Exam Solutions

Ahmed Alhasan ‘ahmal787’

2020-08-20

Q10

- (a)

by adding an additional key-value pair for each unique user name such that the key of such a keyvalue pair is the corresponding user name, prefixed with the string “likedby:”, and the value is an array of the people liked by that person.

“alice_in_se” → “Alice, 1987, [bob95 charlie]”

“bob95” → “Bob, 1995, [charlie]”

“charlie” → “Charlie, 1996, []”

“selaya” → “Alice, 1974, [charlie]”

“likedby:Alice” → “[bob95 charlie]”

“likedby:Bob” → “[charlie]”

“likedby:Charlie” → “[]”

“likedby:selaya” → “[charlie]”

now we can do get(“likedby:Alice”), which would give us the people liked by Alice (i.e., bob95 and charlie).

-(b)

this implementation will make it faster to retrieve people liked by some person and no requirement for the user to implement his own application to search the value

the downside however, we added redundency which means we need more storage for the data now

Q11

- A streaming application in many cases require a read scalability but not necessarily a data scalabality becuase there is huge amount of data that either too big to be stored or not important to have it on disk for long period of time, like a growing number of a CCTV cameras connected to a security system that monitors a city or a country.

Q12

- Wrong, the idea of consistent hashing is to reduce the number of nodes to be copied after node removal or addition, since each compute node is assigned a distinct range of possible hash values, then when adding or removing a node ONLY the range of hash values that was assigned to that node (in case of removal) will be reassigned to next node with the next range, and in case of addition the added node will take only the keys that are close to its range by a given weight

Q13

- in MapReduce the master sends heartbeats to worker nodes to check their liveness, if it finds out that certain worker is slow or not responding it will reassign the its task to another worker

Q14

- (i)

work is the total number of performed elementary operations w = K * N (reading from file) + 1 * N (converting to floating point) + 2 * N (adding and squaring) + 1 * 1 (taking the sqrt of the total) + 1 * 1 (print operation)

w = K * N + 3N + 2 = O(N)

- (ii)

the work will not change

t(P,N) = log(N)

Q15

- it is better to have short lineages becuase it helps with load balancing of transformation operations that happen in the map phase

Q16

- pipelining, because the streamed data comes in a fixed sized packets, pipelining applies a sequence of dependent computations/tasks element wise to the data sequence which make parallelizm possible

Q17

```
# Reading from file => reformat it using map() =>
# persist() or cache() for faster access
train = sc.textFile(...).map(...).cache()


# Compute logistic regression gradient for a matrix of data points
def gradient(matrix, w):
  Y = matrix[:, 0]  # Labels
  X = matrix[:, 1:] # Coordinates
  # For each point (x,y), compute gradient function, then sum these up
  return ((1.0/(1.0 + np.exp(-Y * X.dot(w))) - 1.0) * Y * X.T).sum(1)

nSamples = 100
counter = 1
while counter <= nSamples:
  # We take a sample without replacement of full size with seed = 1
  tr_sample = train.sample(False,1,1)
  # Intial random weights for each sample
  w = np.random.ranf(size = D)
  # calculate the weight of the current sample
  for i in range(iterations):
    w -= tr_sample.map(lambda m: gradient(m, w).reduce(lambda a,b: a+b))

  # add all weights together
  total_w += w
  counter += counter

# take the average
average_w = total_w / nSamples
print("average weight: " + str(average_w))
```