COSC 6339 Big Data Analytics-Assignment1

-Sreekar Reddy Jammula (1422539)

Problem description:

The current task is to compute the word counts on a huge number of books in the Hadoop MapReduce environment. A detailed description of the problem is given below:
This assignment has been split into three tasks:

1. To count the occurrences of words in a book on a per-book basis.
2. To count the number of books in which a particular word occurs.
3. Find the execution times of above programs on 2,5,10 number of reducers.

Input Data Set: A large input data set consisting of 216 books present in the directory /cosc6339_s17/books-longlist/ has been used for this experiments

Outputs: All the outputs have been stored in the /bigd12/ Hadoop dfs directory

Solution Strategy:

The solution strategy is as follows:

1. The pydoop (Python for Hadoop) Interface has been used for this purpose.
2. The count of words in the books will be counted using the program wordcount.py
3. The output of this is program will be files that contain the data in the format <word @ bookname> <count of the word> which is a <key,value> pair.
4. Next, the files produced by the n number of reducers are merged. This will give us an exact count of the words on a per book basis. i.e., the resultant merged file will be having an entry for every word and its count for each time it appears in a particular book.
5. To solve the task 2, the output of task 1 can be used. The solution has been implemented in the file idf.py. As already mentioned the output of part 1 has the entry for the words each time it appears in a book.
6. We can use this fact and count the number of times a word is repeated in the output of part 1 to know how many books it appears in. i.e., as each occurrence of word is pertaining to one book followed by the count, counting the number of such occurrences is nothing but the count of number of books the word appears in.

Programming Environment:

Map Reduce Programming model: Map reduce consists of two phases. It takes the input in <key,value> format and also produces the output in the same format.

1. The map function splits the input into key and its related value format.For example, the word and its count. This is sent to the reduce function as intermediate key and value
2. This is then passed on to the reduces which consolidates the intermediate key and value and provides the output. i.e., in the word count example, the reducer is the function that sums up all its occurrences and finally produces its count.
3. There are many Map reduce frameworks available and Hadoop is one among them.

Pydoop: Pydoop is a python interface for Hadoop that allows us to write the Map Reduce applications very easily.

1. It provides a rich Map Reduce API and also an integrated Hadoop HDFS API so that writing the program and also managing the files is possible simultaneously.
2. The Pydoop MapReduce API provides a wide variety of functions to implement the mappers, reducers, read the data, write the data, manipulate the data using the split() functions etc.
3. Pydoop also offers transparent Avro deserialization.

Solution Explanation:

Wordcount.py: This program solves task 1.

1. It produces the counts of words on per book basis.
2. The mapper() function takes the input and splits it based on spaces. It then considers each split part as word i.e., variable “w” and emits to the reducer using emit() function. Alongside, we also need to get the file in which the word is present. This is observed as the path name is split and the file name is stored in the “filename” variable. The mapper emits the output in the form of (w + “@” + filename , 1) format where 1 is the value.
3. The reducer() performs the sum on the intermediate values and emits the final <key, value> pair
4. It may be observed that the program imports some of the libraries of the pydoop api to enable effortless and easy programming of tasks.

Idf.py: The idf. py program is similar to the wordcount program. It takes input as the part-r-xxxx files produced in the wordcount.py program

1. The mapper() function first splits the input it gets on the basis of space. This allows us to store the word@bookname part in the variable “key”. Then , for every occurrence of @ in key , a split is performed around the “@” symbol and the word is extracted into the “keyWord” variable. This along with value 1 is emitted to the reducer.
2. The reducer() function sums up the values of intermediate keys to give us a count of how many books a word appears in .

Executing the code:

To execute the wordcount.py program follow the following command :

pydoop submit --num-reducers <enter number of reducers here> --upload-file-to-cache wordcount.py wordcount <input directory> <output directory>

To execute the idf.py program follow the following command :

pydoop submit --num-reducers <enter number of reducers here> --upload-file-to-cache idf.py idf <input directory> <output directory

Results and Observations:

Task 1: The wordcount.py program has been executed on 2, 5 and 10 number of reducers. The execution time for these have been observed and tabled as below. Please note that the execution times are influenced by various factors such as number of other executions running on the cluster, availability of nodes etc.,

Number of reducers = 2:

	Trial 1	Trial 2	Trial 3	Average
Execution Time (in secs)	160	159	148	155.6

Comments: Not much variation in three trials. The average for three trials is taken.

Number of reducers = 5:

	Trial 1	Trial 2	Trial 3	Average
Execution Time (in secs)	115	113	95	107.6

Comments: Not much variation in three trials. The average for three trials is taken.

Number of reducers = 10:

	Trial 1	Trial 2	Trial 3	Average
Execution Time (in secs)	62	132	129	130.5

Comments: Variation between first and next two trials. Hence the first trial is ignored and the average is taken for 2nd and 3rd trials only

Observation:

1. As we observe keenly, the average time decreases when we have 5 reducers than 2 reducers. Increasing number of reducers helped. But this is not true at all times.
2. The execution time is more for 10 reducers compared to 5. This states that having more number of reducers does not necessarily decrease the execution time. When we increase the number of reducers, we are trying to parallelize the execution. Increasing parallelization always comes with this own tradeoffs. In this case, we are trying to communicate the data from mappers to 10 reducers. This causes a I/O overhead which increases the time of execution.

Task 2: The idf.py program has been executed on 2, 5 and 10 number of reducers. The execution time for these have been observed and tabled as below. Please note that the execution times are influenced by various factors such as number of other executions running on the cluster, availability of nodes etc.,

Number of reducers =2

	Trial 1	Trial 2	Trial 3	Average
Execution Time (in secs)	37	38	37	37.3

Comments: Not much variation in three trials. The average for three trials is taken.

Number of reducers = 5:

	Trial 1	Trial 2	Trial 3	Average
Execution Time (in secs)	56	57	44	52.33

Comments: Not much variation in three trials. The average for three trials is taken.

Number of reducers = 10:

	Trial 1	Trial 2	Trial 3	Average
Execution Time (in secs)	38	37	38	37.6

Comments: Not much variation in three trials. The average for three trials is taken.

Observation: When we use 5 reducers, more time is consumed. This may not necessarily be because of number of reducers used. A large number of cluster based factors may have influenced this number.

Additionally, another observation has been made based on the number of splits given to the mapper. If the number of splits is more the execution time tends to decrease for any give number of reducers. This has been a general observation and no specific experiment has been performed to illustrate the values.

Resources Used:

These experiments have been performed on the Whale cluster which is under the supervision the parallel software technologies laboratory. The cluster has been accessed through a secured SSH connection from the MobaXterm SSH client.

Given below are the details about the cluster:

Total number of nodes: 57 (whale-001 to whale-0057)

Processor Details: 2.2 GHz quad-core AMD Opteron processor (consisting of 4 cores each)- 2 in number(8 cores total)

Main memory: 16GB

Network Interconnect:

1. 144 port 4xInfiniBand DDR Voltaire Grid Director ISR 2012 switch

2. 48 port HP GE switch- 2 in number

Storage: 20 TB Sun StorageTek 6140 array (/home shared with shark and crill clusters)