Exercise 1 : Federal Statistical Office

You have been recently hired as a Spark consultant at the Federal Statistical Office in Neuchâtel. They have recently started using Spark for all their data processing tasks, but for some reason, they are not entirely satisfied with Spark thus far.

Below is the datatype they use to record information about people living in Switzerland. The `RDD` that contains all records (several millions) is called `people`.

```
case class Person(age: Int, salary: Long, town: Int)
val people: RDD[Person] = ???
```

The Spark cluster they have put in place consists of 8 identical machines, each of which has 4 cores.

Question 1.1

On your very first day, you are immediately tasked with investigating some piece of code. Even though the statistician that wrote the code is absolutely sure it is correct, he is not satisfied with its performance! It's awfully slow!

```
people.groupBy(_.age).map {
  case (age, peopleOfAge) =>
    val salaries = peopleOfAge.map(_.salary)
    val n = peopleOfAge.size
    val mean = salaries.sum / n
    val variance = salaries.map(x => (x - mean) * (x - mean)).sum / n

    (age, (mean, Math.sqrt(variance).toLong))
}
```

Using your own words, explain why the above code is supposed to do and why it is not as efficient as it could be. What happens?

Hint: *You may find it useful to draw a graphical representation of the nodes and show data exchanges between the nodes.*

Answer

A lot of unnecessary communication happens on the cluster. Data is moved over the network from nodes to other nodes, which is called "shuffling". Network communication is extremely slow relatively to other operations.

Question 1.2

Rewrite the above piece of code to be as efficient as possible.

Hint: You may recall from your Probabilities and statistics *course that there exist multiple formulas to compute the variance of a series* x *of* n *values with mean* m. *For instance:*

$$Var(x) = \sum_{i=1}^{n} (x_i - m)^2 / n \qquad \text{or} \qquad Var(x) = (\sum_{i=1}^{n} x_i^2) / n - m^2$$

One of the two forms might be more appropriate in your solution.

Answer

```
people.map(p => (p.age, p.salary))
      .mapValues(s => (1, s, s * s))
      .reduceByKey {
        case ((n1, s1, ss1), (n2, s2, ss2)) =>
          (n1 + n2, s1 + s2, ss1 + ss2)
      }
      .mapValues {
        case (n, sum, sumSquares) =>
          val mean = sum / n
          val variance = sumSquares / n - mean * mean
          (mean, Math.sqrt(variance).toLong)
      }
```

Note the use of reduceByKey instead of groupByKey. Also, mapValues is used whenever possible, which preserves the partitioner, if any. This will be useful later on.

To see the difference in running time between reduceByKey and groupByKey, you may have a look at the following talk: https://www.youtube.com/watch?v=0KGGa9qX9nw (slides: https://www.slidesearch.net/slide/beyond-shuffling-scala-days-berlin-2016).

Question 1.3

Could the computation be done even faster if the data was already partitioned ? Write the code to partition the data, using a `Partitioner`, to improve data locality.

```scala
// Code to partition the data before processing
val pairs = people.map(p => (p.age, p.salary))
val nPartitions = 32  // 8 machines, 4 cores each
val tunedPartitioner = new RangePartitioner(nPartitions, pairs)
val partitioned = pairs.partitionBy(tunedPartitioner).persist()


// NOTE: Once partitioned, we can compute mean and variance as before,
// using the partitioned RDD.
partitioned.mapValues(s => (1, s, s * s))
           .reduceByKey {
             case ((n1, s1, ss1), (n2, s2, ss2)) =>
               (n1 + n2, s1 + s2, ss1 + ss2)
           }
           .mapValues {
             case (n, sum, sumSquares) =>
               val mean = sum / n
               val variance = sumSquares / n - mean * mean
               (mean, Math.sqrt(variance).toLong)
           }
```

Exercise 2 : Partitioners

Question 2.1

What is a partitioner? Come up with **two reasons** why you would want to repartition data.

Answer

Partitioners specify which keys are hosted by the different partitions. Repartitioning in useful for example for:

1) *Improving data locality, and thus avoiding network shuffles.*
2) *Balance the work between the different partitions.*

Question 2.2

Which of the following transformations preserve the partitioner of the parent RDD, if any?

- `map`
- `mapValues`
- `filter`
- `flatMap`
- `flatMapValues`
- `join`
- `reduceByKey`
- `groupByKey`

Within your group, discuss what makes it possible for some transformations to preserve and propagate the parent's partitioner.

Answer

*The transformations that preserve partitioners are in **bold** below.*

- *`map`*
- ***`mapValues`***
- ***`filter`***
- *`flatMap`*
- ***`flatMapValues`***
- ***`join`***
- ***`reduceByKey`***
- ***`groupByKey`***

The partitioner can be preserved because the set of keys held by a partition in the resulting RDD is a subsets of the keys the partition held in the parent RDD. Therefore the partitioner still faithfully describe where the different keys are held.

Question 2.3

Which of the following transformations will return an RDD with a partitioner, even when the parent doesn't have one?

- `map`
- `mapValues`
- `filter`
- `flatMap`
- `flatMapValues`
- `join`
- `reduceByKey`
- `groupByKey`

Answer

The transformations that introduce partitioners are in ***bold*** *below.*

- *`map`*
- *`mapValues`*
- *`filter`*
- *`flatMap`*
- *`flatMapValues`*
- ***`join`***
- ***`reduceByKey`***
- ***`groupByKey`***

Question 2.4

What is the difference between a `HashPartitioner` and a `RangePartitioner`? When would you use one over the other?

Answer

A `HashPartitioner` partitions the data according to the hashcode of the key, while a `RangePartitioner` partitions the data according to an ordering on the keys.

`HashPartitioners` generally split the work evenly between the different partitions. This partitioners does not require a specific ordering to exist on the keys.

`RangePartitioners` allow grouping keys in the same range on the same partition. This can be useful to further improve data locality. In some cases, as seen in the video lecture, it also allows for better work balancing.