IntWritable> values) {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        return sum;
    }
}
```

Pairs approach - Output

```
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/pairs/output/1/*

(A10, B12)	1/2
(A10, D76)	1/2
(A12, A10)	1/12
(A12, B12)	3/12
(A12, B76)	1/12
(A12, C31)	2/12
(A12, D76)	5/12
(B12, A10)	1/15
(B12, A12)	4/15
(B12, B76)	1/15
(B12, C31)	3/15
(B12, D76)	6/15
(B76, A10)	1/6
(B76, B12)	2/6
(B76, C31)	1/6
(B76, D76)	2/6
(C31, A10)	1/17
(C31, A12)	4/17
(C31, B12)	4/17
(C31, B76)	1/17
(C31, D76)	7/17
(D76, A10)	1/12
(D76, A12)	4/12
(D76, B12)	4/12
(D76, B76)	1/12
(D76, C31)	2/12
```

Stripes approach - pseudo code

```
class Mapper

     method map(docid id, doc d)
          for all term w in doc d do
               H = new AssociativeArray
               for all term u in N(w) do
                    H{u} = H{u} + 1
               Emit(term w, stripe H)
```

```
class Reducer

     method reduce(term w, stripes [H1,H2, ...])
          Hf = new AssociativeArray
          for all stripe H in stripes[H1, H2, ...] do
               sum(Hf, H)         // element-wise sum
               total = total(Hf) // element-wise add
          Emit(term w, stripe Hf/total)
```

Stripes approach - Mapper class

```java
public class StripesMapper extends Mapper<LongWritable, Text, Text, Stripe> {
    private Stripe occurenceMap = new Stripe();

    @Override
    public void map(LongWritable key, Text line, Context context) throws IOException, InterruptedException {
        List<String> words = Arrays.asList(line.toString().split("\\s"));
        int wordIndex = 0;
        for (String word : words) {
            occurenceMap.clear();
            for (String neighbor : getNeighbors(words, wordIndex)) {
                IntWritable neighborCount = new IntWritable(1);
                if (occurenceMap.containsKey(neighbor)) {
                    neighborCount = (IntWritable) occurenceMap.get(neighbor);
                    neighborCount.set(1 + neighborCount.get());
                }
                occurenceMap.put(new Text(neighbor), neighborCount);
            }
            wordIndex++;
            context.write(new Text(word), occurenceMap);
        }
    }

    private List<String> getNeighbors(List<String> words, int wordIndex) {
        List<String> neighbors = new ArrayList<>();
        for (int i = wordIndex + 1; i < words.size(); i++) {
            if (words.get(wordIndex).equals(words.get(i)))
                break;
            neighbors.add(words.get(i));
        }
        return neighbors;
    }
}
```

Stripes approach - Reducer class

```
14
15  public class StripesReducer extends Reducer<Text, Stripe, Text, Stripe> {
16
17      @Override
18      public void reduce(Text word, Iterable<Stripe> stripes, Context context) throws IOException, InterruptedException {
19          Stripe stripe = new Stripe();
20          for (Stripe s : stripes)
21              mergeStripes(stripe, s);
22          divideByTotal(stripe, getTotal(stripe));
23          context.write(word, stripe);
24      }
25      private void mergeStripes(Stripe stripe, Stripe s) {
26          for (Writable neighbor : s.keySet()) {
27              IntWritable neighborCount = (IntWritable) s.get(neighbor);
28              if (stripe.containsKey(neighbor)) {
29                  IntWritable count = (IntWritable) stripe.get(neighbor);
30                  neighborCount.set(neighborCount.get() + count.get());
31              }
32              stripe.put(neighbor, neighborCount);
33          }
34      }
35      private void divideByTotal(Stripe stripe, int total) {
36          Text newValue = new Text();
37          IntWritable oldValue;
38          for (Writable key : stripe.keySet()) {
39              oldValue = (IntWritable) stripe.get(key);
40              newValue.set(oldValue.toString() + "/" + String.valueOf(total));
41              stripe.put(key, newValue);
42          }
43      }
44      private int getTotal(Stripe stripe) {
45          int total = 0;
46          for (Writable key : stripe.keySet())
47              total += ((IntWritable) stripe.get(key)).get();
48          return total;
49      }
50  }
```

Stripes approach - Output

```
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/stripes/output/1/*

A10     { B12: 1/2, D76: 1/2 }

A12     { B76: 2/10, A10: 2/10, B12: 2/10, D76: 2/10, C31: 2/10 }

B12     { B76: 3/13, A10: 3/13, A12: 3/13, D76: 3/13, C31: 3/13 }

B76     { A10: 1/4, B12: 1/4, D76: 1/4, C31: 1/4 }

C31     { B76: 4/13, A10: 4/13, A12: 4/13, B12: 4/13, D76: 4/13 }

D76     { B76: 2/12, A10: 2/12, A12: 2/12, B12: 2/12, C31: 2/12 }
```

Hybrid approach - pseudo code

```
class Mapper

     method initialize
          H = new AssociativeArray

     method map(docid id, doc d)
          for all term w in doc d do
               for all term u in N(w) do
                    H{u} = H{u} + 1
               Emit(pair (w;u), H{u})
```

```
class Reducer

     method initialize
          H = new AssociativeArray
          prev = null
          total = 0

     method reduce(pair (w;u), counts [c1,c2,…])
          if(w ≠ prev && prev ≠ null)
               Emit(prev, H / total)
               H = new AssociativeArray
               total = 0
          sum = sum ([c1, c2, ...])
          total = total + sum;
          H{u} = sum
          prev = w

     method close
          Emit(prev, H / total)
```

Hybrid approach - Mapper class

```
17  public class HybridMapper extends Mapper<LongWritable, Text, WordPair, IntWritable> {
18
19      private HashMap<WordPair, Integer> outputMap = new HashMap<>();
20
21      @Override
22      public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException {
23
24          String input = values.toString();
25          String[] readLines = input.split("//.*\n");
26
27          for (String line : readLines) {
28              String[] words = line.split("\\s");
29              for (int i = 0; i < words.length - 1; i++) {
30                  for (int j = i + 1; j < words.length; j++) {
31                      if (words[i].equals(words[j]))
32                          break;
33                      WordPair pair = new WordPair(words[i], words[j]);
34                      if (outputMap.get(pair) != null)
35                          outputMap.put(pair, outputMap.get(pair) + 1);
36                      else
37                          outputMap.put(pair, new Integer(1));
38                  }
39              }
40          }
41
42          for (Entry<WordPair, Integer> mapEntry : outputMap.entrySet()) {
43              context.write(mapEntry.getKey(), new IntWritable(mapEntry.getValue()));
44          }
45      }
46  }
```

Hybrid approach - Reducer class

```
13  public class HybridReducer extends Reducer<WordPair, IntWritable, Text, Stripe> {
14      private HashMap<String, Integer> H;
15      private double total;
16      private String prev;
17
18      @Override
19      protected void setup(Context context) throws IOException, InterruptedException {
20          super.setup(context);
21          H = new HashMap<String, Integer>();
22          prev = null;
23          total = 0;
24      }
25
26      @Override
27      protected void reduce(WordPair pair, Iterable<IntWritable> counts, Context context) throws IOException, InterruptedException {
28          String w = pair.getWord().toString();
29          String u = pair.getNeighbor().toString();
30          if (prev != null && !prev.equals(w)) {
31              context.write(new Text(prev), getUpdatedStripe());
32              H = new HashMap<String, Integer>();
33              total = 0;
34          }
35          int sum = sum(counts);
36          total += sum;
37          H.put(u, sum);
38          prev = w;
39      }
40
41      @Override
42      protected void cleanup(Context context) throws IOException, InterruptedException {
43          super.cleanup(context);
44          context.write(new Text(prev), getUpdatedStripe());
45      }
```

Hybrid approach - Reducer class

```
46      private Stripe getUpdatedStripe() {
47          Stripe stripe = new Stripe();
48          DecimalFormat df = new DecimalFormat("#.###");
49          double frequency = 0.0;
50          for (Entry<String, Integer> entry : H.entrySet()) {
51              frequency = entry.getValue() / total;
52              stripe.put(new Text(entry.getKey()),
53                      new DoubleWritable(Double.valueOf(df.format(frequency))));
54          }
55          return stripe;
56      }
57      private int sum(Iterable<IntWritable> values) {
58          int sum = 0;
59          for (IntWritable intWritable : values) {
60              sum += intWritable.get();
61          }
62          return sum;
63      }
64  }
```

Hybrid approach - Output

```
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/hybrid/output/1/*

A10     { B12: 0.5, D76: 0.5 }

A12     { B76: 0.105, A10: 0.105, B12: 0.263, D76: 0.368, C31: 0.158 }

B12     { B76: 0.087, A12: 0.217, A10: 0.087, D76: 0.391, C31: 0.217 }

B76     { A10: 0.167, B12: 0.333, D76: 0.333, C31: 0.167 }

C31     { B76: 0.08, A12: 0.2, A10: 0.08, B12: 0.24, D76: 0.4 }

D76     { B76: 0.111, A12: 0.278, A10: 0.111, B12: 0.333, C31: 0.167 }
```

Comparison

Metrics	Algorithms of computing Relative Frequencies		
	Pairs approach	Stripes Approach	Hybrid Approach
Map input records	2	2	2
Map output records	128	22	53
Reduce input groups	32	6	26
Reduce input records	128	22	52
Reduce output records	26	6	6
CPU time spent (ms)	2480	2360	2400
Physical memory (bytes) snapshot	368 214 016	433 135 616	468 152 320

Analysis

Resource usage	Algorithms of computing Relative Frequencies		
	Pairs approach	Stripes Approach	Hybrid Approach
Network	Worst	Best	
CPU	Worst	Best	
Memory	Best		Worst

Analyze apache access logs

- Find IP addresses that accessed the server more than 10 times
- Calculate statistics based on the content size
- Count Response Codes
- Top 10 Endpoints

Find IP addresses that accessed the server > 10 times

```
    val ipAddresses: Array[String] = accessLogs
       .map(_.ipAddress -> 1L)
       .reduceByKey(_ + _)
       .filter(_._2 > 10)
       .map(_._1)
```

IPAddresses > 10 times:

[lhr003a.dhl.com, 207.195.59.160,10.0.0.153, prxint-sxb3.e-i.net, cr020r01-3.sac.overture.com, 64.242.88.10, ip68-22
8-43-49.tc.ph.cox.net,ogw.netinfo.bg, 200-55-104-193.dsl.prima.net.ar, pc3-registry-stockholm.telia.net,
ts04-ip92.hevanet.com,market-mail.panduit.com, 216-160-111-121.tukw.qwest.net, 195.246.13.119, proxy0.haifa.ac.il, ts05-ip44.hevanet.com,
mail.geovariances.fr, p213.54.168.132.tisdip.tiscali.de, 128.227.88.79, ns.wtbts.org,208-38-57-205.ip.cal.radiant.net,
,212.92.37.62,203.147.138.233, h24-71-236-129.ca.shawcable.net,h24-70-69-74.ca.shawcable.net]

Calculate statistics based on the content size

```
val contentSizes: RDD[Long] = accessLogs
   .map(_.contentSize).cache()

println("# Content Size Avg: %s, Min: %s, Max: %s".format(
   contentSizes.reduce(_ + _) / contentSizes.count,
   contentSizes.min,
   contentSizes.max))
```

```
# Content Size Avg: 7078, Min: 143, Max: 138789
```

Count Response Codes

```
val responseCodeToCount: Array[(Int, Long)] = accessLogs
   .map(_.responseCode -> 1L)
   .reduceByKey(_ + _)
```

```
# Response code counts: [(404,5),(401,123),(200,1273),(302,6)]
```

Top 10 Endpoints

```
val top10Endpoints: Array[(String, Long)] = accessLogs
   .map(_.endpoint -> 1L)
   .reduceByKey(_ + _)
   .top(10)(Ordering.by[(String, Long), Long](_._2))
```

```
# Top 10 Endpoints:
[(/twiki/bin/view/Main/WebHome,41),(/twiki/pub/TWiki/TWikiLogos/twikiRobot46x50.gif,32),(/,31),(/favicon.ico,28),(/robots.txt,27),(/razor.html,23),(/twiki/bin/view/Main/SpamAssassinTaggingOnly,18),(/twiki/bin/view/Main/SpamAssassinAndPostFix,17),(/cgi-bin/mailgraph.cgi/mailgraph_2.png,16)]
```

Resources and Links used in the Project

Cloudera tutorials:

- https://www.cloudera.com/documentation/enterprise/5-5-x/topics/spark_develop_run.html
- https://blog.cloudera.com/blog/2014/04/how-to-run-a-simple-apache-spark-app-in-cdh-5/

Databricks references:

- https://www.gitbook.com/book/databricks/databricks-spark-reference-applications/details
- https://github.com/databricks/reference-apps

Project report on GitHub:

- https://github.com/ganijon/bigdata/blob/master/projects/Report.pdf

Project presentation on GitHub:

- https://github.com/ganijon/bigdata/blob/master/projects/Presentation.pdf

Source code on GitHub:

- https://github.com/ganijon/bigdata/tree/master/projects

Q&A

The universe is transformation: life is opinion.

~ Marcus Aurelius