CSE587: Project 2 Report

Zhiliang SU, #50027279

1. Implementation

a) WordCount

For word count program, I will just use the example code from Hadoop.

b) Pair relative co-occurrence

For pair relative co-occurrence program, I'll first use build in method ***"split"*** featuring with regular expression of "String" type to split input text. Since the input text read by Mapper will be split by UNIX style change line special character "\n", so that we can think that we are counting relative co-occurrence line by line. The mapper extends the Mapper class and override the map method in the class. The mapper will take Text as input value and emits <Text IntWritable> pairs as its outputs, where Text is a writable String type containing two co-occurrence keys.

Before emitting pairs to the reducer, in order to compute the normalize factor we have to make sure that all Text pair with same left Text key will be sent to a same reducer. To enable this feature, we have to customize our own partitioner by extending the Partitioner class and override the getPartition method. In getPartition method, we create a new method to assign task to reducer. We first define the number of reducers, which is 4 in this case. And then split the input key value and take the left key out. Compute the hash code of this left key and mod it by number of reducers and produce an integer output indicating the number of reducer we should assign current key-value pair to.

Then by extending the Reducer class and overriding the reduce method, we customized our own reducer. In order to compute the normalization factor, we should first not submitting the key-value pair we received and instead, store them in a map using key as the key in map and the value as the value the key mapping to and also stores the current left key. Once the reducer begins receiving different left key emitted by partitioner, first call a customized function named "EmitResult" and start outputting the key-value pairs stored in the map. After finished outputting all key-value pairs clear the map, the left key and the normalization factor and re-do this routine.

c) Stripe relative co-occurrence

The stripe method will be more straight-forward and sharing the same input text separation routine. By extending mapper and reducer class and overriding the map and reduce method in each class respectively and there is no need to modify the default HashPartitioner provided by Hadoop. All we have to do is to add a counter in reducer as the normalization factor. The only difficult point for stripe method is to get the plain-text output out of the MapWritable type. If we directly emit the MapWritable type to the context of reducer, we will get some text outputs that look like pointers instead of the contents in MapWritable. The reason is that MapWritable don't have a ***"toString"*** method. So the simplest way to do is to extend the MapWritable with a toString method outputting the string value of two writable type value in MapWritable.

Then use the extended class as the output value type.

2. Size v.s Performance Chart

I used Gutenberg data as test data and split the data into 3 different size, 5Mb, 10Mb and 50Mb of sub-sets. In job configure, 4 reducers will be used.

All three programs are packed into 1 jar file and use the argument to control which kind of task is to be used:

>> bin/hadoop jar ~/mrWordCooccurrence.jar [input-folder] [output-folder] [task]

Task can be either one of “wordcount”, “pair” or “stripe”.

After running on a single node machine and on different size of data subsets, I have the following size v.s. performance chart:

Task	***5MB***		***10MB***		***50MB***	
WordCount	01:13	01:06	01:37	01:24	07:34	07:01
Pair	02:11	01:03	04:39	03:20	14:37	17:27
Stripe	02:47	03:28	05:11	05:59	22:22	29:41