3rd May 2016

Analysis of Scientific and Data Driven algorithms using Map-Reduce

Group number -11

CIS 612: Cloud Computing

Ninad Lanke (622463983)
Babita Singh (955434341)
Dhaval Dholakiya (327484578)
(UNDER THE GUIDANCE OF DR. JIAN TANG)

Background and Motivations

The motivation behind this project is to compare performance of different data driven and scientific algorithms in different frameworks with a file size of 100MB. The frameworks include standalone Hadoop environment and Amazon AWS.

Based on this execution we derive some significant analysis which can say a lot about the execution cost of the categorized algorithms like scientific and data driven algorithms. Based on this analysis any new algorithm based on its category can get prediction for its efficient implementation.

Related Works and Differences

While researching about the project we came across many research works where different Algorithms were implemented in various framework but there were only limited publications on execution cost of these various algorithms on different frameworks

One paper which we came across was "Services in the Cloud and Big Data era" which was published in IEEE.
(http://wcnc2016.ieee-wcnc.org/sites/wcnc2016.ieee-wcnc.org/files/u42/Invited%20Talk%20-%20Albert%20Zomaya.pdf)

We have tried to simulate the results published in this paper.

The above paper has general analysis for an algorithm on different platforms. We tried to take ahead this research by comparing the performance of different algorithms which were different from each other in the attributes such as the complexity (scientific algorithms) of the algorithm and the amount of data they deal with (Data driven algorithms).

So we came with the proposal to analyze execution cost of categorized algorithms for different frameworks.

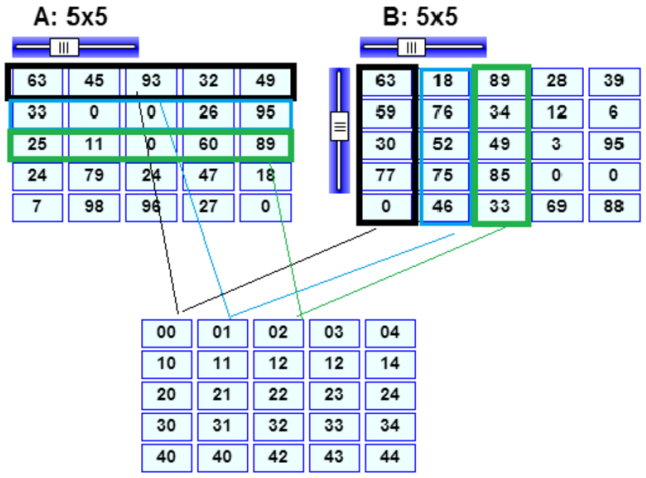
We have implemented six different algorithms which are a mix of scientific, data driven and hybrid algorithms.

Scientific Algorithms	Data Driven Algorithms	Scientific + Data Driven Algorithms
• Page Rank • Matrix Multiplication	• Document Set Search	• Breadth First Search • Semantic Analysis • Sorting Algorithm

Problem Definition (Matrix Multiplication)

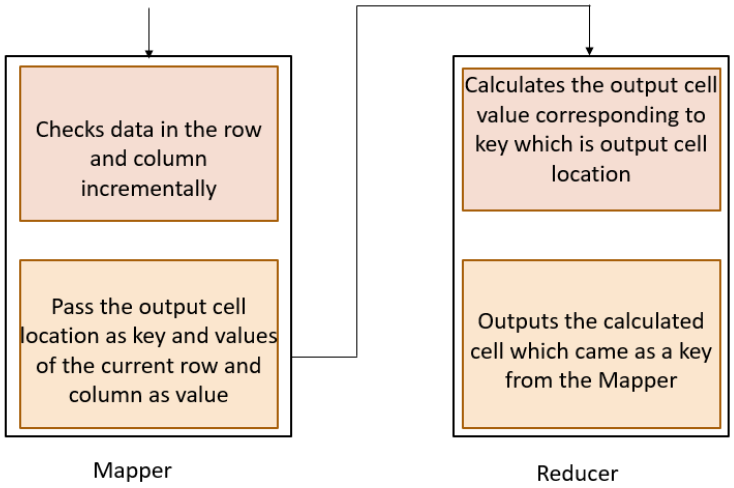
Multiply two matrices M and N using Map Reduce in parallel

Example:



Proposed Method:

1. The mapper gets the file as input.
2. The mapper then searches for the first matrix and the second matrix.
3. It calculates the size of the matrix to know the number of rows and cols
4. Checks the data in the row and col incrementally
5. Passes the output cell location as a key and corresponding values to compute the resultant matrix as value.
6. The reducer then fills each output cell according to the keys passed by reducer
7. It computes the output cell and fills it with data
8. This process happens incrementally to give the final output



Problem Definition (Page Rank Algorithm)

To compare implementation performance of Page Rank algorithm and recommend the best environment to run the application based on the size of input.

Proposed Methods:

Since Page Rank is a stage algorithm, it can be nicely implemented in Hadoop framework. Here is the MapReduce Implementation of the Page Rank algorithm:

I have used the following formula to calculate the rank of the page:

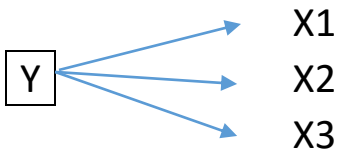
$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

PR(pi) → page rank of pi page
N → total number of nodes
d → is the damping factor(constant)
L(pj) → outgoing links from pi to pj

Mapper:

1. Mapper takes one row at a time of the adjacency matrix and get the outgoing links from that page
2. Initially mapper will assign equal rank to all nodes and emits the initial ranks to reducer
3. In second run it takes each node and takes rank in consideration and check for outgoing links, based on the number of outgoing links mapper divide this rank associated with the node to all it's outgoing links and emits the results as nodes with the ranks

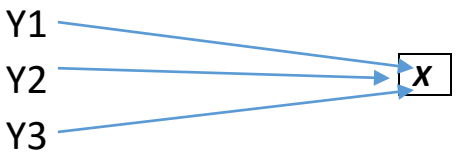
   For j=1 2 3 …… N : emits (xj * { PR(y) / Out(y) })



Reducer:

1. The Reducer gets the node from the mapper and check for all its incoming links
2. Based on the ranks of all the incoming links it aggregates all the contribution by summing up all incoming rank to calculate the rank of the current node:

(xj { (PR(yi) / out(yi)), (PR(yi) / out(yi)), (PR(yi) / out(yi)), …….. (PR(yn) / out(yn)) })



The iteration form mapper to reducer and reducer to mapper goes on until we get the stable state i.e. when ranks of all the pages are stable and doesn't change in further iterations.

The stabilized ranks will be printed out.

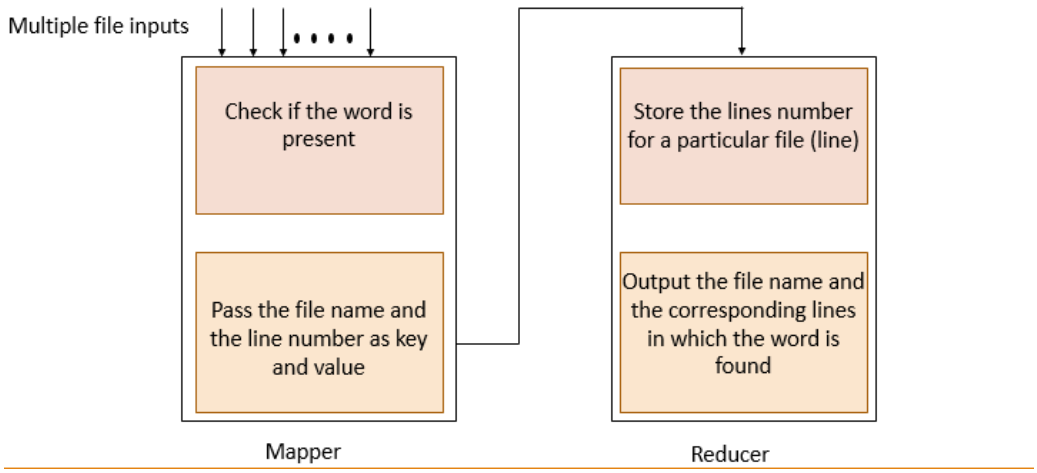
Problem Definition (Document Search)

Search a given set of files for presence of a word and the line number.

For example:
If the word is present in file A on line 45, then the output should be file A – 45

Proposed Method:

1. The mapper gets the multiple files as input.
2. The mapper then searches for the word in the line, if it finds the word, then we retrieve the file name.
3. We pass the file name and the line number to the reducer.
4. The reducer then outputs the filename along with the line numbers.



Problem Definition (Sorting)

We are presented with tuples which are in the key value pair. Our goal is to sort them as per keys and also as per values.

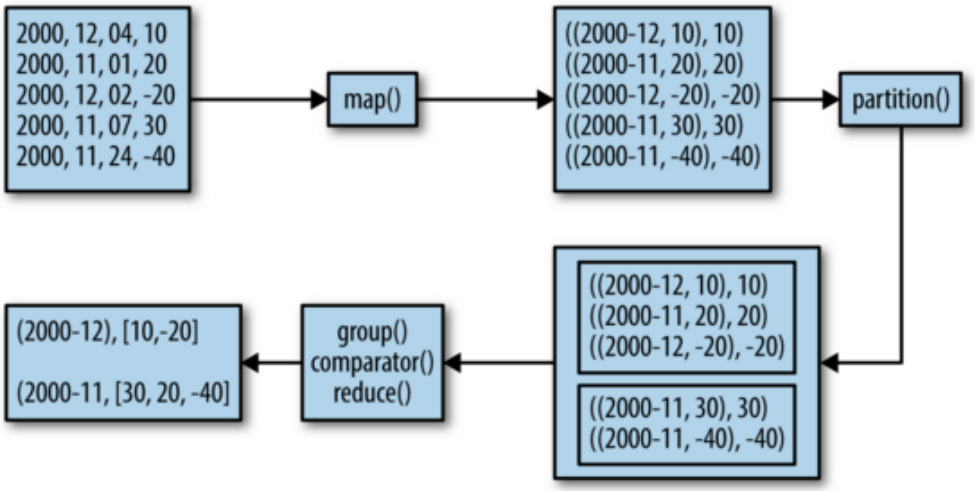
For example:

Year	Month	Day	Temperature
2016	01	11	40
2016	01	10	67
2016	04	01	30
2016	04	10	56

Year	Month	Temperature
2016	01	[40, 67]
2016	04	[30, 56]

Proposed Method:

1. The Mapper gets the input data. It converts it into key value pair. We generate the composite key. This key consists of Year-Month-Temperature. And the value field is the temperature.
2. The partitioner decides which mapper's output goes to which reducer based on the mapper's output key. To accomplish this we need to make use of Custom Partitioner and custom comparator.
3. The custom partitioner ensures that all data with the same year-month pair is sent to the same reducer. Thus the data with the same natural key is grouped together.
4. The custom Comparator does sorting so that the year-month groups arrives at the reducer. Once the year-month sorting is done, we sort the temperature which are associated with the same year-month group. Thus we get the sorted output.



Problem Definition (Sentiment Analysis)

Predict the sentiment of the given text based on the sentiments of other texts.

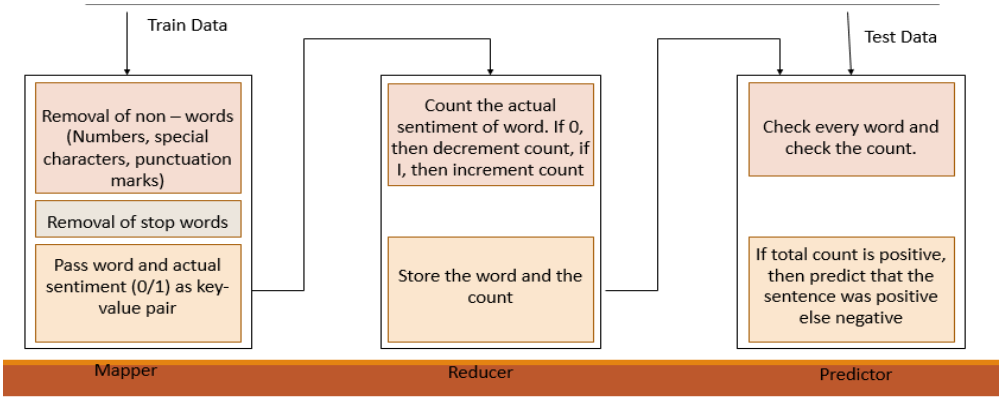
For example:

1	The Da Vinci Code book is just awesome.
1	i liked the Da Vinci Code a lot.
0	And I absolutely loathe the Da Vinci Code.
0	The Da Vinci Code sucked, so you didn't miss anything special.

Where 1 indicates as positive sentence and 0 indicates a negative sentence.
And if I get the text for sentiment detection as:
i love being a sentry for mission impossible and a station for bonkers.
Then the algorithm should predict that the sentence is positive (1).

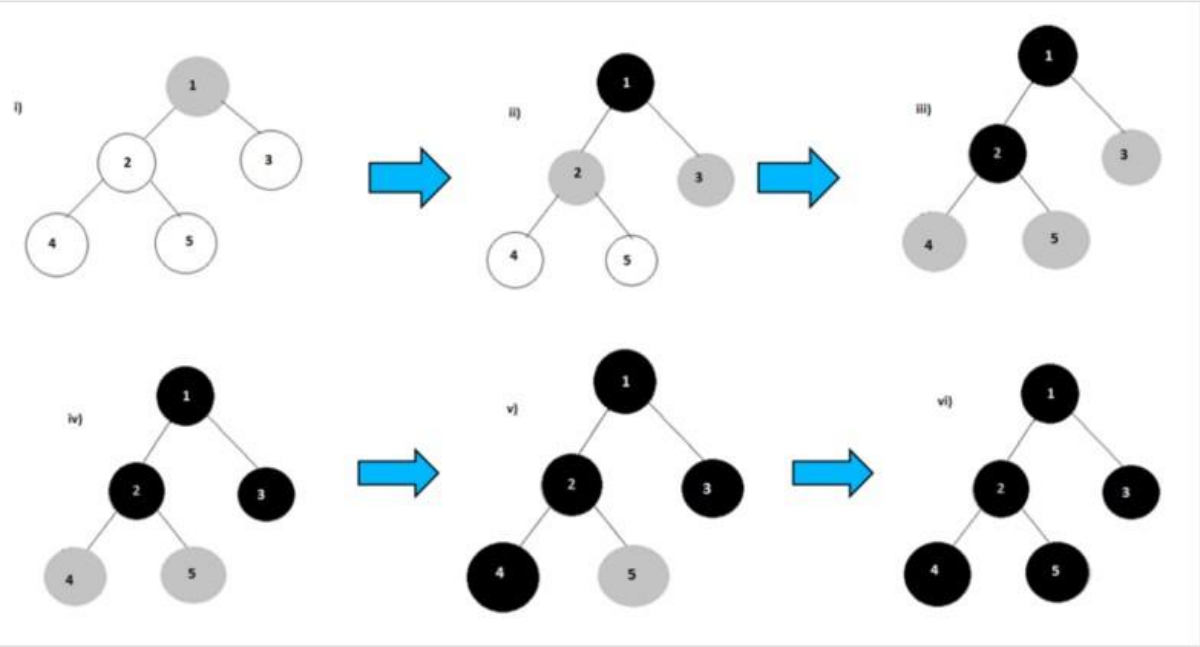
Proposed Method:

1. The mapper gets the training data. The mapper then removes the 0/1 from the text and saves it. It then performs pre-processing on the text, such as removing the non-character symbols, removal of stop words and tokenizing the text.
2. We then pass each token along with the 0/1 saved value to the reducer.
3. The reducer the keeps a count for each token such that, if a 1 appears, then we increment the count and if a 0 appears then we decrement the count.
4. Thus this token value pair is saved.
5. Hence we complete the training phase of the classifier.
6. Next, we input the test data.
7. We perform the preprocessing on the test data and tokenize it too.
8. We check for each token in the saved list. If the word is found, then we check the value of that word in the list. We add this value to the sentiment of the sentence. We do this for all the tokens of a given sentence.
9. Thus if the total sentiment is positive, then we predict that the sentence is positive, else we predict that the sentence is negative.



Problem Definition (Breadth First Search)

To compare implementation performance of Breadth First Search algorithm and recommend the best environment to run the application based on the size of input.



Proposed Methods:

MapReduce Implementation of the BFS algorithm:

1. The MapReduce implementation of BFS follows an iterative MapReduce processing in which the output yielded by the reducer will be taken as input for the mapper.
2. Initially the input file will have sparse matrix representation of the graph's adjacency list.
3. This Mapper will read this file line by line. At each line mapper will get a node, connected edges to this node, distance from the source and status(whether unvisited, visited or about to be visited) for example: ID EDGES|DISTANCE_FROM_SOURCE|STATUS|
4. For each unvisited node mapper will generate a set of about to be visited nodes and change the status of current node as visited. Then again in next turn mapper will take these about to be visited nodes and further generate the about to be visited nodes.

   ATV : About to be visited nodes NV : Not visited V : Visited

   1 2,5|0|ATV|
   2 1,3,4,5|-1|NV|

3 2,4| -1 | NV |
4 2,3,5|-1 | NV |
5 1,2,4|-1 | NV |

After first iteration here is an output:

1 2,5|0|V|
2 NULL|1|ATV|
5 NULL|1| ATV |
2 1,3,4,5|-1 |NV|
3 2,4|-1| NV |
4 2,3,5|-1| NV |
5 1,2,4|-1| NV |

when the mappers process the ATV nodes and creates a new node for each edge, they do not know what to write for the edges of this new node - so they leave it blank.

5. Reducers, of course, receive all data for a given key - in this case it means that they receive the data for all "copies" of each node. for example, the reducer that receives the data for key = 2 gets the following list of values :

   2 NULL|1|ATV|
   2 1,3,4,5|-1|NV|
6. The reducer's job is to calculate values for the null spots, Change the status of the nodes and calculate the distance.
7. This way Mapper and Reducer will keep on sending their outputs to each other and keep on processing inputs from each other, the iterations will continue until the status of all nodes turns to Visited (V)
8. At last we print the maintainer list order of the visited nodes.

Simulation and analysis for the results

Simulation:

This application analyzes algorithms which are scientific, data driven and combination of both. Currently we are have tested our algorithms with file sizes of 100MB. One of the main task in this application was to execute Map Reduce Job on Amazon Cloud as well as standalone applications over these 100 MB files. We had to create clusters to run the jobs on the EMR. S3 was used for storage of the jar as well as the input and the output files. The output from this task is analysis of all the algorithms in three categories.

Type of Cluster	No. of CPU	Core Nodes	Master Node	Specifications	Map-reduce execution time
Matrix Multiplication (Scientific Algorithm)					
Normal Execution	1	2	1	Mem: 4 GB Speed: 2.53GHz Storage: 4GB (SSD)	195.83 mins
Compute Optimized (c3.xlarge)	4	2	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	125 mins
Compute Optimized (c3.xlarge)	4	4	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	113.10 mins
Page Rank (Scientific Algorithm)					
Normal Execution	1	2	1	Mem: 4 GB Speed: 2.53GHz Storage: 4GB (SSD)	65.52 mins
Compute Optimized (c3.xlarge)	4	2	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	41.93 mins
Compute Optimized (c3.xlarge)	4	4	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	38 mins
Breadth First Search (Hybrid Algorithm)					
Normal Execution	1	2	1	Mem: 4 GB Speed: 2.53GHz Storage: 4GB (SSD)	20 mins
Compute Optimized (c3.xlarge)	4	2	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	12.8 mins
Compute Optimized (c3.xlarge)	4	4	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	11.6 mins
Sentiment Analysis (Hybrid Algorithm)					
Normal Execution	1	2	1	Mem: 4 GB Speed: 2.53GHz Storage: 4GB (SSD)	28 mins
Compute Optimized (c3.xlarge)	4	2	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	17.92 mins
Compute Optimized (c3.xlarge)	4	4	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	16.24 mins
Sorting (Hybrid Algorithm)					
Normal Execution	1	2	1	Mem: 4 GB Speed: 2.53GHz Storage: 4GB (SSD)	16 mins

Compute Optimized (c3.xlarge)	4	2	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	10.24 mins
Compute Optimized (c3.xlarge)	4	4	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	9.28 mins
Document Search (Data Driven Algorithm)					
Normal Execution	1	2	1	Mem: 4 GB Speed: 2.53GHz Storage: 4GB (SSD)	1.8 mins
Compute Optimized (c3.xlarge)	4	2	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	1.152 mins
Compute Optimized (c3.xlarge)	4	4	1	Mem:7.5GB Speed: 2.6GHz Storage: 4 x 40GB (SSD)	1.044 mins

Analysis:

From the above calculations we can say that we get a considerable change in the execution from the standalone Hadoop implementation when we run jobs on a file size 100MB.

Comparison with Local Machine implementation:
We already had the Hadoop Job setup on our local environment (4GB RAM, 2 Core, Processor Clock Speed 2.53GZ) when we tried executing the same task it took considerably less time on the cloud environment which is comparable to the Compute Optimized 2 core cluster. The processing time for any algorithm is the least for 4 core cluster when executed on a cloud setup.

Comparison among different types of algorithms:

Scientific:

The scientific algorithms took the maximum amount of time considering the complexity of the algorithms as compared to others. We can also observe that the execution time reduces the most for the scientific algorithm as it took the most time to execute.

Data Driven:

The data driven algorithms takes the least amount of time as we can parallelize the data in the cloud setup. The processing is very little in these algorithms so the execution time improves tremendously. As we can observe from the table, these algorithms have the least running time for cloud as well as standalone Hadoop.

Scientific and data driven (Hybrid):

The hybrid algorithm which is scientific as well as data driven lie between the other types as it has little processing going along with the data driven nature which allows parallelization.

Contribution

Ninad Lanke	Natural Language Processing, Document Search, Sorting Algorithms
Babita Singh	Breadth First Search, Page Rank Algorithm
Dhaval Dholakiya	Matrix Multiplication Algorithm, Implementation of cloud services and execution of algorithms on Amazon AWS

All three members contributed equally to research about the topic and algorithms that would be the best fit for the application. Implementation part was done by all the three members with equal contributions. Although the contribution is mentioned separately but everyone has part in each and every implementation of the application as well as cloud implementation as the use of Amazon web service for done for the first time, everyone has equal difficulties in their implementation which were solved by collaborating the ideas and thinking of all the three members

Also everyone has equal contribution to the report. Individually every member has provided his part of work performed and added the information about the work into this report.

Conclusion

We have successfully built the application that gives analysis of three categories of algorithms over different frameworks. We concluded that EMR gave us fast and cost effective executable platform for map-reduce jobs. We successfully simulated the results as proposed in the “Services in the Cloud and Big Data era” paper.