COMP9313: Big Data Management

Lecturer: Xin Cao

Course web site: http://www.cse.unsw.edu.au/~cs9313/

Question 1 MapReduce

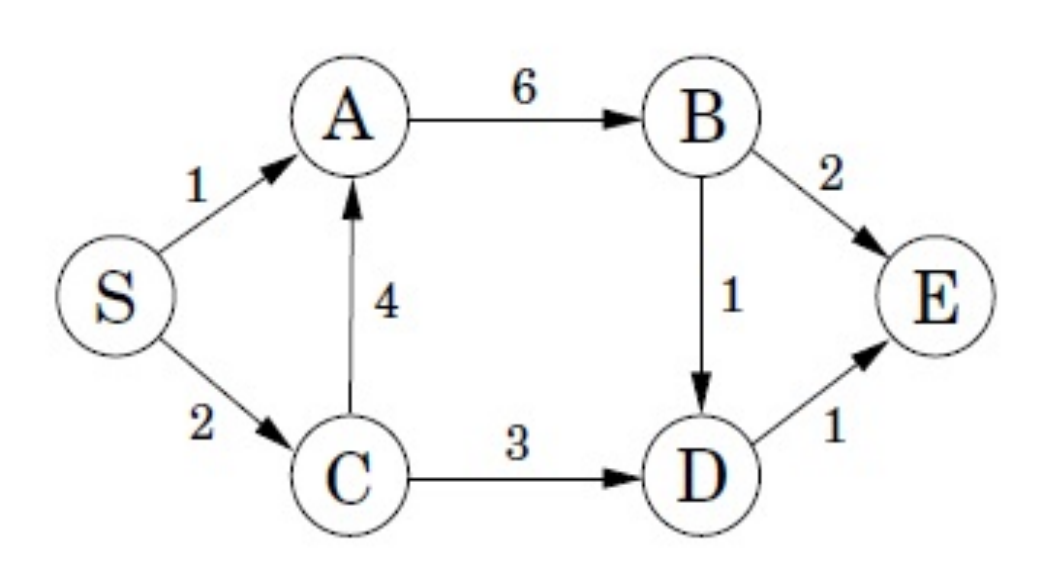
- Assume that you are given a data set crawled from a location-based social network, in which each line of the data is in format of (userID, a list of locations the user has visited <loc1, loc2, …>). Your task is to compute for each location the set of users who have visited it, and the users are sorted in ascending order according to their IDs.

Solution

```
class Pair
        userID, locID
        int compareTo(Pair p)
                int ret = this.locID.compareTo(p.getLoc)
                if(ret == 0) ret = this.userID.compareTo(p.getUser)
                return ret

class Mapper
        method Map(userID, list of locations)
                foreach loc in the list of locations
                        Emit( (loc, userID), userID)

class Partitioner
                method int getPartition(key, value, int numPartitions)
                        return key.first.hashCode() & Integer.MAX_VALUE % numPartitions

Class PairGroupingComparator extends WritableComparator
                method int compare(WritableComparable wc1, WritableComparable wc2)
                        return ((Pair) wc1).getLoc().compareTo(((Pair) wc1).getLoc())

class Reducer
                method Reduce(key, userList [])
                        Emit(key.getLoc(), userList)
```

Question 1 MapReduce

- Given the following graph, assume that you are using the single shortest path algorithm to compute the shortest path from node S to node E. Show the output of the mapper (sorted results of all mappers) and the reducer (only one reducer used) in each iteration (including both the distances and the paths).



Solution

1.
Mapper:
(A, 1), (C, 2)
Reducer:
A: 1 | S->A | B:6
C: 2 | S->C | A:4, D:3

2.
Mapper:
(B, 7), (A, 6), (D, 5)
Reducer:
B: 7 | S->A->B | D:1, E:2
D: 5 | S->C->D | E:1

3.
Mapper:
(E, 9), (D, 8), (E, 6)
Reducer:
E: 6 |S->C->D->E | empty

Algorithm terminates

Question 1 MapReduce

- Assume that in an online shopping system, a huge log file stores the information of each transaction. Each line of the log is in format of "userID\tproduct\t price\t time". Your task is to use MapReduce to find out the top-5 most expensive products purchased by each user in 2016.

```
class Mapper
        initialize an associate array H(integer UserID, priority queue Q of log record
based on price)
        method Map(key, log record R)
                if R.time == 2016
                        H(R.userID).add(R)
                        if(H(R.userID).size >5)
                                H(R.userID).remove(first element)
        method CleanUp()
                foreach(entry E in H)
                        Emit(E.userID, E.Q)

class Combiner ???

class Reducer
        method Reduce(userID, list of queues[])
                P <- get top 5 products from the list of queues
                Emit(userID, P)
```

Question 1 MapReduce

- Given a large text file, find the top-k words that appear the most frequently.

Answer: Do word count first, and then in each reducer, output only the top-k words. In the second round map/reduce task, read the local top-k and compute the global top-k.

Question 2 Spark

- Write down the output

a) val lines = sc.parallelize(List("hello world", "this is a scala program", "to create a pair RDD", "in spark"))

```
val pairs = lines.map(x => (x.split(" ")(0), x))
pairs.filter {case (key, value) => key.length <3}.foreach(println)
```

Output: (“to”, “to create a pair RDD”) (“in”, “in spark”)

b) val pairs = sc.parallelize(List((1, 2), (3, 4), (3, 9), (4,2)))

```
val pairs1 = pairs.mapValues(x=>(x, 1)).reduceByKey((x,y) => (x._1 + y._1,  x._2+y._2)).mapValues(x=>x._2/x._1)
pairs1.foreach(println)
```

Output: (1, 0) (3, 0) (4, 0) (because no “.toDouble” used)

Question 2 Spark

- Given a large text file, your task is to find out the top-k most frequent co-occurring term pairs. The co-occurrence of (w, u) is defined as: u and w appear in the same line (this also means that (w, u) and (u, w) are treated equally). Your Spark program should generate a list of ***k*** key-value pairs ranked in descending order according to the frequencies, where the keys are the pair of terms and the values are the co-occurring frequencies (**Hint:** you need to define a function which takes an array of terms as input and generate all possible pairs).

```
val textFile = sc.textFile(inputFile)
val words = textFile.map(_.split(" ").toLowerCase)

// fill your code here, and store the result in a pair RDD topk

topk.foreach(x => println(x._1, x._2))
```

Solution

```
def pairGen(wordArray: Array[String]) : ArrayBuffer[(String, Int)] = {
   val abuf = new ArrayBuffer[(String, Int)]

   for(i <- 0 to wordArray.length -1){
     val term1 = wordArray(i)
     if(term1.length()>0){
       for(j <- i+1 to wordArray.length - 1){
         val term2 = wordArray(j)
         if(term2.length()>0){
           if(term1 < term2){abuf.+=:(term1 + "," + term2, 1)}
           else {abuf.+=:(term2 + "," + term1, 1)}
         }
       }
     }
   }
   return abuf
 }


 val textFile = sc.textFile(inputFile)
 val words = textFile.map(_.split(“ “).toLowerCase)

 val pairs = words.flatMap(x => pairGen(x)).reduceByKey(_+_)
 val topk = pairs.map(_.swap).sortByKey(false).take(20).map(_.swap)

 topk.foreach(x => println(x._1, x._2))
```

Question 3 Finding Similar Items

- k-Shingles:

Consider two documents A and B. Each document's number of token is O(n). What is the runtime complexity of computing A and B's k-shingle resemblance (using Jaccard similarity)? Assume that comparison of two k-shingles to assess their equivalence is O(k). Express your answer in terms of n and k.

Answer:

Assuming $n >> k$,

Time to create shingles = $O(n)$

Time to find intersection (using brute force algorithm) = $O(kn^2)$

Time to find union = $O(n)$

Total time = (kn^2)

Question 3 Finding Similar Items

- MinHash:

We want to compute min-hash signature for two columns, C_1 and C_2 using two pseudo-random permutations of columns using the following function:

Row	C_1	C_2
0	0	1
1	1	0
2	0	1
3	0	0
4	1	1
5	1	1
6	1	0

$h_1(n) = 3n + 2 \bmod 7$

$h_2(n) = 2n - 1 \bmod 7$

Here, n is the row number in original ordering. Instead of explicitly reordering the columns for each hash function, we use the implementation discussed in class, in which we read each data in a column once in a sequential order, and update the min hash signatures as we pass through them.

Complete the steps of the algorithm and give the resulting signatures for C_1 and C_2.

Solution

	Sig1	Sig2
h1(1) = 1	∞	∞
h2(1) = 3	∞	∞
h1(0) = 1	∞	2
h2(0) = 3	∞	6
h1(1) = 5	5	2
h2(1) = 1	1	6
h1(2) = 1	5	1
h2(2) = 3	1	3
h1(4) = 0	0	0
h2(4) = 0	0	0

Question 3 Finding Similar Items

- LSH:

Suppose we wish to find similar sets, and we do so by minhashing the sets 10 times and then applying locality-sensitive hashing using 5 bands of 2 rows (minhash values) each. If two sets had Jaccard similarity 0.6, what is the probability that they will be identified in the locality-sensitive hashing as candidates (i.e. they hash at least once to the same bucket)? You may assume that there are no coincidences, where two unequal values hash to the same bucket. A correct expression is sufficient: you need not give the actual number.

Answer: $1-(1-0.6^2)^5=0.893$

Question 4 Mining Data Streams

- DGIM

Explain the space complexity of maintaining one bucket in the DGIM algorithm in terms of the window size N. How many buckets you need to store at most?

Answer:

a). The timestamp of its end [$O(\log N)$ bits]

b). The number of 1s between its beginning and end [$(\log\log N)$ bits]

Reason of $(\log\log N)$: N bits, each bucket contains at most $(1/2N)$ 1s, and thus log(N) bits can store this value. The number of 1s are orders of 2, and thus can only store the orders rather than the original number, and hence $(\log\log N)$.

c). At most $\log N$ buckets

Question 4 Mining Data Streams

- DGIM

Suppose we are maintaining a count of 1s using the DGIM method. We represent a bucket by (i, t), where i is the number of 1s in the bucket and t is the bucket timestamp (time of the most recent 1).

Consider that the current time is 200, window size is 60, and the current list of buckets is: (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (1, 197) (1, 200). At the next ten clocks, 201 through 210, the stream has 0101010101. What will the sequence of buckets be at the end of these ten inputs?

Solution

There are 5 1s in the stream. Each one will update to windows to be: [each step 2 marks]

- (1) (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(1, 197)(1, 200), (1, 202)

=> (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (1, 202)

- (2) (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (1, 202), (1, 204)
- (3) (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (1, 202), (1, 204), (1; 206)

=> (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (2, 204), (1, 206)

=> (16, 148)(8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206)

- (4) Windows Size is 60, so (16,148) should be dropped.

(16, 148)(8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206), (1, 208) => (8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206), (1, 208)

- (5) (8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206), (1, 208), (1, 210)

=> (8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (2, 208), (1, 210)

Question 5 Recommender Systems

- Consider three users u_1, u_2, and u_3, and four movies m_1, m_2, m_3, and m_4. The users rated the movies using a 4-point scale: -1: bad, 1: fair, 2: good, and 3: great. A rating of 0 means that the user did not rate the movie. The three users' ratings for the four movies are: u_1 = (3, 0, 0, -1), u_2 = (2, -1, 0, 3), u_3 = (3, 0, 3, 1)
  - Which user has more similar taste to u_1 based on cosine similarity, u_2 or u_3? Show detailed calculation process.
  - Answer: sim(u1, u2) = (3*2 -1*3)/(sqrt(10)*sqrt(14)) ≈ 0.2535, sim(u1, u3) = (3*3-1*1)/(sqrt(10)*sqrt(19)) ≈ 0.5804. Thus u3 is more similar to u1.
  - User u_1 has not yet watched movies m_2 and m_3. Which movie(s) are you going to recommend to user u_1, based on the user-based collaborative filtering approach? Justify your answer.
  - Answer: You can use either cosine similarity or Pearson correlation coefficient to compute the similarities between users. However, the conclusion should be that only m3 is recommended to u1.