**CPRE/SE 419 Software Tools for Large Scale Data Analytics**

**Spring 2023**

**Quiz**

**(Take-home) Due: Saturday, March 25 (11:59PM)**

**Part I – Provide brief/concise answer:**

1. **(7 pts.) Would you say that MR is a good paradigm for interactive analysis of large datasets? Briefly justify your answer.**

No, MR is not a good paradigm for the interactive analysis of large datasets. MapReduce has high latency because it was designed for its batch processing nature and this makes it unsuitable for real-time or low-latency queries.

1. **(7 pts.) Although physically distributed in blocks, when processing an MR job the HDFS files are divided in (logical) Splits. What is it that determines the number of Splits?**

The number of Splits in the block is determined by the size of the input and the Hadoop configuration parameter of *mapreduce.input.fileinputformat.split.minsize* or *mapreduce.input.fileinputformat.split.maxsize* control the min and max size of the Split.

1. **(7 pts.) Describe briefly the concept of combiner in MR.**

A combiner is an optional optimization step that aggregates intermediate key-value pairs generated by the Map phase, reducing the amount of data sent to the Reduce phase which essentially improves performance.

1. **(7 pts.) Does Pig Latin require the use of types when defining a schema (e.g., for loading the data)?**

Pig Latin doesn’t require the use of types when defining a schema since it’s a dynamically typed language that infers the schema of the data at runtime. Specifying types is recommended to help avoid errors and also to improve performance.

1. **(9pts) What are the advantages and disadvantages of using distributed lock management for concurrency control in distributed databases. Does Hadoop suffer from any of those?**

Advantages of using distributed lock management for concurrency control in distributed databases would include scalability, consistency and performance.

Some disadvantages would be overhead, deadlocks and complexity. The use of distributed lock management can introduce additional overhead in terms of communication and coordination among nodes, which can negatively impact performance. It can be susceptible to deadlocks, where multiple transactions are blocked waiting for locks that are held by other transactions. And it can add complexity to the implementation, requiring additional code and management to ensure that locks are properly acquired and released.

Hadoop doesn’t suffer from these issues because it uses a write once, read many model which avoids the need for locks and concurrency control.

**Part II**

1. **(13 pts.) It is well known that if the probability of a disk failure (within a given time-frame) is p, then the probability of a disk failing from among a collection of M of them is surely greater than p. Then, why is it that Hadoop uses multiple nodes to store the data/files?**

Hadoop uses multiple nodes to store the data/files because of fault tolerance, load balancing, parallel processing and scalability. Fault tolerance is a reason because replicating data across multiple nodes, Hadoop can recover from disk or node failures. If one disk or node fails, the system can retrieve the data from another replica. Load balancing is a reason as distributing data and tasks across nodes ensures that the workload is evenly distributed which in turn leads to better performance and avoids bottlenecks. Parallel processing because Hadoop can process data in parallel by dividing tasks among multiple nodes which increases data processing and analysis. Finally, scalability is another reason because Hadoop can store and process large amounts of data by distributing it across multiple, allowing the system to grow by adding more nodes as needed. So even though there is higher probability of a single disk failure with multiple nodes, Hadoop’s mechanisms ensure data availability and reliability.

1. **(15 pts.) Suppose that a given file has k blocks, and each block is stored on a different machine (no replication – single copy of each block). Assume that the probability of a machine failure is p, and that failures of different machines are independent of each other. What is the probability that the file is lost due to machine failure?**

Since the probability of a machine failure is p, the probability of a machine not failing is (1-p). Because failures are independent, the probability that all of them do not fail is (1-p)k where k is machines storing the file’s blocks. Therefore, the possibility of the file being lost due to machine failure is 1-(1-p)k

**Part III – Coding questions**

1. **(19 pts.) Out-of-Gap-Average: The average of a finite set of numbers is defined as the sum of all numbers in the set divided by the size of the set. Suppose that we are given a (large) set of numbers on a file in HDFS, with one number per line. The objective is to determine the average value of the numbers that are closer to the extreme values in the input, without considering the impact of the numbers that are in some gap “in-between”. Write the MR pseudo-code of an efficient algorithm to compute the OGA (Out-of-Gap-Average) of a given set of input values – i.e., the average of all the input values which are \*not\* inside a given interval [gap\_lower\_bound, gap\_upper\_bound].**

The algorithm would work by first defining a mapper than reads in each line of the input file, parses the double value and checks whether the value falls inside the specified gap. If the value is inside the gap, it is skipped, otherwise it is emitted as the output with a NullWritable key. The reduced then receives all non-gap values and computes their average OGA by summing them up and dividing by the count. Finally, the result is written to the output file. The gap bounds are passed as configuration parameters to the job.

Pseudocode:   
// Map function processes each line (number) in the input file

function map(line):

// Convert the line to a float

number = float(line)

// Emit a key-value pair with a fixed key and a tuple containing the number and count 1

emit("key", (number, 1))

// Reduce function processes the key-value pairs emitted by the map function

function reduce(key, values):

// Initialize the sum and count of numbers outside the gap

sum\_numbers = 0

count\_numbers = 0

// Iterate through the values

for value in values:

// Unpack the number and count from the value tuple

number, count = value

// Check if the number is outside the gap

if number < gap\_lower\_bound or number > gap\_upper\_bound:

// Update the sum and count of numbers outside the gap

sum\_numbers += number

count\_numbers += count

// Calculate the average of numbers outside the gap

average = sum\_numbers / count\_numbers

// Emit the result with the label "Out-of-Gap-Average"

emit("Out-of-Gap-Average", average)

1. **(21 pts.) Assume that you are given a large file named sales\_data.txt in which the records are of the format: (time, sales, item, store). Write a pseudo-code for a PigLatin program which, for each store and for each year (per store) will calculate the average sales, but only for the items that cost > $55. (You can assume that the time-stamps are at the year-level of granularity)**

-- Load the sales data from the input file

sales = LOAD 'sales\_data.txt' USING PigStorage('\t') AS (time:chararray, sales:float, item:chararray, store:chararray);

-- Filter the data to only include items that cost over $55

sales\_filtered = FILTER sales BY sales > 55;

-- Extract the year from the timestamp

sales\_filtered\_year = FOREACH sales\_filtered GENERATE SUBSTRING(time, 0, 4) AS year, sales, item, store;

-- Group the data by store and year

sales\_grouped = GROUP sales\_filtered\_year BY (store, year);

-- Calculate the average sales for each store and year

average\_sales = FOREACH sales\_grouped GENERATE FLATTEN(group) as (store, year), AVG(sales\_filtered\_year.sales) as average;

-- Store the results in an output file

STORE average\_sales INTO ‘output’ USING PigStorage(‘\t’);