Heidelberg University
Distributed Systems 1 (IVS1)
Winter Semester 2018/19

Duc Anh Phi,
Michael Tabachnik,
Edgar Brotzmann

Solutions to Problem Set 3

Due: 13.11.2018, 2pm

Exercise 1

1.Output the biggest of the input numbers:

```
reduce:
    current_max = 0
    for each input_value:
        current_max = max(current_max, input_value)
```

2. Output the geometric mean of the input numbers:

```
map:
    value to (value, 1)
reduce:
    product = 1
    count = 0
    for each input_pair:
        product = product * input_pair[0]
        count = count + input_pair[1]
map:
    value to value[0]^(1/value[1])
```

Heidelberg University
Distributed Systems 1 (IVS1)
Winter Semester 2018/19

Duc Anh Phi,
Michael Tabachnik,
Edgar Brotzmann

3. Output the input set of numbers, but without duplicates:

```
reduce:
    input_set = {}
    for each input_value:
        if input_value not in input_set:
            add value to input_set
```

4. Output the size of the input set (ignoring muliplicity):

```
map:
    value to 1
reduce:
    count = 0
    for each input_value:
        count = count + input_value
```

5. Output the frequencies of a word given from a huge text file:

```
map:
    value to 1 if value is the given word, else value to 0
reduce:
    count = 0
    for each input_value:
        count = count + input_value
```

Heidelberg University
Distributed Systems 1 (IVS1)
Winter Semester 2018/19

Duc Anh Phi,
Michael Tabachnik,
Edgar Brotzmann

Exercise 2

1.

Cluster computing framework like MapReduce and Dryad have been widely adopted. However, they lack abstractions for leveraging distributed memory, which is crucial for the efficient computation of applications which reuse intermediate results across multiple computations.

2.

Distributed Shared Memory offer an interface based on fine-grained updates to mutable state (e.g. cells in a table). With this interface the only ways to provide fault tolerance are to replicate the data across machines or to log updates across machines. Both approaches are expensive for data-intensive workloads, as they require copying large amounts of data over the cluster network or to log updates across machines.

RDDs provide interface based on coarse-grained transformations that apply the same operation to many data items. This allows them to efficiently provide fault tolerance by logging the transformations used to build a dataset rather than the actual data.

Heidelberg University
Distributed Systems 1 (IVS1)
Winter Semester 2018/19

Duc Anh Phi,
Michael Tabachnik,
Edgar Brotzmann

<u>Exercise 2</u>

2.

The main difference between RDDs and DSM is that RDDs can only be created (‘written’) through coarse-grained transformations, while DSM allows reads and writes to each memory location. This restricts RDDs to applications that perform bulk writes.

RDDs would be less suitable for applications that make asynchronous fine-grained updates to shared state, such as a storage system for a web application or an incremental web crawler. RDDs are best suited for batch applications that apply the same operation to all elements of a dataset.

3.

RDDs are represented through a common interface that exposes five pieces of information:

- a set of partitions, which are atomic pieces of the dataset;
- a set of dependencies on parent RDDs;
- a function for computing the dataset based on its parents;
- and metadata about its partitioning scheme and data placement.

Dependencies are classified into two types:

- narrow, where each partition of the parent RDD is used by at most one partition of the child RDD
- wide, where multiple child partitions may depend on it

Heidelberg University
Distributed Systems 1 (IVS1)
Winter Semester 2018/19

Duc Anh Phi,
Michael Tabachnik,
Edgar Brotzmann

Exercise 3

1.

join(*otherDataset*, [*numTasks*]):
When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key.

Example:
input:

```
    firstNames = [(0, Alex), (1, Anton), (2, Bob), (3, Chandler)]
    lastNames = [(0, Armin),(1, Brech),(1, Krug),(2, Kraft),(3, Bing)]
```

output:

```
    firstNames.join(lastNames):
        [
            (0, (Alex, Armin)),
            (1, (Anton, Brech)),
            (1, (Anton, Krug),
            (2, (Bob, Kraft)),
            (3, (Chandler, Bing)
        ]
```

Heidelberg University
Distributed Systems 1 (IVS1)
Winter Semester 2018/19

Duc Anh Phi,
Michael Tabachnik,
Edgar Brotzmann

<u>Exercise 3</u>

1.
sort():
There is no sort() function for RDD's (only for Dataframes), however the **sortby(**keyfunc, ascending=True, numPartitions=None**)** function sorts the RDD using a key function.

Example:
input:
```
        letters = [('c', 1),('a', 3),('d', 6),('e', 15),('b', 10)]
```

output:
```
        letters.sortby(lambda letter: letter[0]):
                [('a', 3),('b', 10),('c', 1),('d', 6),('e', 15)]
        letters.sortby(lambda letter: letter[1]):
                [('e', 15),('b', 10),('d', 6),('a', 3),('c', 1)]
```

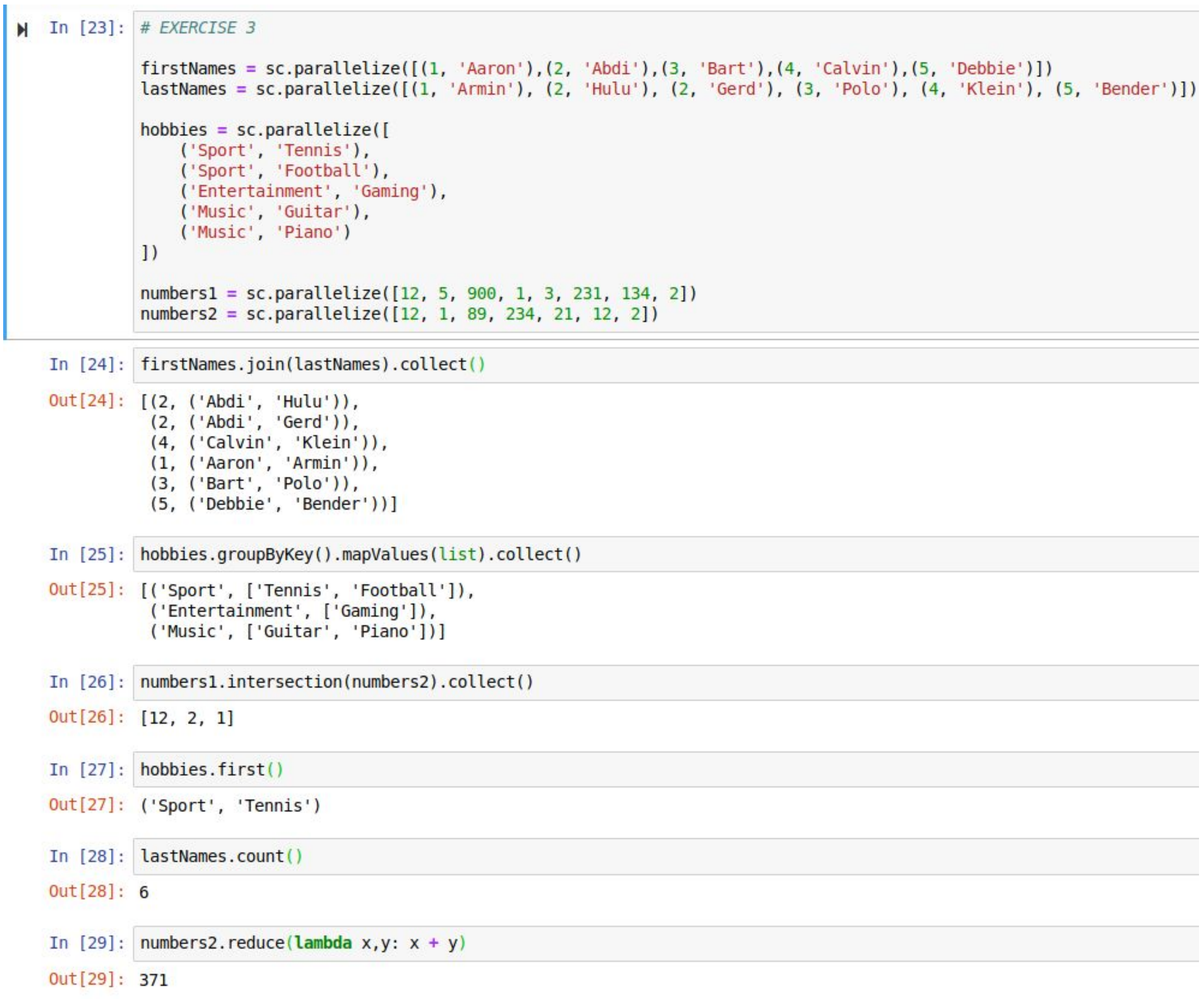
groupBy(f, numPartitions=None, partitionFunc=<function portable_hash>)
Creates keys for each entry using function f, then groups all entries with the same key as the value to that key

Example:
input:
```
        numbers = [1, 2, 3, 4, 5, 6, 7, 8]
```
output:
```
        numbers.groupBy(lambda num: num % 2):
                [(0, [2, 4, 6, 8]), (1, [1, 3, 5, 7])]
```

Heidelberg University
Distributed Systems 1 (IVS1)
Winter Semester 2018/19

Duc Anh Phi,
Michael Tabachnik,
Edgar Brotzmann

2.



```
In [23]: # EXERCISE 3

         firstNames = sc.parallelize([(1, 'Aaron'),(2, 'Abdi'),(3, 'Bart'),(4, 'Calvin'),(5, 'Debbie')])
         lastNames = sc.parallelize([(1, 'Armin'), (2, 'Hulu'), (2, 'Gerd'), (3, 'Polo'), (4, 'Klein'), (5, 'Bender')])

         hobbies = sc.parallelize([
             ('Sport', 'Tennis'),
             ('Sport', 'Football'),
             ('Entertainment', 'Gaming'),
             ('Music', 'Guitar'),
             ('Music', 'Piano')
         ])

         numbers1 = sc.parallelize([12, 5, 900, 1, 3, 231, 134, 2])
         numbers2 = sc.parallelize([12, 1, 89, 234, 21, 12, 2])

In [24]: firstNames.join(lastNames).collect()

Out[24]: [(2, ('Abdi', 'Hulu')),
          (2, ('Abdi', 'Gerd')),
          (4, ('Calvin', 'Klein')),
          (1, ('Aaron', 'Armin')),
          (3, ('Bart', 'Polo')),
          (5, ('Debbie', 'Bender'))]

In [25]: hobbies.groupByKey().mapValues(list).collect()

Out[25]: [('Sport', ['Tennis', 'Football']),
          ('Entertainment', ['Gaming']),
          ('Music', ['Guitar', 'Piano'])]

In [26]: numbers1.intersection(numbers2).collect()

Out[26]: [12, 2, 1]

In [27]: hobbies.first()

Out[27]: ('Sport', 'Tennis')

In [28]: lastNames.count()

Out[28]: 6

In [29]: numbers2.reduce(lambda x,y: x + y)

Out[29]: 371
```

Heidelberg University
Distributed Systems 1 (IVS1)
Winter Semester 2018/19

Duc Anh Phi,
Michael Tabachnik,
Edgar Brotzmann

Exercise 4

1.

In contrast to writing Spark applications, the SparkSession has already been created for you so that you can just start working and not waste valuable time on creating one. A SparkSession is the main entry point for Spark functionality: it represents the connection to a Spark cluster and or can use it to create RDDs and to broadcast variables on that cluster.

2.

The **collect()** action should be avoided on large datasets because it returns **all** elements of a theoretically infinitely large dataset which is costly to compute and would exceed the systems memory.
Alternatively you can use **take(n)** action to get the first n elements of the dataset.
You can also check the number of elements in the dataset with **count()**.

3.

Dataframes is an abstraction which imposes a schema on unstructured data and represents it like a table in a database.
Dataframes enable high-level expressions, to perform SQL queries to explore your structured data further. Furthermore Dataframes are more performant than RDDs.