CMPSCI590D: Algorithms for Data Science

Homework 2

Instructor: David Wemhoener — Posted: Oct 6th, Due: Oct 16th at 4 pm

1 Question 1

We have n bits of main memory and m items in the set S.

Case 1. We divide the n bits of main memory into k arrays $A_1, A_2, .., A_k$ and hash once in each array, say using hash functions $h_1, h_2, .., h_k$ respectively.

In this case, for each array, probability a particular array location has a 0 bit is $\left(1 - \frac{1}{n/k}\right)^m = \left(1 - \frac{k}{n}\right)^m$, since we are hashing m items in each array and probability of hashing a particular item in a given location is $\frac{k}{n}$.

Thus, the probability that a particular array location has a bit 1 is $1 - \left(1 - \frac{k}{n}\right)^m$.

Now, given an item x which is not in set S, we falsely decide its membership to be positive if

$$Pr(A[h_1(x)] = 1 \text{ and } A[h_2(x)] = 1 \text{ and } A[h_3(x)] = 1.... \text{ and } A[h_k(x)] = 1) = \left(1 - \left(1 - \frac{k}{n}\right)^m\right)^k$$

Case 2. We hash in a single array of size n using k hash functions. In that case, as shown in the class, the false positive probability is

$$\left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^k$$

Now, $\left(1 - \frac{1}{n}\right)^k \geq 1 - \frac{k}{n}$, by simply expanding the series and noting that $(1 - x)^a \geq 1 - ax$, $x > 0, a \geq 1$. Thus, the false positive probability in Case 2 is smaller than for Case 1. This difference is however not much if n and m are large.

2 Question 2

There are other valid ways to approach some of these within the mapreduce framework. These are just examples of approaches one could take:

2.1 Part A The largest integer

Map outputs the int values directly, with a default key since all values will go to a single reducer
Combine takes the values for that map task and calculates the max, which it returns
Reduce takes all the values and calculates max, which it returns

2.2 Part B The average of all the integers

Map outputs the int values directly, with a default key since all values will go to a single reducer
Combine sums all the input values, then returns a default key with pair of values (partial sum, count of input values)
The single reduce receives a collection of (partial sum, input values) in a list. To calculate the average, it calculates the sum of all the partial sums, and then divides this by the sum of the input value counts.

2.3 Part C The average of all the integers

For each int in the list, map emits a tuple with the int as the key and a value of 1 Reduce takes a key and list of 1s as input and returns the key (thus from all the reduces we get a unique list of the integers in the original list)

2.4 Part D The count of the number of distinct integers in the input

For each int in the list, map emits a tuple with the int as the key and a value of 1 Reduce takes a key and list of 1s as input and returns as a value 1 (so we get 1 for each distinct int) 2nd reduce returns the the sum of all the inputs

3 Question 3

3.1 Part A

In the first case, no, as the number of reducers should be high relative to the number of tasks, and the end result should be that the average run time is similar. In the second case, we would expect the skew to be significant.

3.2 Part B

No, using a combiner will collapse the number of tuples output from each map task to one, which will significantly reduce the gap between the number of tuples for common words like the versus uncommon words.

4 Question 4

As in question 2, there are other valid ways to approach some of these within the mapreduce framework. The pseudocode are just examples of approaches one could take:

4.1 Number of nodes in the graph

265,214

```
map(edge of form (u,v))
    emit a pair of tuples (1,u) (1,v)

Combine(tuples of form (1,u))
    emit set of all the values (so there will be one entry for each unique node), v

Reduce(key, [list1, list2,...]):
    combines them to create a single merged set
    returns the length of this set
```

4.2 Average (and median) indegree and out degree

Avg in: 1.58
Avg out: 1.58

For indegree:

```
Map(edges of the format (sender, receiver)):
    emit (receiver, 1)

Reducer1(receiver, value list):
    emit 'count', sum value list

Reducer2(key, value list):
   emit average(value list)
   emit median(value list)
```

The approach for outdegree is the same

4.3 Average (and median) number of nodes reachable in two hops

Avg: 174.47
Med: 72

```
Map(edges of the format (sender, receiver)):
    emit receiver,('in', sender)
    emit sender,('out',receiver)

Reducer1(node u, value list):
    outgoing = all node from values where value[0] = 'out'
    incoming = all node from values where value[0] = 'in'
    //for every node that goes into this node
```

```
    //emit the set of nodes known to be receivers of this node
    for v in incoming:
        emit v, outgoing
    emit u, []

Reducer2(key, value list)
    s = merge all node lists into a single set
    emit 'count', len(s)

Reducer3(key, value list):
   emit average(value list)
   emit median(value list)
```

4.4 Number of nodes with indegree > 100

702

```
Map(edges of the format (sender, receiver)):
    emit (receiver, 1)

Reducer1(receiver, value list):
    emit 'count', sum value list

Reducer2(key, value list):
    s = value list remove values with size >= 100
    emit len(s)
```

5 Question 5

X_t =time taken to complete a job which needs t seconds
$X_t = p(X_t + 10t) + (1-p)X_{t-1}$
$(1-p)(X_t - X_{t-1}) = 10pt$
Adding the telescopic sum we get:
$(1-p)X_t = \sum_{i=1}^{t} p_i$
$X_t = \frac{10pt(1+t)}{2(1-p)}$
For n tasks expected time:
$n * (\frac{10pt(1+t)}{2(1-p)})$