CSE587 – LAB 5

SPARK

May 13, 2017

University at Buffalo

The State University of New York

Name	UBIT Name	Person #
Shad Khan	shadulla	50208391
Amardeep Virdy	avirdy	50208831

Vignette

The Vignette for this Activity has been performed in Jupyter. There were specific steps that were followed to integrate pyspark with jupyter, most critical of which was adding Some enviorment configurations for pyspark in bashrc.

```
export SPARK_HOME=/spark-2.1.1-bin-hadoop2.7

export PATH=$SPARK_HOME/bin:$PATH

export PYSPARK_PYTHON=python3

export PYSPARK_DRIVER_PYTHON=jupyter

export PYSPARK_DRIVER_PYTHON_OPTS='notebook'
```

Following configuration were added in bashrc. Now, running the command pyspark starts a jupyter notebook which is capable of running pyspark. It is also worthy to note that the python version of your computer and the notebook should match too.

Vignette can be easily opened through Jupyter. However running the notebook requires your jupyter to be capable of running pyspark.

Activity:

The environment that we used for the activity was the environment provided to us in the spark virtual machine. Language chosen was Python.

2-Gram Word Co-occurrence

The code for this activity can easily be run by just running the following command

```
Spark-submit 2_gram.py
```

It should be noted that new_lemmatizer.csv file should be in the same location as that of the code. Also there should be a input folder in the same location as the code. All the files upon which processing has to be performed should be within this input folder.

After running the code, a folder with the name of output2 will be created containing the results of the processing.

The output received will be in the following format:

((Word1(or lemma1),Word2(or lemma2)), [‘<description1>’, ‘<description2>’])

for example:

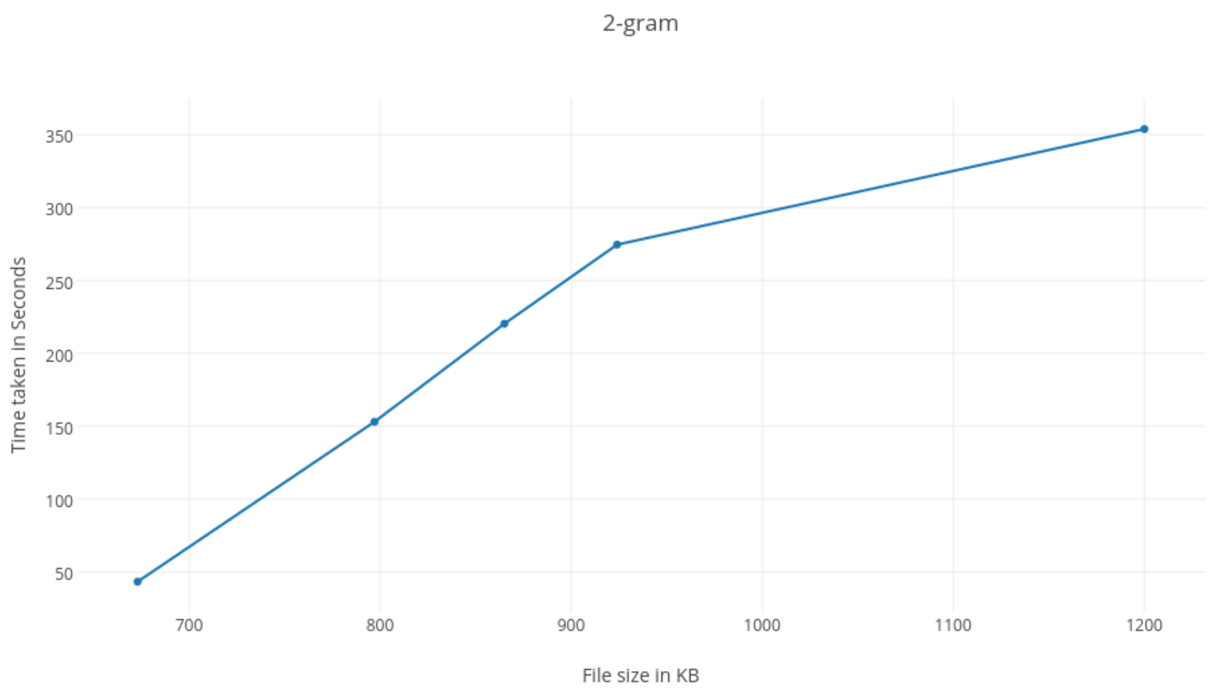
```
(('Volcanus', 'ad'), ['<verg. aen. 9.76>'])
```

Where the words, ‘Volcanus’ and ‘ad’ are the words whose co-occurrence has been computed, these words are within a bracket separated by commas. After the word bracket, a comma and a whitespace separates the word bracket with the location result. Locations results are stored in a Square bracket inside which various different location descriptions are present within angular brackets (<>).

It is also notable that efforts have been taken to sort the Unicode error by converting the lines to Unicode. This also removed u’ in front of some words, however, u’word or u’<description> may still be encountered for some files, in which case u’ should be ignored.

The performance of 2-gram for spark was very fast when compared to Hadoop. It is also worthy to note that unlike lab-4 lemmatization was performed in mapper itself, still the performance was very fast.

Plot for performance for 2 gram.



3-gram

The code for this activity can easily be run by just running the following command

Spark-submit 3_gram.py

It should be noted that new_lemmatizer.csv file should be in the same location as that of the code. Also there should be a input folder in the same location as the code. All the files upon which processing has to be performed should be within this input folder.

After running the code, a folder with the name of output3 will be created containing the results of the processing.

The output received will be in the following format:

((Word1(or lemma1), Word2(or lemma2), Word3(or lemma3)), ['<description1>', '<description2>'])

for example:

(('tum', 'et', 'demeo'), ['<verg. aen. 3.153>', '<verg. aen. 8.35>'])

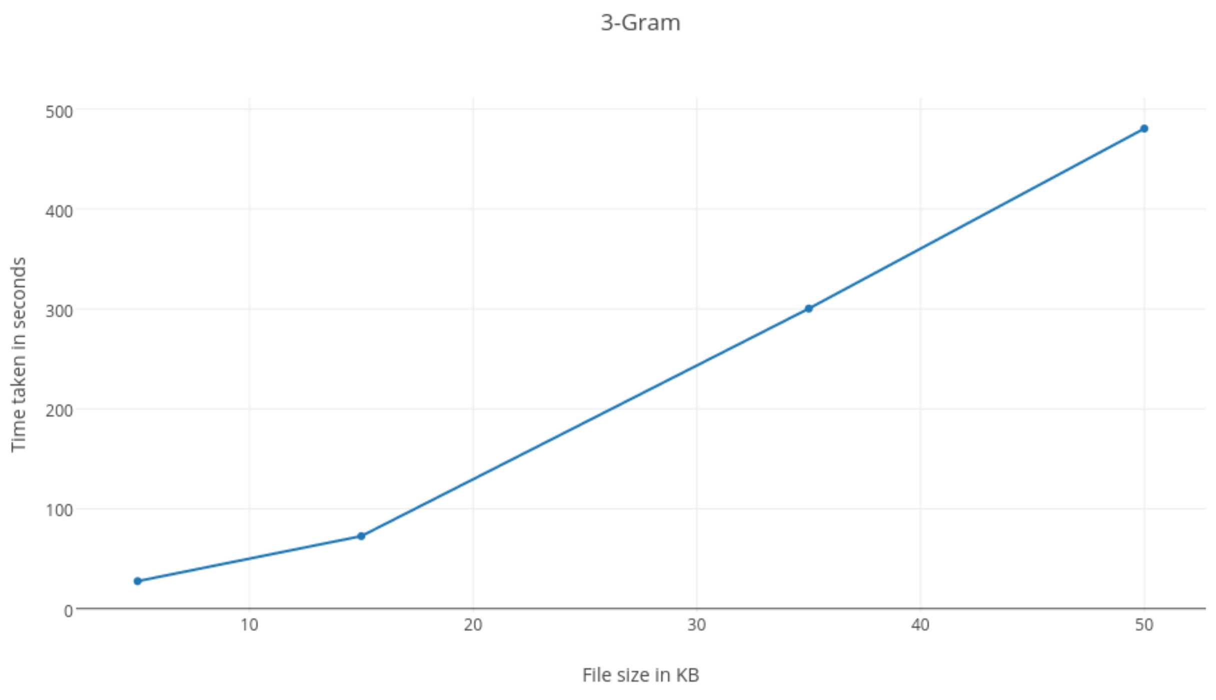
Where the words, 'tum', 'et' and 'demeo' are the words whose co-occurrence has been computed, these words are within a bracket separated by commas. After the word bracket, a comma and a whitespace separates the word bracket with the location result. Locations results are stored in a Square bracket inside which various different location descriptions are present within angular brackets (<>).

It is also notable that efforts have been taken to sort the Unicode error by converting the lines to Unicode. This also removed u' in front of some words, however, u'word or u'<description> may still be encountered for some files, in which case u' should be ignored.

The performance of 3-gram for spark was very fast when compared to Hadoop. It is also worthy to note that unlike lab-4 lemmatization was performed in mapper itself, still the performance was very fast.

It is also notable that performance of 3-gram was slower than 2 gram which is to be expected. Hence the file sizes were smaller than 2-gram.

Plot for performance of 3-gram



Performance and scalability in Activity

- It is worth noting that Lemmatization in Mapper is a lot slower than when it is done in the reducer. Doing the lemmatization process in the reducer drastically reduces the computation time. However, the con of this approach is that we lose out on Accuracy, as now there are duplicates in the final output.
- However, Since Spark is many times faster than Hadoop, it was possible for us to do lemmatization in Mapper itself hence maintaining the accuracy.
- It is also worth noting that the Hardware specifications of the machine on which the program is running also makes a big difference to the computation time required for the program to finish. I personally increased the RAM of our personal machine from 512 MB to 2GB to process data faster.
- It is also clearly observable that finding out Word co-occurrences for a Pair is many times faster than finding co-occurrences for triplets. This is very obvious due to the additional computation involved to accommodate the third word and its lemma's. Additionally, our observations while finding out the times required for computation strengthened this belief as the time required was significantly higher. We had to scale down on the sizes of the input to obtain observable times.
- We can also observe a linear relationship between the time required for computation and the total size of the input. Bigger the input, more will be the time required.
- It is also observable that the number of files selected for input, and the input files themselves also have some relationship with computation time.

-