Assignment 2 - Matrices and Graphs

Anirudh Narasimhamurthy(u0941400) and Soumya Smruti Mishra(u0926085)

September 17, 2015

1 Matrix-Vector and Matrix-Matrix Multiplications

In this part of the assignment we implement the basic matrix vector multiplication. The steps followed for implementing matrix vector multiplication are as follows:

Matrix-Vector Multiplication

- Read the matrix data from the text file and apply map() function to convert all the values to floating point numbers.
- Map the individual row in each file to a key value pair of the form (i, [Mij]) i.e an index and a list of values .
- Apply a reduceByKey() operation on the rdd to get an index and list of all the elements. This basically gives us the data points of the entire row of the matrix.
- Read in the vector data from the input text file. We had our own function to remove the characters and pull out just the data from the vector file. The function which we have used is available in the script attached.
- The matrix vector multiplication is then performed by doing a cartesian on the vector and matrix RDDs. The resultant RDD is then run through a map function where in we use numpy's dot product to compute the product of the first row and the vector.
- The same process applies for every single row of a matrix.
- **We see in this process that the replication rate for the vector is very high. We find that the vector is getting replicated 'n' times where n is the number of rows in the matrix**

Matrix Matrix multiplication (Single block approach)

- For the matrix matrix multiplication, we just followed similar approach to how we mapped our values for the matrix vector multiplication. Our matrix data values were mapped in the form of (i, Mij) and (k,Mjk).
- The cartesian of the two matrices ensures that all possible and required row-colulmn multiplications are performed.
- Again we made use of numpy's dot product method which conveniently performs the multiplication and also sums up the result, which is effectively what will be the value of each cell in the resultant matrix product.

- In effect the implementation we have performed multiplies elements of the first row of the first matrix or one block with the first column of the second matrix (or another block) and the result is the value of the cell(0,0) in the resultant product matrix.
- The cartesian() function ensures all possible multiplication pairs are generated and the matrix multiplication results given by our program matches exactly with the expected result. We just verified our results by constructing a smaller matrix in the same format as that of the input files and used them in our code.

One-pass approach:

- We also implemented the one-pass approach for matrix multiplication.
- The logic implemented is exactly similar to the algorithm posted in the lecture slides.
- We used the join() transformation to combine the two RDD's as opposed to using cartesian() in our previous approach because the keys are generated in such a way that join() spits out the required row and column we wanted for the multiplication.
- The multiplication was then performed using numpy's dot product function.
- The results produced by the one-pass approach exactly matched the results produced by our other approach mentioned earlier.

On analysing, both the methods the one-pass method seemed to do better

Block-approach

- We have taken block sizes of 10*20 and 20*5 and divided our matrixes respectively for all files. We can change the block sizes by going into the file as desired.
- We implemented the block matrix algorithm according to the pseudo code given in this link: http://magpiehall.com/block-matrix-multiplication-with-hadoop/ .It is similar to what was taught in the class but for better understanding of the code, we preferred to attach/site this link.
- Experiment with different block sizes. Do you need to make changes to your code to accomodate arbitrary block sizes? Answer: As mentioned above we can change block sizes by going into the code and all block sizes will work fine. We could have added the functionality of passing the block sizes from command line but due to lack of time we couldn't do it.

Regarding the Outputs:

- The outputs in both the cases of matrix-vector multiplication and matrix-matrix multiplication were finally stored in text files. We used saveAsTextFile() method to write the results to the file.
- Interestingly the saveAsTextFile() method produces the output file in different parts/formats when run from an Ipython notebook and as a normal python program.
- The iPython notebook produced a single file as output but whereas when the program was run from the terminal as a standalone python program utilizing Spark libraries, the output was produced in a file containing several parts.

- So if you are running it from the cluster or as a standalone python program the resultant text file will probably have several parts with each part containing the results of part of the operations.
- We did have an orderByKey() function before writing our results, so that would ensure that the results are in ordered fashion while you look across different files.

  Note: We have attached the samples of both single file output and part file outputs in the project directory in the folder named 'results'.

2 Shallow Graphs

To implement the shallow graph detector we tried and experimented with three different approaches which are explained below:

Approach 1: Constructing the full adjacency matrix and computing $A^2 + A$ to detect a shallow graph

- In this approach we constructed a full adjacency matrix for the given input data by first constructing a matrix of zeroes for the maximum given input and then building a RDD out of it.
- We then used the union() function to combine the above created RDD with the RDD created by loading the original data, so that the matrix represents the actual data.
- We then performed matrix multiplication for this matrix(A) using our code used for Problem 1 and again used a union() to perform the addiition of two matrices.
- We have applied further map and reduce operations to bring the data in such a form, which will quickly enable us to determine if the graph is shallow or not.
- Our final output format was in the form of (vertex,value). If the value for any of those vertex,value pair were 0, then the graph is not shallow, else it shallow.Basically it implies one node could not be reached from another node by atmost 2 hops.

Comment: Although the approach of constructing the entire/full adjacency matrix for a given sparse input involves computational and storage overhead, the method did seem to run slightly faster when compared to the other approaches which was surprising.Maybe it could be due to the fact that the code was run on my laptop and not on the cluster.

Approach 2: Working with the given data/adjacency list as such and computing $A^2 + A$ to detect a shallow graph

- As mentioned above, the process of constructuting an adjacency matrix by adding zeroes is inefficient. In this approach we just built the matrix with the given inputs without filling in any zeroes for the rest of the data.
- The key value pairs that were generated were in the form of (vertex,dict of vertex,value). We then used our matrix multiplication used in Problem1 to compute A^2.
- The only difference being the multiplication was now a user defined operation instead of numpy's dot() since we need to multiply those values for which the i and j matches.
- After computing A^2, similar approach to Approach 1 was followed to compute $A^2 + A$.

– The check for shallow graph in this case was made by checking if the count() after the appropriate map and reduce operations was equal to the input data size. If it wasn't then it implies atleast one of the nodes couldn't be reached from another node in atmost 2 steps, thereby invalidating the shallow graph property.

Comment: This method is really efficient in terms of the processing we do and is much more elegant solution compared to building the entire adjacency matrix.

Approach 3: Computing A.(A+I) to determine if the graph is shallow or not

– Another method of determining if a graph is shallow or not is to try computing $A^2 + A$ in terms of $A.(A + I)$

– We implemented the code for this approach and again the factor which was key was, to perform the matrix addition, we needed to create the identity matrix for the full graph and so had to in a way create the full adjacency matrix which was again inefficient.

– The multiplication operation was performed like in the approaches mentioned above with the only difference being it was not the same matrix which was getting multiplied here as the addition with Identity matrix modifies the input matrix.

Answer to point no 5 posted in Assignment:

– As far as the implementation goes, implmenting A(A+I) would again involve or require you to have the full adjacency matrix because the addition of A+I has to be done or can be done only on two matrices of same size. And so this approach would require us to have the full matrix built with zeros and values.

– So we feel this approach of doing $A.(A + I)$ might still not be radically efificent when compared to computing $A^2 + A$ directly from the graph. On the efficiency front, computing $A^2 + A$ using just the given input without adding zeroes was much better. But implementation and understanding wise, its easier to correlate when we have the full matrix.

Can we do better than A^2

– Another approach of doing a shallow graph detection without computing A^2 could be as follows:

– From the given data, for a given node we know all the nodes which the given node can directly reach by 1 hop, which are basically represented by 1's in the data.

– Add them to a list with key being the vertex.

– Now go over the list of immediate neighbor nodes and again repeat the process of finding each of the node's immediate neighbors.

– Apply a flatMap() transformation to finally get a node and its list of vertices which could be reached by 1 or 2 hops.

– Now that we have a list of nodes for every node apply a reduceByKey operation and then check if any of the list have a length which is less than the total number of nodes in the graph.

– If so, then the graph is not shallow, else the graph is shallow.

The code for all three approaches have been added to the submission fiolder and their filenames are suggestive of the apprpoaches discussed above.