COMP9313 Project4 Report

Name: Bowen Zhou Id: z5127532

1. Main Steps

In this project, we are required to use Jaccard similarity function to compute the 'self-join' text file similarity and output the similar pairs together with their similarities, which are larger than the threshold.
To solving this problem, I use the following method:

(1) In the first step, I sort the token of each record by frequency. First, I use the Word-Count method to get the count of each token in this file. Then I build a dictionary to store the statistical results by using the **Map()** function. Next, I use **ArrayBuffer** when reading each line and save the token and the corresponding frequency in the dictionary. The format is like that:

(Record Id, [(token, frequency)…])

Then, I sort the token by their frequency. By finishing this step, it could support us to use the method of prefix filtering which mentioned in PPT.

(2) In the second step, when reading each record, we use the formula to compute the prefix length.

PrefixLen = RecordLength – math.ceil(RecordLength*Threshold) +1

Then we emit the token and record, the format is like (token, Record)

(3) In the third step, I use Join function to get all the possible pairs. Then, according to PPJoin method on the paper[1] and the requirement of the output format, I do some filter operations.

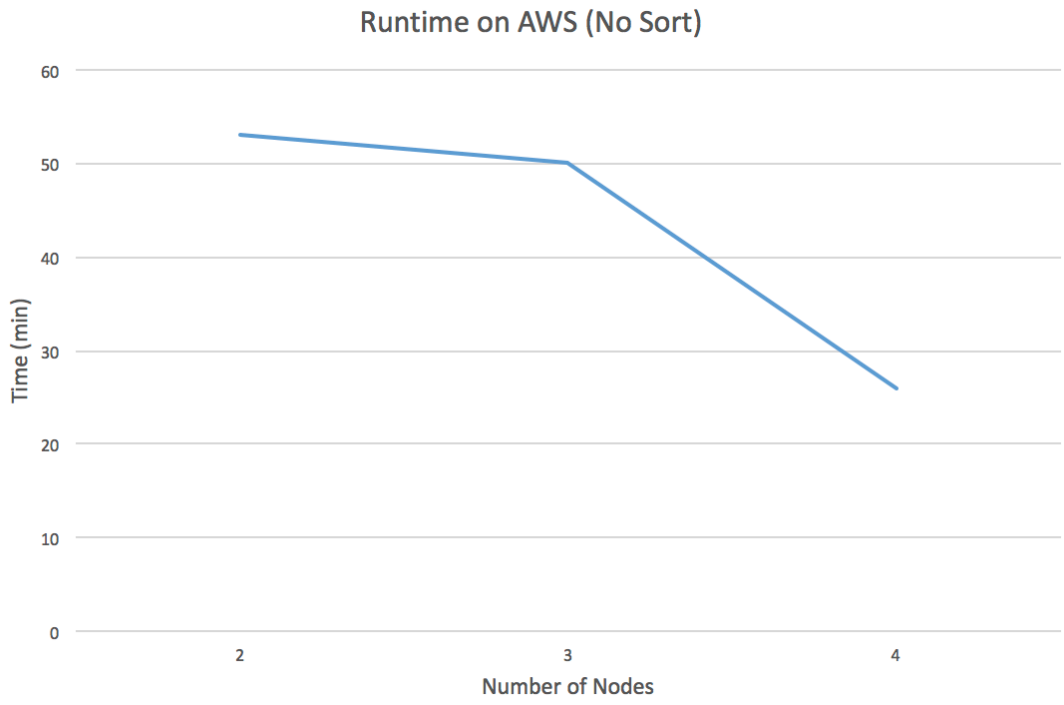
a. RecordID1 < RecordID2

b. $\max(Record\_a.len, Record\_b.len) \geq \frac{threshold}{1+threshold}(Record_a.len + Record\_b.len)$

After filtering, we use the Jaccard Similarity formula to calculate the similarities and out put the results which meet the requirement of the threshold.

2. Results Evaluation

The program has a good performance on my local machine for all the given small tests, but when running the program on AWS, I meet the problem for the long running time, after optimizing the program and do some experiments on AWS, I find that the ReduceByKey function will have a high time cost on the large test file when using it before filter operations. Thus, after some modifications, I get a good performance on AWS.



3. Reference

[1] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In WWW, pages 131–140, 2008