Mining Big Data assignment 1

Made by

Yu Zhang a1795409, Qiming Liu a1798587

Exercise 1

Draft is included in appendix.

The number of suspected pairs under three different conditions:

Assume the Large number of combination C (n x) = (n ^ x)/(x!)

Number of pairs people = C (5*10^9 2) = (25 * 10^18) / 2 = 1.25 * 10^19

Number of pairs days = C (5000 4) = 2.6042 * 10^13

Possibility: two people fit the condition (arrive at same hotel at the same day) and happened in four different days = 1.6 * 10^-39

Suspicious pairs = Sample size * Possibility = (1.25 * 10^19) * (2.6042 * 10^13) * (1.6 * 10^-39)
=5.2084E-07

There are 5.2084E-07 pairs suspicious

Exercise 2

Question 1

TF.IDF, Term Frequency times Inverse Document Frequency.

It is the formal measure of how concentrated into relatively few documents are the occurrences.

Formula:

$$TF_{ij} = \frac{f_{ij}}{max_k f_{kj}}$$

f_{ij}: frequency of word i in document j.

$max_k f_{kj}$: maximum number of occurrences of any word (stop words excluded) in the same document.

IDF_i: term i appears in n_i of the N documents in collection.

$$IDF_i = log_2(\frac{N}{n_i})$$

$$TF.IDF = TF_{ij} * IDF_i = \frac{f_{ij}}{max_k f_{kj}} * log_2(\frac{N}{n_i})$$

Question 2

IDF for 40 docs:

IDF = log2(10^7/40) = 17.93156856932417 ≈18

IDF for 10000 docs:

IDF = log2(10^7/10000) = 9.965784284662086 ≈10

Question 3

TF.IDF for w appears once:

TF.IDF = (1/15) * log2(10^7/320) = 14.93156856932417 ≈1

TF.IDF for w appears five times:

TF.IDF = (5/15) * log2(10^7/320) = 4.97718952310806 ≈5

Exercise 3

Q1:

1. Copy and paste a new example project, create a new package and a class which inhere the Hadoop configured and Tool interface.
2. Copy the example code on the new class, put the material text file in the project directory (wait for analysis).

For standalone mode:

3. Set the run configuration of the project, create a new java application config, set the first argument as the input text file “100-0.txt”, second as the “output”, the new folder in project directory to store the output file.
4. Click run as java application, select the Wordcount in the package created before. Then run the code, wait for console execution. The result can be found in the “output” file was pointed in config before, in the project directory.

For pseudo-distributed mode:

3. Export the project as a jar archive file

4. Use the terminal, first input the text file into the HDFS file system. Command: $hadoop fs -put path/file.txt

5. Use the terminal command to run the jar file in the HDFS system. Command: $hadoop jar jar_file_name_generated_before.jar Package.WordCount file.txt output. The output is the directory in HFDS file system, which could not directly check in the user’s OS.

6. View the job-related information in the browser of Hadoop management system.

7. If user need to export the result file, use the Command: $hadoop fs -get HDFS/source/path destination/path.

The example output and the source code are in the Appendix folder: WordCount

Q2:

The provided code is trying to roughly count the occurrences of each word, but it doesn't exclude the distraction like punctation and number. The first part is Map function which split the string and build the hash pair for each word. The second part is Reduce function which sums up the value of every pair with same key.

The output of two modes is exactly same. But the process of two modes are different.

1. Standalone mode run the code in local Eclipse environment, with local OS file system. Pseudo mode need to control the HDFS distribute file system by terminal, and also related some local network request and respond (which you can see many sockets operation in the log)
2. Standalone mode can only deal with local resource which need to import and export file in project directory, Pseudo mode can use terminal to input and output in any location, by Hadoop server, even can deal with the jobs on other user's computer and provide the result.
3. Standalone mode only provides limited information for jobs running. Meanwhile in the pseudo mode, user can check the Hadoop system for more like logs and Node information.

Exercise 4

Part 1:

Result and logs are in file *Apendix.zip*: Result/*Firstfile_part1* and *Secondfile_part1*

Source code is in file *Apendix.zip*: WordCount_part1

Part 2:

Q1: How many words are there with length 10 in FirstInputFile?

10649

Q2: How many words are there with length 4 in FirstInputFile?

210081

Q3: What is the longest length between words and what is its frequency in FirstInputFile?

Length 27, frequency 1

Q4: How many words are there with length 2 in SecondInputFile?

63689

Q5: How many words are there with length 5 in SecondInputFile?

35762

Q6: What is the most frequent length and what is its frequency in SecondInputFile?

Length 3, frequency 73149

Part 3:

Result and logs are in file *Apendix.zip*: Result/*Firstfile_part3* and *Secondfile_part3*

Source code is in file *Apendix.zip*: WordCount_part3

Part 4:

Q7: How many words are there with length 10 in FirstInputFile?

1700

Q8: How many words are there with length 4 in FirstInputFile?

2021

Q9: What is the most frequent length and what is its frequency in FirstInputFile?

Length 7, frequency 4281

Q10: How many words are there with length 5 in SecondInputFile?

1826

Q11: How many words are there with length 2 in SecondInputFile?

72

Q12: What is the second-most frequent length and what is its frequency in SecondInputFile?

Length 8, frequency 2628

Exercise 5

Question 1, summarize Section 2.4:

Extensions to MapReduce.

MapReduce spawned a lot of extension and modification systems. They both have some common concept for MapReduce. Such as they all built on the base of distributed file system; all can manage large number of tasks by small number of functions; they also have incorporated methods dealing with failures, when executing a large job, without restart whole jobs if there is failure. One cluster called Workflow, which supporting acyclic networks of

functions (each implemented by collection of tasks). UC Berkeley's Spark and Google's TensorFlow are most popular. Another family uses Graph Model of data, which let computation implemented in the node of the graph. Google's Pregel is the most famous one. Now many workflow systems also implement the graph-model facility.

Workflow system extend the MapReduce two step system, which is Map -> Reduce, to an acyclic graph structure with multiple steps. A briefly example is chain two MapReduce jobs together. It can divide work among task by a master controller, like a regular map task. Data is passed by file collection and feed the input to each function node. There is a great property inhere from MapReduce is "Blocking property". Which means it only provide output after complete. Though once a task fails, it won't passed to any successors in flow graph. The master controller will re-activate it in another nod. So, user do not need to worry about the previous output will be duplicate. Some advantages of this cascades, chain like structure, are: 1. All system control by one master controller, it is easy to control. 2. The Blocking property system do not need to store temporary file. 3. That also means reduce the time on file communication.

Spark is a kind of advanced workflow system. It is more efficient on coping failure and group the task. Spark use RDD (Resilient Distributed Dataset) as the central data abstraction. RDD can be broke into chunks and hold by different compute nodes. Compared with regular MapReduce, it doesn't have restriction of elements type (such as key-value pairs). The RDD has two kinds of transformation which one is produce another RDD by some functions from previous RDD, another is generate from surrounding files. Spark has two important improvement which are RDD lazy evaluation and RDD lineage. The lazy evaluation means the execution will not start until an action is triggered (in Spark, which is transformations). The RDD resilience is a method that can tell system how to recreate or split RDD by recording the lineage of every RDD trace, but without a backup RDD storage.

Tensor Flow is a system developed for machine learning by google. The Major difference between Spark is TensorFlow pass the multidimensional matrix instead of RDD. An important advantage of TensorFlow data structure is the linear algebra operation can be used, which is good for machine learning model.

During the MapReduce based job, there are a lot of Recursion steps, and they can cause problem, for example when the job recovers from the failure, it cannot simply restart the failed tasks. There are three different approaches to deal with the failure while in recursive program. 1. Iterated MapReduce, which write recursion as repeated MapReduce jobs to handle failure by its initial implementation. 2. Spark: Spark has already included iterative statement, the failure management implemented by using lazy evaluation and lineage mechanism to deal with failure (mentioned above). 3. Bulk-synchronous system. Bulk-synchronous system is represented by Google's Pregel system, it uses graph-base model. The Bulk system view its data as a graph structure, each node is considered as task. It uses the check point mechanism to deal with recursion failure. Once fail happen, the system will restart form recent check point. It divides the job into several super steps.

Question 2, summarize Section 2.5:

This section is about a model, which is called Communication-Cost. It can evaluate the performance of algorithms on computing clusters. We all know that moving data between tasks can be very time consuming. So, it is important to reduce this cost.

In the acyclic network, the result of these algorithm activities might be used as input for the Map tasks. Such as the MapReduce algorithm, which cascades the output of MapReduce tasks.

There are three reasons why communication cost is the dominant cost. To begin with, each task's duty is straightforward. Second, communication speeds of computing cluster are slower than those of the CPU. Third, the time it takes to deliver data into main memory usually larger than the time it takes to operate on the input data once it is there.

Usually, the output of one task is the input of another task, and the output size is taken into account when calculating the input size. In general, the output size of an algorithm is much smaller than the input size and intermediate data. When calculating communication costs, we only calculate the inputs and not the outputs. If the output is larger than the input, aggregation must be used to lower the output size, which is usually done in reducer. The result will be transmitted to another collection in this case to complete the aggregate. As a result, the cost of transmission is always proportional to the cost of calculation.

Here is an example: If $R(A,B) \bowtie S(B,C)$ and the side of R is r, the size of S is s. Each chunk of the files containing R and S is supplied to a Map task, resulting in a total communication cost of r + s for all Map tasks. In general, each Map task will be run on a copy chunk of compute node. The Map tasks do not require any inter-node communication, but they must still read their data from disc. The total of the Map tasks' output is about equal to their inputs. Instead of memory-to-disk communication, communication from Map jobs to Reduce tasks is likely to happen through the cluster's interconnect. So, all map tasks costs are $O(r+s)$.

The time it takes for a parallel algorithm to complete we call it Wall-Clock. If all work is assigned to a single task to minimize communication costs, then the Wall-Clock will be extremely high. Given the number of compute nodes, the Wall-Clock time should also be as low as possible.

In order to study multiway join, this section sets out a general theory. Select some attributes in a multi-relational natural join and hash these attributes into buckets. Let the product of the count of buckets is k. Set a bucket count vector for the k reducers. Send each relation's tuples to all reducers that could have tuples to join with. Because we do not know some components of the vector, for all vectors with any value in these unknown components, the tuple is sent to reducers. So, the example is: If $R(A,B) \bowtie S(B,C) \bowtie T(C,D)$, and R, S, and T have sizes r, s, and t, and p is the probability that some of these tuples agree on one of the relations. $O(r + s + t + pst)$ is the final calculated result cost. The section also gives a method to join all relations at once.

For the join in 3 ways, the reducers should calculate the part of all relations. In example 2.16, the authors calculate that no more than a certain number of reducers can be used to achieve a 3 way join at a lower cost than a 2 way join.