HW#7 CS35D Bikram. Maharjan

(a) ⇒ This is required for consistency of EBS volume data under all potential failure modes. Reading & writing at same time will appear stuck.

(b) Amazon's EBS is configured in such a way that if there is a long time-out period then it will create slow API calls to back up and consequentially resulting in thread starvation in the EBS control plane.

(c) In Amazon system, there is a leader (EBS control plane) which will police EC2 instances and EBS nodes with the volume data. EBS here says which data is real and which one is a replica of it. If failure occurs in node that EBS control plane is responsible to negotiate with EBS nodes.

(2) A → 1, 2, 3, 4
B → 1, 2
C → 1, 3
D → 2, 4

Mapper class:

Example

Mapper class will map out as:

A, B, C, D
1 2 3 4
C ⇔ A
Soc → B
Fist → C
V → D

Mapper output

(A, 1) (A, 2) (A, 3) (A, 4)
(B, 1) (B, 2)
(C, 1) (C, 3)
(D, 2) (D, 4)

Reducers

Collect by key:

for 2:
(2, A)
(2, B)
(2, D)

for 1
(1, A)
(1, B)
(1, C)

+ Reducers will pass in 21) after collecting:

+ {1, (A, B, C)}, {2, (A, B, C)}

outputs from reducer are as above

Passing the value into 2nd Mapper class:

output made by Mapper class II

{(1,2), {A, 3, 3}}

{(1,3), {A, 3, 2}}

{(1,4), {A, 3, 2}}

⋮

{2,4), {D, 3, 2}}

Reducer II

output

{(1,2), $\frac{2}{3+3-2}$}

fraction

{(A,C), $\frac{2}{3+3-2}$}

Assignment #

(3) (1) Java code submitted

(2) logic: Inorder to do this problem I used 2 mapper and 2 reducer class. 1 set of (Mapper & reducer) is similar to word-count [Mapper-reducer] class. Mapper sent in key value which is Text and other is IntWritable type with one as its value. Reducer here will count all the ones. This computable value is again sent to a mapper class. Here Inputs is (A, 146), (B, 20) etc. Here A is a node and 146 is number of connection I took tuple and send in (146, one) to 2nd reducer class. In this reducer class I counted frequency of set such as (x, ones). Repeating same process as before. Only

total x as word count example. In this way we will increment value with similar connection nodes.

(c) Pseudo-code

Mapper Class I:

```
fun map (key, values, context):
  + assigns values to iter and convert value to string.
    itr := new stringTokenizer (value.toString);
  +
    Loop (untill itr has values):
        word.set(itr.nextToken());
        Word.write(word, one);
    Ends
```

} loops and emits value to reducer I

Reducer Class I

```
fun Reduce (key, values, context):
    + Creates a variable that will hold total
      int sum
    + loop through iterable variable (value) and increment by one
      loop: (val: values)
          Sum += val.get()  // val has(one) value passed from mapper Class.
      Ends
    + casts int to IntWritable and sent to Mapper Class II
      result.Set(sum);
      context.write (key, result);
```

Mapper Class II

```
fun map( key, values, context):
    + get value from values
    StringTokenizer itr = new StringTokenizer (values.toString)
    Loop ( itr has some values):
```

+ Sets word and emit value to Reducer II
  title = itr.nextToken();
  num = itr.nextToken(); // ← This is what we want
+ Emitting to reducer class II
  word.set(num);
  context.set(word, one);

Reducer class II :

+ Create a variable that will hold total
  int sum
+ loop through iterable variable (value) and increment by one
  loop : (val : values)
    Sum += val.get() // val has (one) value passed from mapper Class.
  End
+ cast int to IntWritable and sent to Mapper Class II
  result.Set(sum);
  context.write(key, result);

④ ① Code Submitted

② Logic (K, Frequency) : logic here is very simple.

In mapper class, I created a method that checks whether a word has vowels or not. If it has vowel letters then collects the number of vowels and word and pass it to reducer It will look like (vowel_Counter, one). Reducer will collect this and count the frequency of vowel_Counter of particular size.

Ex : I am a Student.

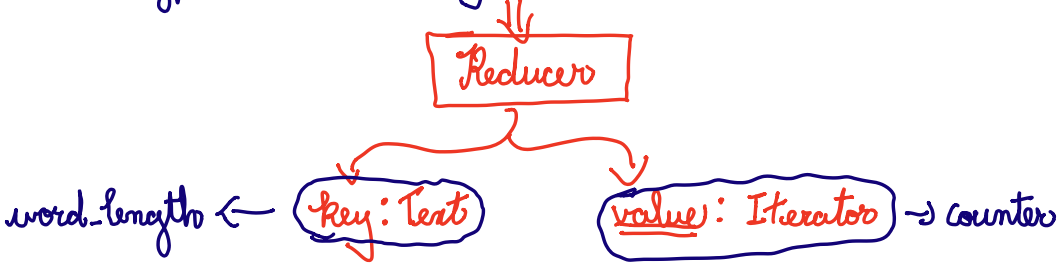
Mapper (1, one), 1, one

Reducer (1, 2) # frequency of 1 vowel length □

4 iii) Logic (n, k, fn): Atmost same as (i) but here we have to add fraction. Here in concatinated word_length and no of vowel in a word and passed to reduced. At reducer, I iterates through all the values and created the fraction { $\frac{\text{no \# of occurance of n-number}}{\text{Total}}$ } = Fraction

iii) Pseudo-Code:

Input data type ⇒ < Text key, Iterator <IntWritable> value):



pseudocode:

+ obtain value from Iterator

+ create a loop from 0 to counter :

+ Text ⇝ + Casts it as String
+ Create instance of new Text( )

+ text.set(String)

int index = 0;
for (index = 0; index < counter ; index ++) {

emits this ⟹ FORMAT ((word_counter, index), funx) value
to amazon clusters

Text key

encapsulation and cast as string type

+ funx (index, values)

+ define funx → double (fraction)

+ Inputs should be index and Iterator values.

+ Traverse through values

```
v_counter = 0;
While ( values.hasNext() ) {
    v_instance = value.Next();
    if ( index == v_instance)
    {
        v_counter++;
    }
} // end_of_while loop
```

+ To do ~> Cast it to <IntWritable>

refers to an object [not actual datatype]

+ add in the fraction:

```
double fraction = 0;
fraction = v_counter /(values.size() - 1)
```

+ finally return the fraction:

```
return fraction;
```

"This is a word" (4,1,one) (4,2,one) ⇒ (4,1, 1/2) (4,2, 1/2)

(4,1,one)
(2,1,one)
(2,1,one) ⟹ (2,1,one) (2,1,one) ⇒ (2,1, 2/3)
(4,2,one)
(2,0,one)

```
for ( i=0; i< word; i++)
Context.write ("word_size" " size", i)
```