Big Data Analytics

Assignment 1

Arjun Subramanyam Varalakshmi (1546081)

Problem Description:

In this assignment, we will be dealing with the problem of calculating term frequency and inverse document frequency for large set of books. Term frequency indicates how often a word appear in a book, and inverse document frequency indicates in how many books does a term appear, which can be used to compute Tf-Idf, "it's a numerical static that is intended to reflect how important a word is in a document in a collection or corpus." [1]. Computing Tf-Idf would aid in removing stop-words in various text summarization and classifications.

The assignment consists of 3 parts:

Part – 1: Computing term frequency
Part – 2: Computing Inverse document frequency
Part – 3: Analyzing the impact of change in number of reducers on the performance of the part – 1 and part – 2 programs.

In this assignment, we would be using Hadoop framework to complete the assignment.

Solution Strategy:

We would be solving the above problem using the map reduce programming model. Using map reduce paradigm to solve the above problem in a novel idea.
We would be using Pydoop framework to implement map reduce programming model to provide solution for the above problem.

The reason to choose Pydoop framework over various interface to Hadoop framework is for the following reason: Pydoop provides a pure Python client for Hadoop pipes and easy access to HDFS files using the inbuilt library in Pydoop to access HDFS. It also uses simple Python. It provides simple python interface for writing the code and provides various configuration options to optimize the execution.

There are different possible solutions which can be provided using the map reduce programming model, the approach followed by me to solve this problem is as described below:

Step 1:

Input:

File containing list of books:

/cosc6339_s17/books-longlist-directory.txt

Output:

Files containing Document name followed by the words and it's count in the document:

/bigd21/Step1Output

Mapper Input:

Key-Value pair consisting of byte-offset as key and line of text as value

<byte-offset, line of text>

Mapper Output:

<Document Name, {Word: List of 1's}>

Reducer Input:

Same as mapper output since the key emitted by mapper will have only one value

Reducer Output:

<Document Name: "Book name">
<Word: Count>

Step 2:

Input:

File containing path to merged output of Step 1

/bigd21/step2_input

Output:

File containing word and its occurrence in no of documents

/bigd21/Step2Output

Mapper Input:

<Byte offset, path to merged output of step1>

Mapper Output:

<Word, Document name>

Reducer Input:

<Word, list of documents in which its present>

Reducer Output:

<Word, count indicating no of docs its present>

Step 3:

Would run the Step 1 and Step 2 programs with different number of reducers like 2, 5, 10, 15, 20, 100 and compute the time taken by each step and the impact of change in number of reducers on the performance of the program.

Alternate Solution: Pass directory containing books as input and then read each line, obtain the file name from which split is obtained, and do the following:

In Step - 1, we can emit the below key value pair:

<Document Name, word+" ="+1>

In reducer split the value using "=" as delimiter then sum all the one's which would provide same result.

Why I didn't use alternate solution?

Since the assignment insisted on using Pydoop framework, I decided to take advantage of the HDFS API provided by Pydoop. Using HDFS API we could load the entire file contents once which would reduce the I/O latency, compared to reading line by line from each file. By following first approach I am reducing the time taken to obtain results.

Code Explanation:

Step 1:

- We would provide a text file consisting of list of book titles as input to the program.
- The mapper by default takes byte offset as key and the line of text pointed by the current byte offset as value, which is the title of the book.
- We would use Pydoop HDFS API and load the contents of file into a variable.
- To load the contents of the file we would require the absolute path of book, which we would compute using the current path of the input file.
- We assume the parent directory of input file also contains the books.
- We would emit following key-value pair as intermediate results from mapper: <Document Name, {word: list [1…1]}>
- Document name would be key and a dictionary consisting of word as key and list containing 1 for each occurrence of word in the document.
- In the reducer phase, for each document we would get a dictionary consisting of words and the list of 1's indicating its occurrence. By summing the values of list for each we can obtain the total number of occurences of word in the document.
- The reducer would emit the "Document Name: <Doc name>" for each document, which is followed by the words and its occurrence in the document.

Step 2:

- In the Second part, we would be merging the output obtained from part – 1 and would place the merged file onto HDFS.
- We would pass a file consisting of the HDFS path to the merged part – 1 result as input to the mapper.
- The mapper would just execute once and obtain the contents of the part – 1 output using HDFS API and which we would be formatting to emit the following intermediate results from mapper: <word, document name>
- The reducer would receive document name and list of document names that contain that word. We would create a variable named count at the beginning of reducer which would be incremented while iterating through reducer values.
- The reducer would emit <word, no of docs in which word is present>

How to run the code:

1. Unzip the HW1_1546081.zip folder and place the contents of the folder in the whale cluster.
2. cd to HW1 directory, it consists of a shell script named run_hw1.sh, to run the shell script give below command in terminal:
   a. ./run_hw1.sh
   b. If in case it gives permission error change permission using below command and execute it
      i. chmod +x run_hw1.sh (or chmod 777 run_hw1.sh) and execute.
3. The shell script would provide you with following 4 options:
   a. Run step1, step2, run all and quit.
   b. If first 3 options are selected the script asks the user to enter the number of reducers he would like to use to execute the code.
4. If in case it's required to change input directory path or how the input is given you need to edit the shell script and execute.

Results:

Description of Resources used:

The Assignment – 1 of this course is executed on Whale cluster, The components of Whale cluster can be as described below:

It consists of 57 Appro 1522H nodes

Hardware Details:

- CPU: Two 2.2 Ghz quad core AMD Opteron processor (8cores total)

- RAM: 16 GB
- Ethernet NIC: Gigabit Ethernet
- 4xDDR InfiniBand HCAs

Network interconnect:

- 144 port 4xinfiniBand DDR Voltaire Grid Director ISR 2012 switch (donation from TOTAL, shared with crill)
- two 48 port HP GE switch

Storage:

- 4 TB NFS /home file system
- 7 TB HDFS file system (using triple replication)

Software Details:

- Hadoop Framework version 2.7.2 (distributed – mode)
- Pydoop Framework version 1.2.0
- Python version 2.7.2
- Bash scripting

Description of measurements performed:

In the part 3 of the assignment we would measure the execution time of part – 1 and part – 2 separately by modifying the number of reducers:

No. of reducers	2	5	10	15	25
Avg Time Taken (in seconds)	72	70	68	104	104

Table 1: Time taken by Part - 1

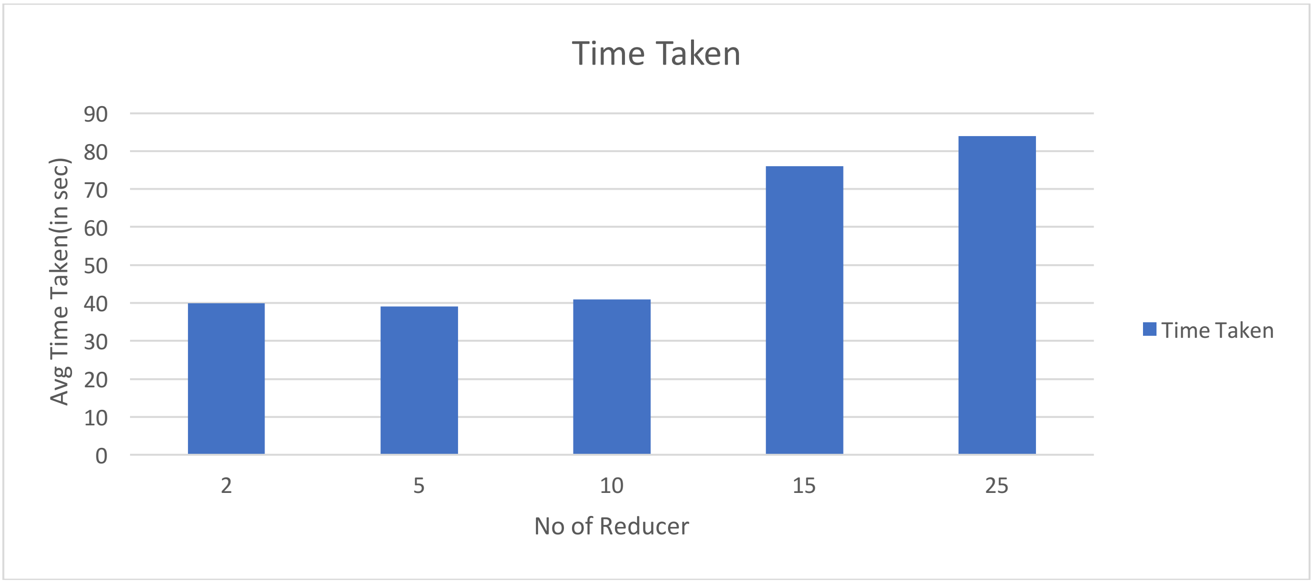

Fig 1. Time Taken by Part – 1

No. of reducers	2	5	10	15	25
Avg Time Taken (in seconds)	40	39	41	76	87

Time Taken by Part -2

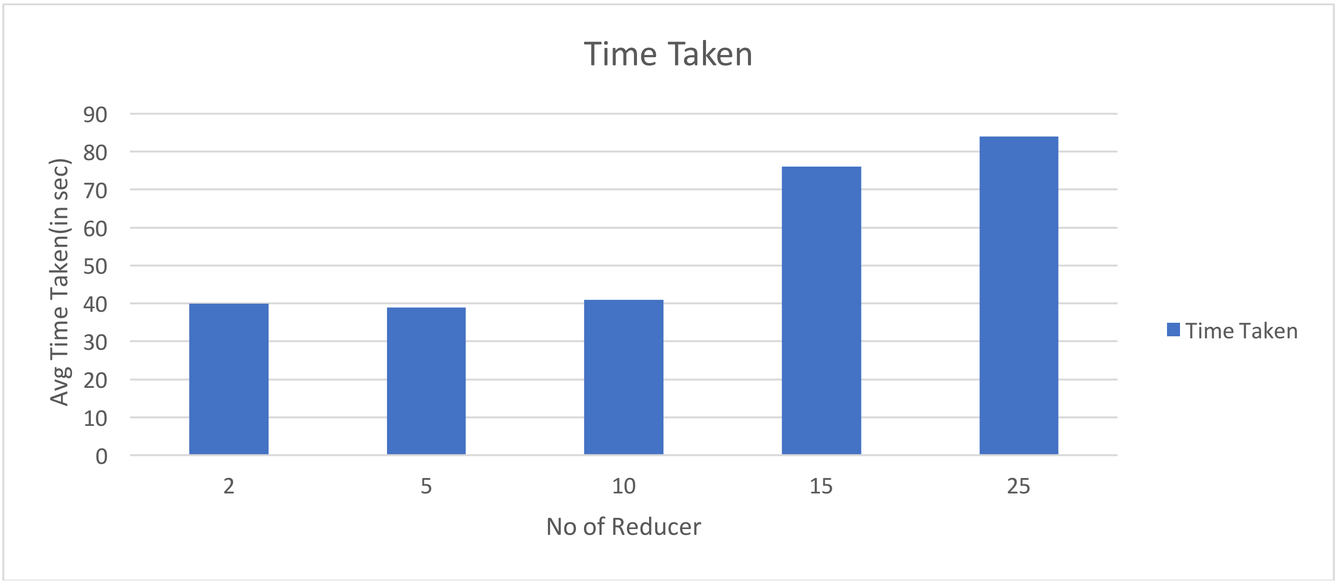

Fig 2. Time taken by Part – 2

If we observe from above tables and bar plots we can notice that initially when the number of reducers are increased we can see the time taken by the programs are reduced, but after certain number of reducers the time taken would increase rather than decreasing. The reason behind this can be explained as below:

a. When the output obtained from mapper is not proportional to the number of reducers, the I/O operations are increased and as each reducer creates its own output file the writes will be high.
b. In the solution approach, I followed in part – 1 there will be only n key value pairs, where "n" is the no of documents, if we assume n to be 10 then if we start 100 reducers obviously 90 reducers would do no work as each key and its corresponding values are assigned to the same reducer partition by default.

The ideal no of reducers for any task is obtained by doing random sampling of data and computing average times for no of reducers. We need to compute average time because a single measure of time would be error prone.

Why we need to compute ideal number of reducers for any map reduce task?

Because this would prevent the starvation of other execution processes and helps in effective utilization of available resources.

The ideal number of reducers for both Part – 1 and Part – 2 according to the above analysis is 5.

Conclusion

Learnings from above assignment:

- The effective usage of MapReduce programming model by leveraging on top of Hadoop distributed framework.
- An insight into how text processing problems containing large data sets can be solved used big data analytics.

Future Work:

- The above assignment can be extended to compute Tf-Idf and use it for analyzing stop word during text summarization and classification.

References:

1. http://pstl.cs.uh.edu/resources/whale
2. https://en.wikipedia.org/wiki/Tf%E2%80%93idf