Universität Heidelberg
Wintersemester 18/19

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

Problem Set 4 for lecture Distributed Systems I (IVS1)

Due: 20.11.2018, 14:00 Uhr

Exercise 1 (6 Points)

Take a look at a dataset *prices-ap-northeast-2-2017-11-10* from Moodle, which contains prices of so-called spot instances from Amazon Elastic Compute Cloud (EC2) service. This dataset contains prices collected in the past *four* weeks for one availability zone, and inside of the compressed file there is a tab-delimited text file with the structure

<Type>|t<Price>|t<Timestamp>|t<InstanceType>|t<ProductDescription>|t<AvailabilityZone>

containing the prices of EC2 spot instances of the last 100 days.

a) Implement a subroutine in Python/Spark which takes as an argument a file name of a compressed (*.gz) file with the structure as described above, reads its content into a new Spark dataframe, and returns a reference to this dataframe. To this aim define an appropriate schema with a correct name and (memory-efficient) data type for each column (e.g. use *TimestampType*() and not *StringType*() for column *Timestamp*, see also https://goo.gl/rkxENJ). The resulting dataframe should **not** contain the column *Type* (as each row has the value "SPOTINSTANCEPRICE"). Submit your code as the solution.

b) To test your implementation, read the file *prices-ap-northeast-2-2017-11-10.txt.gz* into a dataframe, and then calculate and output the average price per each combination *InstanceType*/*ProductDescription*.

Exercise 2 (3 Points)

Watch the presentation about *Advanced Spark Features* by Matei Zaharia[1] and answer the following questions:

a) What does broadcast provide? Which other mechanism does it improve and how? Which features of the distributed program determine the number of times the variable will be actually transmitted over the network? Explain the role of tasks and nodes in this.

b) Describe the function of an accumulator. What is the alternative implementation without an accumulator and why is an accumulator a preferred option? Which example application for an acculumator is discussed in the video? What has to be implemented in order to define a custom accumulator? Compare the accumulator mechanism to the `reduce()`-function.

c) Give three examples of RDD operators that result in RDDs with partitioning. Explain the connection between partitioning and network traffic. How does the modification on the PageRank example use partitioning to make the code more efficient? How does Spark exploit the knowledge about the partitioning to save time in task execution? How can you create a custom Partitioner?

[1] *Advanced Spark Features* by Matei Zaharia: Slides, Video

Exercise 3 (3 Points)

Analyse the NetworkWordCount example from the Spark Streaming programming guide[2]. 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 4 (4 Points)

In the video „*A Deeper Understanding of Spark Internals*“ Aaron Davidson describes aspects of Spark implementations that deeply impacts performance of underlying applications. Watch the video and answer the questions below:

1. Describe each step of Spark execution model.
2. In the execution phase, Spark tries to pipeline operations as much as possible. How does pipelining affect performance? Give examples of operations that can be pipelined.
3. List the four most common issues described by Aaron. What is the recommended setting and guidelines to deal with the problems described in the talk?
4. Transcribe the code given in the talk into the language of your choice (Scala, Java or Python). Download the list of last names (~180MB after extraction) provided in Moodle and use it to evaluate the performance of both versions in your local machine. Experiment with the number of partitions in the second version: what number of partition yield the fastest results on your computer?

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

Exercise 5 (4, Bonus Points)

K-Means is one of the most popular clustering algorithms in data mining. Take a look in the aditional slides to understand how k-means work and how to implement it in Spark. The implementation uses a for-loop which executes several identical operations for each of the k clusters. Improve the code by using Spark group operations instead of the for-loop.

a) Analyse the k-means code in Spark repository[3]. Describe the iteration loop of this code, especially how `reduceByKey()` is used.

b) Rewrite the above code, using `groupByKey()` (and possibly other operations) to achieve the same result. Test the correctness. What is the scalability of this solution? Submit your code and explanations of it as a solution.

[3] `https://github.com/apache/spark/blob/master/examples/src/main/python/kmeans.py`