590D - HW2

1. Krishna Prasad Sankaranarayanan
2. Vinayak Mathur

1. Dividing into k arrays. ∴ k arrays ⇒ size n/k

Function of n, m & k $f(n, m, k)$

∴ False probability for an array size ⇒ n/k

$$\Rightarrow 1 - e^{-m/n/k}$$

∴ For k arrays ⇒ (probability)^k.

$$\Rightarrow \left(1 - e^{-\frac{m}{n/k}}\right)^k \quad \text{—— (1)}$$

False probability for k hash functions for one array ⇒ $\left(1 - e^{-\frac{m}{n/k}}\right)^k$ — (2)

From (1) & (2) both k hash functions and dividing arrays have the same false +ve probability

5. Each job contains n tasks, and t seconds make a task.

Total execution calculation ⇒ nt seconds

Probability of task failing ⇒ p

Restart will increase the team by 10t seconds

$P(\text{failure}) = p$

$p(\text{success}) = 1-p$

After each failure, there is 10t seconds restart.

$\Rightarrow$ time : $t((1-p)10t + p)$

$\Rightarrow pt + 10t^2 - 10pt^2$

For all n - nodes

$\Rightarrow (pt + 10t^2 - 10pt^2) \cdot n$

The execution time (operation time) per second

$\Rightarrow p^n + (1-p)^n 10t$

each time, system fails $\Rightarrow$ we need 10t seconds to restart

$\therefore$ Total time $\Rightarrow t p^n + 10t(1-p)^n$

If For all nodes $\Rightarrow n$

Total time $\Rightarrow n(tp^n + 10t(1-p)^n)$

$\Rightarrow ntp^n + 10nt(1-p)^n$

3

1. Increasing the number of reduce tasks drastically increases the time framework overhead and hence skewness

On the other hand, it lowers the cost of failure by increasing load balancing

Combining the reducers thereby decreasing no of reduce tasks can sometimes serve a better result by reducing the overhead

Too ~~many~~ few reducers can cause undue load on the node leading to inefficient failure recovery mechanisms.

Too many reducers affects the combiner shuffle pipeline crossbar.

Hence, skew occurs when data is not evenly partitioned across tasks

For 10,000 reduce task - high skew

" 10 reduce tasks - lower skew

2. If we use a combiner after 100 maps, it will facilitate efficient data partitioning among reduce tasks. Hence equal amt of

work task would be done by each future reduce task

Hence skew would not rise. Data partioning would get benefitted from a combiner.

2 a) Largest Integer

```
def map (self, '_', numbers)
    local_max = INTEGER-MIN
    For x in numbers
        if x > local-max
            local-max = x
    yield ("Local Max", local-max)

def reduce (self, key, value)
    total_max = INTEGER_MIN
    for x in value
        if x > total_max
            max_total = x
    yield ("Total Max is", max_total
```

b) Average of integer

```
def map ( self ,'-', numbers)
    #val - number - count = 0
    #avg - average
    for x in numbers
        yield ( 1, (val, avg))
        val++
def Reduce ( self, key, value)
        num += val * avg
        den += val          ans = num/den
        yield ("Average is ", ans)
```

c) Removing Duplicates

```
def map(self,'-', number)
    for x in numbers
        yield (x , 1)
def reduce (self , key, value):
        yield ("unique number", key)
```

d) def map-init (self) Use o/p of (c)

```
global count = 0
def map (self,'-', number)
    for i in numbers
        yield (1, i)
```

```
def reduce( self, key, value)
    l = [ ]
    l = next (value) # list of all distinct
                                    elements
p = len(l)
    yield ("No of Distinct Elements", p)
```

4) The answers obtained from the python scripts are listed below:

i) Total number of nodes in the graph = 265214

ii) Map:
Map each source node to the corresponding destination and each destination to the corresponding source

Combine:
Use the combiner to map all the nodes to a single key

Reduce:
Counting the number of nodes in value will give us the total number of nodes in the graph

ii) out Degree

The total out degree of the graph = 420045
median out degree = 1
average out degree = 1.5837

- map:
map each source node to the corresponding destination
- reduce 1:
Use the reducer to map each key to the length of the values in the list
- reduce 2:
Use the reducer to map all the lengths to a constant key
- reduce 3:
count the number of values in the combine list to get the total outdegree
- reduce 4: sort the outdegrees

•reduce 5:
Count the outdegrees in the value argument
•reduce 6:
Calculate the median using the standard formula
•reduce 7:
Divide sum by count to get the average

In degree :

The total In degree of the graph = 420045
Median In degree = 0
Average In degree = 1.5837

• Map:
Map each destination node to the corresponding source node.
• Rest of the steps remain same for as the outdegree calculations.

iii) 2 hops

Total number of nodes reachable in 2 hops = 50 610 762
Average number of nodes reachable in 2 hops = ~~1452~~
Median number of nodes reachable in 2 hops = 1

Logic:
For 2 hop neighbours to exist a common node should be present both in the source nodes & the destination nodes.
We use map reduce to find these common nodes.
For each occurence of common node 'k' in the source node there are 'l' possible pathways for 2 hop neighbours to exist, where l is the occurence of node 'k' in the destination node list

Therefore the total number of 2 hop neighbours =

Occurrence of common node in source nodes × Number of occurrence of common node in destination

Steps:

- map-init:
Have a global list which can store a combined list

- map:
map 1 to all nodes in the source and 2 to all nodes in the destination

- reduce 1:
Append all the source nodes in a list (s) and all the destination nodes in a list (d). Combine these lists in the global list and yield that as the value

- reduce 2:
Add the source and destination nodes to sets and take the intersection of these sets, to get the common elements.
Multiply the occurrences of these common elements in the source and destination to get the total number of 2 hops.

- reduce 3:
Calculate the average by dividing by total number of nodes and median using the standard formula.

iv) Number of nodes with indegree >100 = 5955

- Map:
Map destination nodes to source nodes

- Reduce 1:
Count the total length of values and yield the length as the new value

- Reduce 2:
For each destination node check if the value of length is greater than 100 and increment a counter

- Reduce 3:
Yield the count with indegree >100