Heidelberg University
Winter semester 2017/2018

Group Parallel and Distributed Systems (PVS)
Artur Andrzejak, Diego Costa

Problem Set 10 for lecture Mining Massive Datasets

Due January 15, 2018, 11:59 pm

Exercise 1 (1 point)

Suppose you have to select hash functions for a Bloom filter algorithm. Propose 3 hash functions which are as independent from each other as possible and briefly justify your choice.

Exercise 2 (2 points)

Consider a Bloom filter with a bit array of $n = 5$ bits.

a) Compute the probability that a random element gets hashed by a hash function to a given bit in the bit array by using the probability formula from lecture 11 (slide *Analysis: Throwing Darts (2)*).
Now we want to use the $k = 2$ hash functions
$h_1(x) = x \mod 5$,
$h_2(x) = 2 \cdot x + 3 \mod 5$.
Is every bit equally likely to be hit by one of the two hash functions? Suppose the numbers 4 and 1 are in the set that we want to find with the filter. Show the state of the bit array after you apply the hash functions to these numbers.

b) Consider again the hash functions and the bit array state as in **a)**. How likely is it for a number in the stream to be a false positive? Use the false positive probability formula from lecture 11 (slide *Bloom Filter - Analysis*).

Exercise 3 (2 points)

Consider the Bloom filtering technique. Calculate the expected false-positive rate for a bit-array of 8 billion bits, 1 billion members of the set S (i.e., keys that should not be filtered from the stream), and three hash functions. What if we use four hash functions?

Exercise 4 (2 points)

Suppose we have a stream of tuples with the schema `Grades(university, courseID, studentID, grade)`. Assume universities are unique, but a courseID is unique only within a university (i.e., different universities may have different courses with the same ID, e.g., "IPI") and likewise, studentIDs are unique only within a university (different universities may assign the same ID to different students). Suppose we want to answer certain queries approximately from a 1/20th sample of the data. For each of the queries below, indicate how you would construct the sample. That is, tell what the key attributes should be.

a) For each university, estimate the average number of students in a course.

b) Estimate the fraction of students who have an average grade of 2.0 or better.

c) Estimate the fraction of courses where at least half of the students got the grade 1.7 or better.

Exercise 5 (3 points)

Analyse the NetworkWordCount example from the Spark Streaming programming guide[1]. Run the sample code as described in the guide:

```
run-example streaming.NetworkWordCount localhost 9999
```

Then, in another terminal window, run the following commands:

```
mkfifo fifo
cat > fifo &
nc -lk 9999 < fifo &
```

Now the netcat process is running in the background, listening on the named pipe `fifo`. All that is sent through this named pipe will be forwarded by netcat to port 9999, where in turn the Spark Streaming program is listening. When you forward the standard output of any process with the help of the `>` operator to `fifo`, this output will be streamed to the NetworkWordCount program.

a) Run the command

```
echo "A small step for a man, a giant leap for mankind." \
    > fifo
```

and describe what happens. Does the output of the streaming program behave as expected?

b) Run the command

```
yes "k thx bai bai" > fifo
```

What do you notice? How big is roughly the throughput when you assume one byte per character?

Exercise 6 (6 points)

This exercise is the third in the series of tasks related data processing with Spark. You should ideally reuse the implementation developed in the previous problem sets.

At this point, every combination of <InstanceType>, <ProductDescription>, <AvailabilityZone> are saved in ts own *price timeseries* file, containing only the <Timestamp, Price> of the reported price instances. In this exercise, you will implement subroutines to assess and report the quality of those timeseries on two aspects: coherency and completeness. For each of the following subtasks submit your code as a part of the solution.

a) Coherency. As required by the last data processing task, each *price timeseries* file is free of duplicates[2], but it might still contain multiple distinct prices reported in the same timestamp: an inconsistency. Implement a subroutine `checkCoherency(S)` that reads a *price timeseries* S from disk, and verifies if there are multiple distinct prices reported on the same Timestamp. Return the inconsistent tuples found in this coherency check.

[1] https://spark.apache.org/docs/latest/streaming-programming-guide.html

[2] Make sure your implementation of the function saveTimeseries (from Problem Set 8) only removes duplicates of the tuple <Timestamp, Price>, but retains all different prices (found in the original input) for the same timestamp.

b) Completeness. Implement a subroutine `checkCompleteness(S, T)`, that reads a *price timeseries* S from disk, and checks whether there are any gaps in the time of the reported instances price. A gap is defined as an absence of price report in a period greater than T minutes. Any gap larger than T (between two price events) should be reported in the format: <price-series-name> <timestamp-gap-start> <timestamp-gap-end>.

c) To test your implementation, run both checkCoherency and checkCompleteness (use $T = 24 * 60$, one day) using the entire Amazon EC2 dataset. Generate a report containing per each timeseries the **number of found inconsistencies** by each check. Submit as your solution only the report on the timeseries related to *prices-ca-central* availability zone.