CS5560 Knowledge Discovery and Management

Spark MapReduce Programing

PS2-B

6/12/2017

Class ID: Megha Nagabhushan

Name: 15

Spark MapReduce Programming – Calculate everyone's common friends for Facebook

Facebook has a list of friends (note that friends are a bi-directional thing on Facebook. If I'm your friend, you're mine). They also have lots of disk space and they serve hundreds of millions of requests everyday. They've decided to pre-compute calculations when they can to reduce the processing time of requests. One common processing request is the "You and Joe have 230 friends in common" feature. When you visit someone's profile, you see a list of friends that you have in common. We're going to use MapReduce so that we can calculate everyone's common friends once a day and store those results. Later on it's just a quick lookup. We've got lots of disk, it's cheap.

1) Draw a MapReduce diagram similar to the word count diagram below.
2) Sketch a MapReduce algorithm for the common Facebook friends (referring to the word count code below).
3) Sketch Spark Scala implementation (referring to the word count code below).

Example

Assume the friends are stored as Person->[List of Friends], our friends list is then:

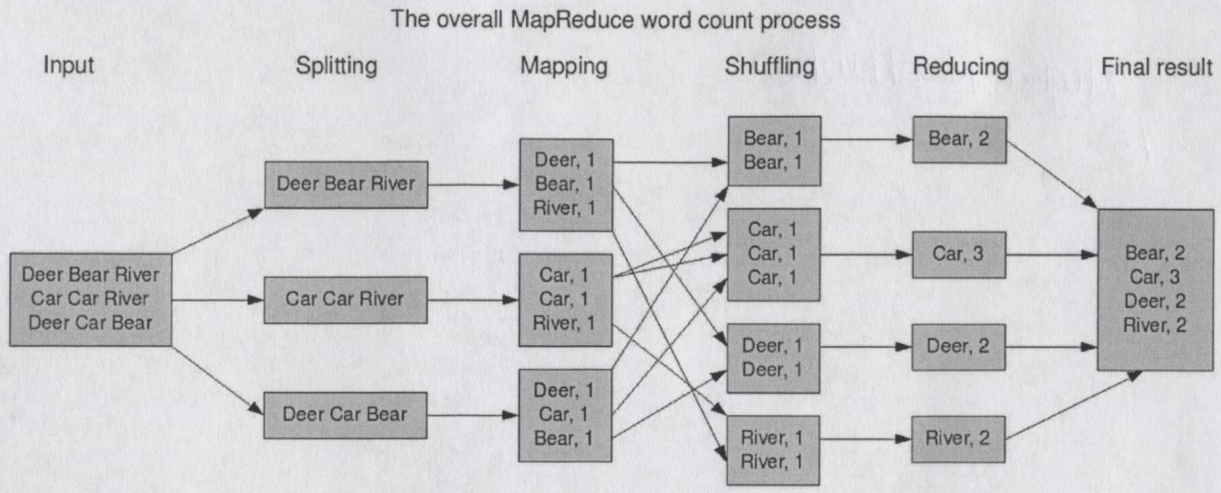
A -> B C D
B -> A C D E
C -> A B D E
D -> A B C E
E -> B C D

The result after reduction is:

(A B) -> (C D)
(A C) -> (B D)
(A D) -> (B C)
(B C) -> (A D E)
(B D) -> (A C E)
(B E) -> (C D)
(C D) -> (A B E)
(C E) -> (B D)
(D E) -> (B C)

Now when D visits B's profile, we can quickly look up (B D) and see that they have three friends in common, (A C E).

WORD COUNT EXAMPLE



Algorithm 2.1 Word count

The mapper emits an intermediate key-value pair for each word in a document. The reducer sums up all counts for each word.

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term $t \in$ doc d do
4:             Emit(term t, count 1)

1: class Reducer
2:     method Reduce(term t, counts $[c_1, c_2, \ldots]$)
3:         $sum \leftarrow 0$
4:         for all count $c \in$ counts $[c_1, c_2, \ldots]$ do
5:             $sum \leftarrow sum + c$
6:         Emit(term t, count sum)
```

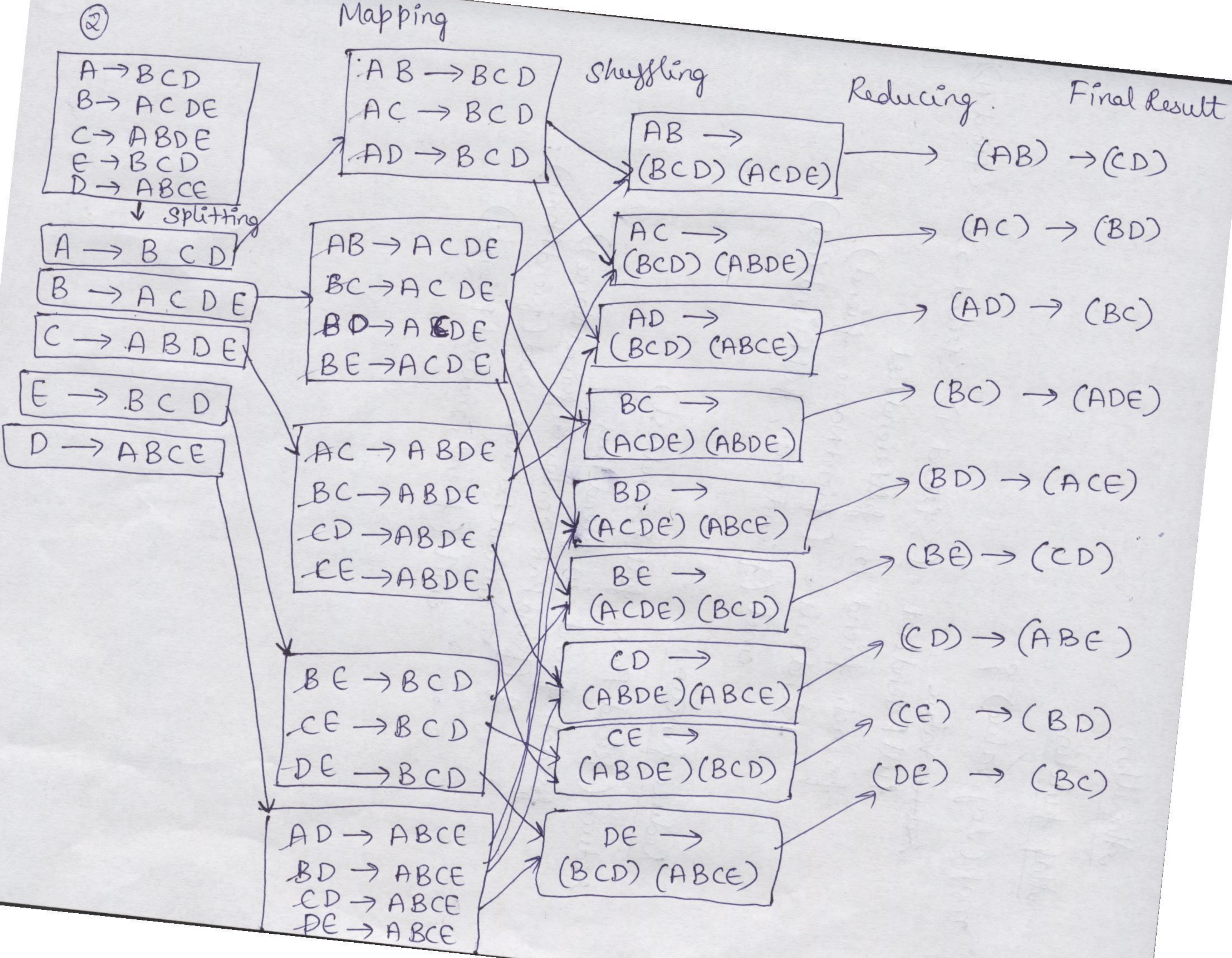
MapReduce Scala Code for WordCount

```
// This class performs the map operation, translating raw input into the key-value
// pairs we will feed into our reduce operation.
class TokenizerMapper extends Mapper[Object,Text,Text,IntWritable] {
  val one = new IntWritable(1)
  val word = new Text

  override
  def map(key:Object, value:Text, context:Mapper[Object,Text,Text,IntWritable]#Context) = {
    for (t <-  value.toString().split("\\s")) {
      word.set(t)
      context.write(word, one)
    }
  }
}

// This class performs the reduce operation, iterating over the key-value pairs
// produced by our map operation to produce a result. In this case we just
// calculate a simple total for each word seen.
class IntSumReducer extends Reducer[Text,IntWritable,Text,IntWritable] {
  override
  def reduce(key:Text, values:java.lang.Iterable[IntWritable], context:Reducer[Text,IntWritable,Text,IntWritable]#Context) = {
    val sum = values.foldLeft(0) { (t,i) => t + i.get }
    context.write(key, new IntWritable(sum))
  }
}
```

Spark Scala Code for WordCount

```
val textFile = spark.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).
reduceByKey(*func*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.



②
A → BCD
B → ACDE
C → ABDE
E → BCD
D → ABCE
Splitting
A → BCD
B → ACDE
C → ABDE
E → BCD
D → ABCE
Mapping
AB → BCD
AC → BCD
AD → BCD
AB → ACDE
BC → ACDE
BD → ACDE
BE → ACDE
AC → ABDE
BC → ABDE
CD → ABDE
CE → ABDE
BE → BCD
CE → BCD
DE → BCD
AD → ABCE
BD → ABCE
CD → ABCE
DE → ABCE
Shuffling
AB → (BCD) (ACDE)
AC → (BCD) (ABDE)
AD → (BCD) (ABCE)
BC → (ACDE) (ABDE)
BD → (ACDE) (ABCE)
BE → (ACDE) (BCD)
CD → (ABDE)(ABCE)
CE → (ABDE)(BCD)
DE → (BCD) (ABCE)
Reducing
Final Result
(AB) → (CD)
(AC) → (BD)
(AD) → (BC)
(BC) → (ADE)
(BD) → (ACE)
(BE) → (CD)
(CD) → (ABE)
(CE) → (BD)
(DE) → (BC)

Algorithm

Map function

```
map(key, value) {
    ~~reducer Value~~
    fullFriendList = (<friend-1> <friend-2> ... <friend_N>);
    foreach friend in fullFriendList
        friendPair = (friendName, friend)
            emit(friendPair, fullFriendList)
```

Reduce function

```
reduce(friendPair, fullFriendList)
    foreach friend in friendPair.
        If (friendPair contains(friendName))
            mutualFriends.add(friendName);
        emit(friend, mutualFriends).
```

Mutual Friends Code.

```
public class MutualFriendMapper {

  public void map(LongWritable key, Text value,
     OutputCollector <FriendPair, Text> output, Reporter
     reporter) throws IOException {

    String fullFrndList = value.toString(). split("->")
            ~~MutualFriends. NAME_LIST_SPLIT)~~;
    String frndName = fullFrndList [0];
    String [] listOfFrnds = fullFrndList [1]. split("->");
      for (String frnd : listOfFrnds) {
        . FriendPair fp = new FriendPair(new
            Text(frndName), new Text(frnd));
        output. collect (fp, new Text(fullFrndList[1]));

      }

  }

}
```

```
public class MutualFriends Reducer {

  public void reduce(FriendPair key, Iterator <Text>
   values, OutputCollector <NullWritable, Text> output,
       Reporter reporter) {

HashSet<String> rec = new HashSet <String>();
 List<String> mutualFends = new ArrayList <String>();
String fends = values. next. toString ();
   String [] names = fends. split(" → ");
   for (String name: names) {
        rec. add (name);

   }
   if (values. hasNext()) {
          fends = values. next (). toString ();
       names = fends. split ("→");
         for (String name: name) {
               if (rec. contains (name)) {
                    mutual Fends .add (name);

         }
      }
   }
if (mutualFrnds. size() >0) {
    String val = key.toString()+ "->" + " ";
    boolean commaflg = false;
    for (String mutualFrnd : mutualFrnds) {
        if (commaflg) {
            val += ", ";
        }
        val + = mutualFrnd;
        commaflg = true;
    }
    output. collect (NullWritable . get(), new Text(val))
                                                          ;
  }
 }
}
```