Question 1:

The probability of any 2 people deciding to visit on any given day is:

$$\frac{1}{100} \times \frac{1}{100} = 0.0001$$

The probability of any 2 people will visit the same hotel on any given day is:

$$0.0001 \times \frac{1}{50 \times 10^4} = 2 \times 10^{-10}$$

⇒ The probability that 2 people will visit the same hotel at the same time on 4 different days is:

$$(2 \times 10^{-10})^4 = 1.6 \times 10^{-39}$$

The number of pairs of people: $\binom{5 \times 10^9}{2} = 1.25 \times 10^{19}$

The number of pairs of days: $\binom{5000}{4} = 2.601 \times 10^{13}$

Therefore, the expected suspects are:

$$1.25 \times 10^{19} \times 2.601 \times 10^{13} \times 1.6 \times 10^{-39} = 5.202 \times 10^{-7}$$

Exercise 2

Q1.

TF.IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

Its formulation is:

$$TF.IDF_{ij} = TF_{ij} \times IDF_i$$

which is the TF.IDF score for term (word) i in document j, where TF_{ij} is the term frequency of term (word) i in document j, and IDF_i is the Inverse Document Frequency of term (word) i across a set of documents.

Q2.

10000000 documents — IDF of a word that appears in

a. 40 docs. $IDF = \log\left(\frac{10\,000\,000}{40}\right) = \log(250\,000) \approx 17.931 \approx 18$

b. 10000 docs $IDF = \log\left(\frac{10\,000\,000}{10\,000}\right) = \log(1000) \approx 9.966 \approx 10$

Q3.

$$IDF_w = \log\left(\frac{10\,000\,000}{320}\right) = \log(31250) \approx 14.931 \approx 15$$

a) $TF_{wd} = \frac{1}{15}$

$\Rightarrow TF.IDF = \frac{1}{15} \times \log(31250) = \frac{1}{15} \times 15 = 1$

b) $TF_{wd} = \frac{5}{15} = \frac{1}{3}$

$\Rightarrow TF.IDF = \frac{1}{3} \times \log(31250) = \frac{1}{3} \times 15 = 5$

Exercise 3:

❖ The provided code is trying to get the number of occurrences of each word in each input set (100-0.txt in this case).

❖ Differences between standalone and pseudo-distributed modes:

⇒ Standalone mode does not use HDFS, it uses the local file system instead. However, in pseudo-distributed mode, HDFS is used.

⇒ With standalone mode, Hadoop is configured to run as a single Java process based on a non-distributed environment. There are no daemons used in this mode and everything runs in a single JVM instance. However, with pseudo-distributed mode, all daemons are operated on a single machine, therefore it simulates a cluster on a small scale.

⇒ In standalone mode, the jobs of the program run as if there is only one mapper and reducer. On the other hand, in the pseudo-distributed mode, there can be multiple mappers and reducers.

⇒ Standalone mode can be very useful for the development and debugging of the program, whereas pseudo-distributed mode is used as a mimicry to the cluster environment behaviour.

❖ Do you see any difference in the outputs of these modes?

- There is no difference in the outputs of these two modes. The outputs are the same since the two modes are used on the same Java file. However, while operating the pseudo-distributed mode for the project, the output is recorded in the task tracker's log files. This can be accessed easily by pointing the web browser to port 8088 of the server. For this project, the server will be localhost.

Exercise 4:

How many words are there with length 10 in FirstInputFile (pg100.txt)?

Answer: 11449

How many words are there with length 4 in FirstInputFile (pg100.txt)?

Answer: 205022

What is the longest length between words and what is its frequency in FirstInputFile (pg100.txt)?

Answer: Longest length is 33 and its frequency is 2

How many words are there with length 2 in SecondInputFile (3399.txt)?

Answer: 63500

How many words are there with length 5 in SecondInputFile (3399.txt)?

Answer: 35025

What is the most frequent length and what is its frequency in SecondInputFile (3399.txt)?

Answer: Most frequent length is 3 and its frequency is 72428

How many words are there with length 10 in FirstInputFile (pg100.txt)?

Answer: 2264

How many words are there with length 4 in FirstInputFile (pg100.txt)?

Answer: 1938

What is the most frequent length and what is its frequency in FirstInputFile (pg100.txt)?

Answer: The most frequent length is 7 and its frequency is 4931

How many words are there with length 5 in SecondInputFile (3399.txt)?

Answer: 1824

How many words are there with length 2 in SecondInputFile (3399.txt)?

Answer: 89

What is the second-most frequent length and what is its frequency in SecondInputFile (3399.txt)?

Answer: The second-most frequent length is 8 and its frequency is 2801

Exercise 5:

Q1: Summary for contents of Section 2.4 in our own words (approx. 600 words).

MapReduce produces extensions and modifications systems that are constructed on a distributed file system, can provide task management operations that are instantiations of functions and is able to incorporate methods to deal with failures during code execution without the need to restart from the beginning. These systems can be categorized into workflow systems like Spark and the working architecture of Tensorflow, which extend MapReduce by supporting acyclic networks of task-implemented functions, and systems that use a graph model of data such as Pregel, which sent messages from any node to any adjacent node.

Using an acyclic graph representing workflow among functions, the workflow systems oversee the extension of MapReduce from the assigned two-step workflow to any collection of functions. If a function has a single input for data, Map and Reduce functions will be applied to their input elements individually; whereas, if the function is given inputs from multiple files, elements from each file can be combined in methods such as union and relational join for them to be applied into the function. As the workflow functions can be executed by many tasks, the master controller is needed to divide the work among these tasks via hashing the input elements to deliver the output file of data for tasks on successor functions and have the blocking property to restart failed tasks

without duplicating output. This reduces storage in distributed file systems and minimizes communication.

Spark is an efficient engine in dealing with failure tolerance, tasks grouping among compute nodes and scheduling function execution and integration of the assigned programming language features and their function libraries. With Resilient Distributed Dataset (RDD). Some commonly used transformations in Spark include Map, Flatmap, Filter and Relational Database Operations like Join and GroupByKey, added with actions like Reduce.

Spark can be improved through lazy evaluation, which allows RDD to drop once used locally to avoid storing on disk or transmitted to another computer nodes, and lineage for RDD's to inform the system while reconstructing lost split and apply the transformations.

Unlike Spark, TensorFlow is the type of data that is passed between program steps. For RDD, TensorFlow uses multidimensional matrices called tensors to represent a model and training data to operate machine learning algorithms like gradient descent.

However, computations under the system are recursive, with is problematic for failure recovery, because of the lack of blocking property in recursive tasks to restart independently on failed tasks. As each task has an output that is produced to operate other tasks as input, no task will receive input if it requires the task to only produce output when the task ends. Therefore, other than simple restart, there are three failure management approaches for recursive programs, which are:

1) Iterated MapReduce like HaLoop, which rely on failure mechanism
2) Spark, which used lazy-evaluation and lineage mechanisms
3) Bulk-Synchronous Systems like Pregel, which views data as a graph and processes its task in the corresponding nodes through recursive algorithms on a computing cluster.

The computations are organized via supersteps, which for each superstep, it allows the processing of all received messages from nodes at the previous supersteps. This approach saves the orders of magnitude in the required time to output all the messages while communicating over a network because a task is assigned to bundle all messages received by its nodes and reduce large overheads in sending messages. It also checkpoints the computation after some supersteps as a recovery strategy to restart from the most recent checkpoint if a compute node fails. Since the system ensures that the average time between failures should exceed the time to recover from a failure, Pregel checkpoints after several supersteps to reduce the probability of a failure during that computation.

Q2: Summary for contents from Section of 2.5 in our own words (approx. 600 words).

The Communication-Cost Model provides a measurement benchmark to assess the quality of the implemented algorithms on a computing cluster. The communication cost for a computational task is calculated from the input size based on bytes or the number of tuples in relational database operations. Alternatively, the algorithm’s communication cost is the sum of the communication cost of all tasks that used the algorithm. The significance in measuring efficiency using communication cost lies in the simple, linear algorithms computed by each task, the impact of interconnecting speed for a computing cluster and the time taken to retrieve data into the computer’s main memory.

Besides communication cost, wall-clock time, a term which is defined as the time it takes to complete a parallel algorithm, also holds importance in deciding the algorithms used in a cluster computing environment. This is because when all the work are assigned to one task, it will increase

the wall- clock time and therefore, an inefficient algorithm despite minimizing the communication cost.

Upon examination on a multiway join to analyse the impact of communication cost in choosing the ideal algorithm, it requires the selection of defining attributes from the relations involved in the natural join. At least three relations of the natural join are required to have their values hashed to buckets. The selection of the number of buckets for each of these attributes, which is constrained by the product from the numbers of buckets for each attribute and the number of reducers that will be used, the identification of each of the reducers with a vector of bucket numbers, and finally the dispatchment of relational tuples to all reducers, in which tuples can be identified to join with and apply the hash function to those values. This helps to determine vector components that identifies the reducers.

Given an example of join L (D, E) ⊳⊲ M (E, F) ⊳⊲ N (F, G), suppose the relations L, M and N have sizes in their respective lower-case characters, added with p as the probability that L-tuple and M-tuple agree on E. This is followed by another step, where M-tuple, and a N- tuple also need to agree on F. For this case, there are three ways to perform the join, which are:

1) Joining the second component from L tuple and the first component of M tuple, followed by joining the outcome with N tuple in the second job. This finalised the communication cost as $O(l + m) + O(n + plm) = O(l + m + n + plm)$.
2) If M and N is joined first, the communication cost would be $O(l + m + n + pmn)$.
3) Use a single MapReduce job to join all the relations.