

732A54/TDDE31 Big Data Analytics

Exercise Session

Huanyu Li

Agenda

- Aims of this exercise session
- Review
 - ✓ Map-Reduce: Working with key-value pairs
 - ✓ Lambda functions
- How to design and write PySpark code
- Lab introduction and exercises
 - ✓ Conceptual design
 - ✓ Write PySpark code
- *How to work on Sigma

Aims

- Give you an overview of the labs
- Help you to understand how to design and write code using Spark in python
- Exercises: start to solve the assignments in the labs

Map-Reduce: Working with key-value pairs

- Data elements: key-value pairs
- Python's tuple structure fit this key-value pair: (key, value)
 - ✓ (1,2), (1,3), (1,4), (1,5), (2,2), (2,3)
- A tuple is a sequence of immutable Python objects
 - (*'a'*, 3), (1, (3, 4)), ((1,1), 2)
- Accessing elements done with [index]
 - $x = (3, ('c', [1])), y = ((3, 'a'), ('c', [1]))$
 - $x[0] = 3, x[1] = x[-1] = ('c', [1]), x[1][1] = [1]$
 - $y[0] = (3, 'a'), y[0][0] = 3, y[1][1] = [1]$
- 'Shuffle' operations by a key work on RDDs containing built-in Python tuples
 - ✓ 'repartition' operations
 - ✓ 'byKey' operations
 - ✓ 'join' operations

Lambda functions– a way to pass function to a RDD operation

➤ General form

lambda arguments: expression

➤ Examples:

*lambda a: 2 * a*

–double the argument *a*

lambda a, b: a + b

–produce the sum of arguments *a* and *b*

lambda input_list : (input_list[0], input_list[1])

lambda input_list : max(input_list)

RDD - Operations

Transformations	$map(f : T \Rightarrow U) : RDD[T] \Rightarrow RDD[U]$ $filter(f : T \Rightarrow Bool) : RDD[T] \Rightarrow RDD[T]$ $flatMap(f : T \Rightarrow Seq[U]) : RDD[T] \Rightarrow RDD[U]$ $sample(fraction : Float) : RDD[T] \Rightarrow RDD[T]$ (Deterministic sampling) $groupByKey() : RDD[(K, V)] \Rightarrow RDD[(K, Seq[V])]$ $reduceByKey(f : (V, V) \Rightarrow V) : RDD[(K, V)] \Rightarrow RDD[(K, V)]$ $union() : (RDD[T], RDD[T]) \Rightarrow RDD[T]$ $join() : (RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (V, W))]$ $cogroup() : (RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (Seq[V], Seq[W]))]$ $crossProduct() : (RDD[T], RDD[U]) \Rightarrow RDD[(T, U)]$ $mapValues(f : V \Rightarrow W) : RDD[(K, V)] \Rightarrow RDD[(K, W)]$ (Preserves partitioning) $sort(c : Comparator[K]) : RDD[(K, V)] \Rightarrow RDD[(K, V)]$ $partitionBy(p : Partitioner[K]) : RDD[(K, V)] \Rightarrow RDD[(K, V)]$
Actions	$count() : RDD[T] \Rightarrow Long$ $collect() : RDD[T] \Rightarrow Seq[T]$ $reduce(f : (T, T) \Rightarrow T) : RDD[T] \Rightarrow T$ $lookup(k : K) : RDD[(K, V)] \Rightarrow Seq[V]$ (On hash/range partitioned RDDs) $save(path : String) : \text{Outputs RDD to a storage system, e.g., HDFS}$

Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.

➤ You need more than the above to solve all assignments in the lab.

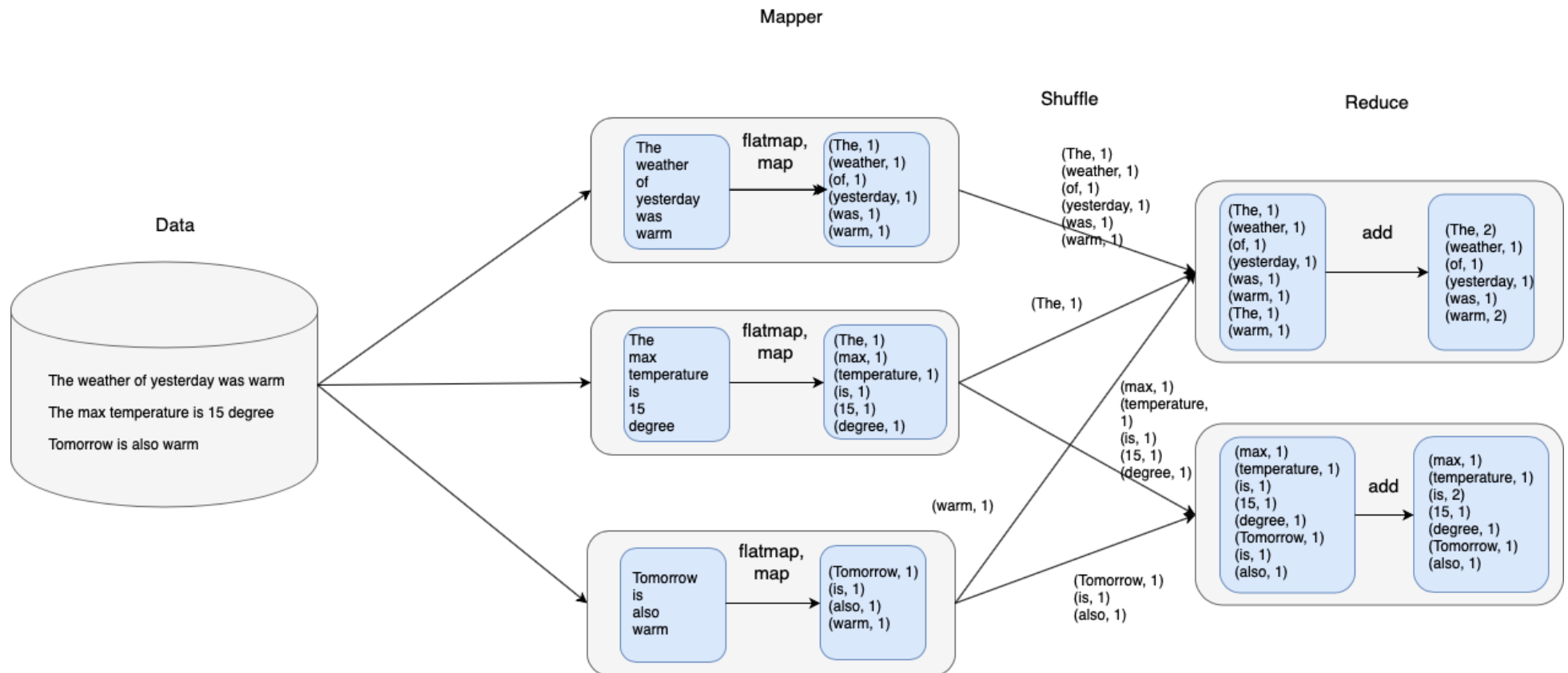
- ✓ PySpark library: <https://spark.apache.org/docs/2.4.3/api/python/index.html>
- ✓ Spark 2.4.3 RDD programming guide: <https://spark.apache.org/docs/2.4.3/rdd-programming-guide.html>

Agenda

- Aims of this exercise session
- Review
 - ✓ Map-Reduce: Working with key-value pairs
 - ✓ Lambda functions
- **How to design and write PySpark code**
- Lab introduction and exercises
 - ✓ Conceptual design
 - ✓ Write PySpark code
- *How to work on Sigma

Word Count – Conceptual Design

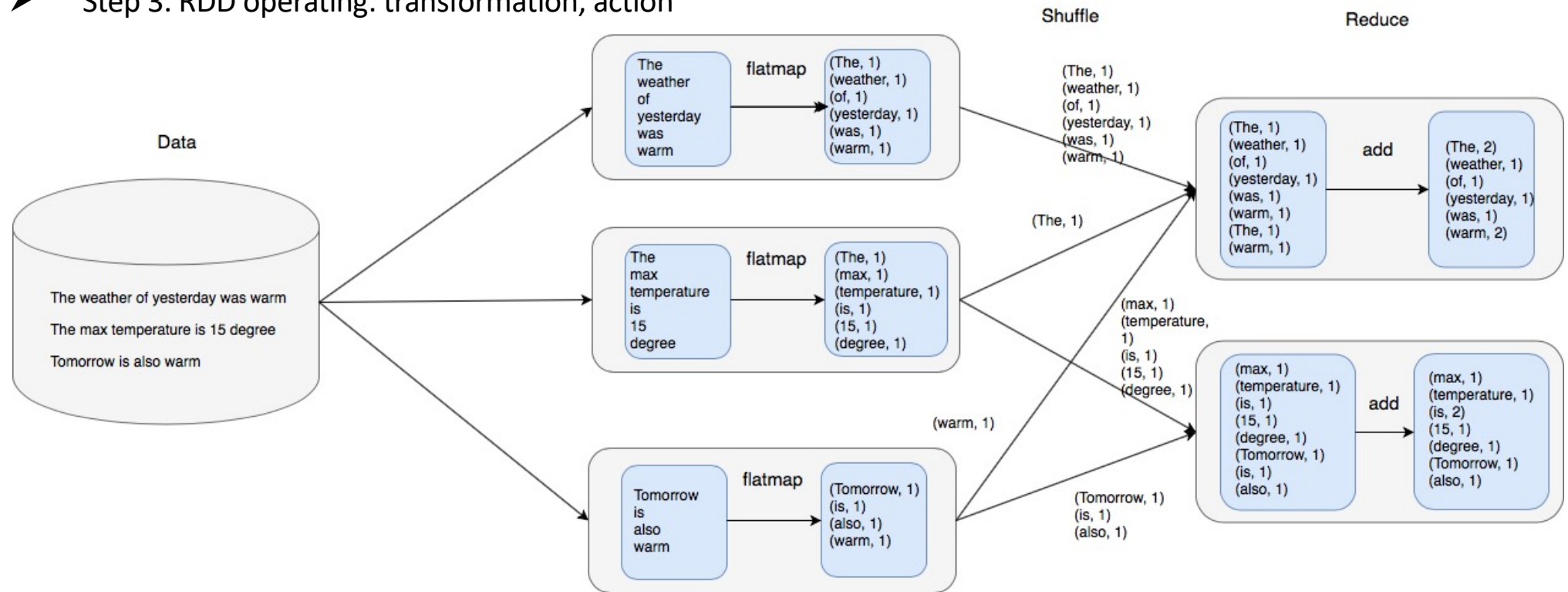
- In terms of map-reduce programming model, how to form key, value pair and what kind of transforms are needed.
- During reduce process, what functions are needed on the values.



How to write PySpark code

- Step 1. To create a SparkContext object which tells Spark how to access a cluster.
- Step 2. To create distributed datasets (RDD)
 - ✓ Use external datasets by local file system or HDFS
- Step 3. RDD operating: transformation, action

- Step 1. To create a SparkContext object which tells Spark how to access a cluster.
- Step 2. To create distributed datasets (RDD)
 - ✓ Use external datasets by local file system or HDFS
- Step 3. RDD operating: transformation, action



```

1 from pyspark import SparkContext
2 sc = SparkContext(appName = "exercise test")
3 news_file = sc.textFile("/user/x_huali/data/news.txt")
4 words = news_file.flatMap(lambda line: line.split(" "))
5 word_count = words.map(lambda word: (word, 1))
6 counts = word_count.reduceByKey(lambda v1, v2: v1+v2)
7 counts.saveAsTextFile("word_count_result")
  
```

Annotations for the code steps:

- Step 1: `sc = SparkContext(appName = "exercise test")`
- Step 2: `news_file = sc.textFile("/user/x_huali/data/news.txt")`
- Step 3: RDD transformation(s): `words = news_file.flatMap(lambda line: line.split(" "))` and `word_count = words.map(lambda word: (word, 1))`
- Step 3: RDD action: `counts.saveAsTextFile("word_count_result")`

Agenda

- Aims of this exercise session
- Review
 - ✓ Map-Reduce: Working with key-value pairs
 - ✓ Lambda functions
- How to design and write PySpark code
- **Lab introduction and exercises**
 - ✓ Conceptual design
 - ✓ Write PySpark code
- *How to work on Sigma

Lab Introduction

- Working with the historical meteorological data from Swedish Meteorological Hydrological Institute (SMHI)
 - ✓ The data includes air temperature and precipitation readings from 812 stations in Sweden.
- Three labs
 - ✓ BDA1 – Spark: 5 assignments
 - In general, you need to do filtering, grouping, aggregating ... over the data.
 - Map-reduce programming model, PySpark,
 - working with key-value pairs
 - ✓ BDA2 – Spark SQL: Redo the 5 assignments in BDA1 with Spark SQL
 - ✓ BDA3 – Machine Learning with Spark:
- Example: Find highest temperature for a certain period
 - ✓ temperature-readings.csv

Headers for *temperature-readings.csv*

Station number	Date	Time	Air temperature (in °C)	Quality ³
----------------	------	------	-------------------------	----------------------

```
102170;2013-11-01;06:00:00;6.8;G
102170;2013-11-01;18:00:00;3.8;G
102170;2014-11-02;06:00:00;5.8;G
102170;2014-11-02;18:00:00;-1.1;G
102170;2015-11-03;06:00:00;-0.2;G
102170;2015-11-03;18:00:00;5.6;G
102170;2015-11-04;06:00:00;6.5;G
.....
```

Find the highest temperature in 2014 and 2015.

Show the year and highest temperature in the result

- Conceptual design
 - ✓ Understand the question and data
 - ✓ How to form key, value pair and what RDD operations are needed
 - ✓ What operations are needed during mapping and reducing?
- Write pyspark code

Headers for *temperature-readings.csv*

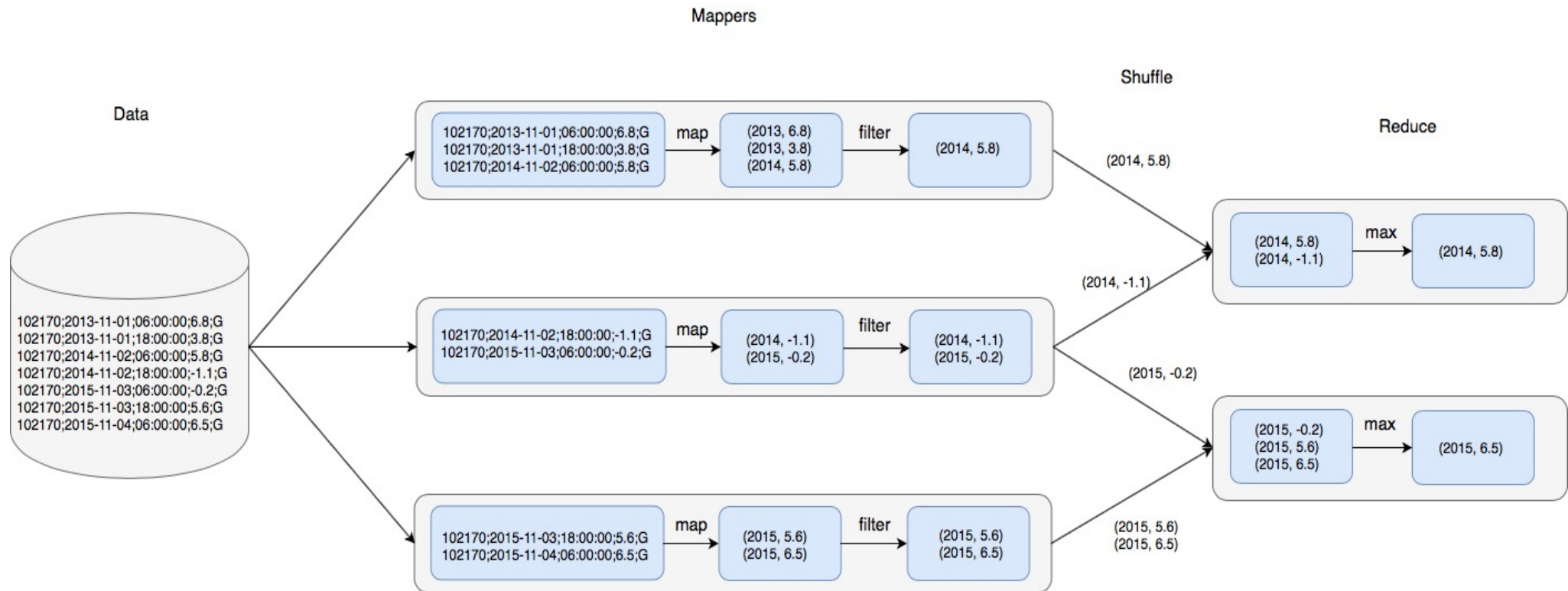
Station number	Date	Time	Air temperature (in °C)	Quality ³
----------------	------	------	-------------------------	----------------------

```
102170;2013-11-01;06:00:00;6.8;G
102170;2013-11-01;18:00:00;3.8;G
102170;2014-11-02;06:00:00;5.8;G
102170;2014-11-02;18:00:00;-1.1;G
102170;2015-11-03;06:00:00;-0.2;G
102170;2015-11-03;18:00:00;5.6;G
102170;2015-11-04;06:00:00;6.5;G
```

.....

Solution – Conceptual design

- Extract year as key and temperature as value
- Filter data (2014 and 2015)
- Reduce by key and then compare each two values to get the higher temperature.



- Step 1. To create a SparkContext object which tells Spark how to access a cluster.
- Step 2. To create distributed datasets (RDD) from HDFS.
- Step 3. RDD operating: transformation, action

Solution

- Question : Find the highest temperature in 2014 and 2015.

```
1 from pyspark import SparkContext
2 def max_temperature(a,b):
3     if a>=b:
4         return a
5     else:
6         return b
7 sc = SparkContext(appName = "exercise test")
8 temperature_file = sc.textFile("/user/x_huali/data/temperature-readings.csv")
9 lines = temperature_file.map(lambda line: line.split(";"))
10 year_temperature = lines.map(lambda x: (x[1][0:4], float(x[3])))
11 year_temperature = year_temperature.filter(lambda x: int(x[0])==2014 or int(x[0])==2015)
12 #max_temperatures = year_temperature.reduceByKey(lambda a,b: a if a>=b else b)
13 #max_temperatures = year_temperature.reduceByKey(max)
14 max_temperatures = year_temperature.reduceByKey(max_temperature)
15 max_temperatures.saveAsTextFile("max_temperature_2014_2015")
```

- line 7: create SparkContext object
- line 8: get the file on hdfs, default home path '/user/USERNAME/'
- line 9: transform the data by splitting each line
- line 10: transform the data by extracting year and temperature as tuple
- line 11: filter data by year
- line 14: reducer, to get the max temperature,
 - line 12, line 13, line 14 show the different ways of passing functions to Spark
- line 15: save result in a directory

For the first assignment in BDA1

- 1) What are the highest temperatures measured each year for the period 1950-2014. Provide the listed sorted in the descending order with respect to the maximum temperature
- Exercise (30 mins)
 - ✓ Conceptual design (how to form key, value pairs and what RDD operations are needed)
 - ✓ Write pyspark code (Pseudocode)

Headers for *temperature-readings.csv*

Station number	Date	Time	Air temperature (in °C)	Quality ³
----------------	------	------	-------------------------	----------------------

102170;2013-11-01;06:00:00;6.8;G
102170;2013-11-01;18:00:00;3.8;G
102170;2014-11-02;06:00:00;5.8;G
102170;2014-11-02;18:00:00;-1.1;G
102170;2015-11-03;06:00:00;-0.2;G
102170;2015-11-03;18:00:00;5.6;G
102170;2015-11-04;06:00:00;6.5;G

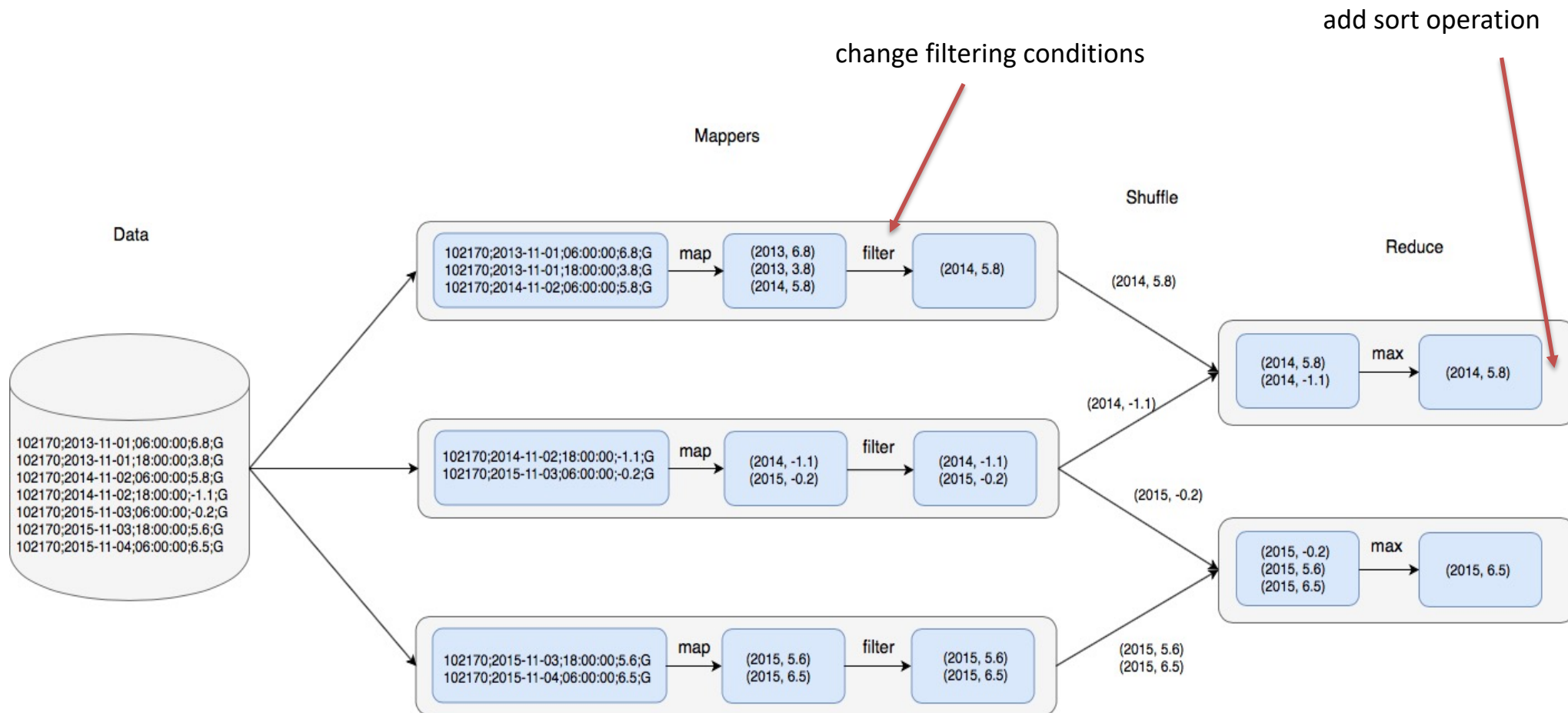
.....

How to write PySpark code

- Step 1: To create a SparkContext object
- Step 2: To create distributed datasets (RDD)
- Step 3: RDD operating

Solution – Conceptual design

- Extract year and temperature
- Filter data (~~2014 and 2015~~) **1950-2014**
- Reduce by key to get maximum
- **Sort**



The previous design for finding max temperatures in 2014 and 2015

Solution

- Pre steps: Distribute your data
- Step 1. To create a SparkContext object which tells Spark how to access a cluster.
- Step 2. To create distributed datasets (RDD) from HDFS.
- Step 3. RDD operating: transformation, action

```
1 from pyspark import SparkContext
2 def max_temperature(a,b):
3     if a>=b:
4         return a
5     else:
6         return b
7 sc = SparkContext(appName = "exercise test")
8 temperature_file = sc.textFile("/user/x_huali/data/temperature-readings.csv")
9 lines = temperature_file.map(lambda line: line.split(";"))
10 year_temperature = lines.map(lambda x: (x[1][0:4], float(x[3])))
11 year_temperature = year_temperature.filter(lambda x: int(x[0])>=1950 and int(x[0])<=2014)
12 #max_temperatures = year_temperature.reduceByKey(lambda a,b: a if a>=b else b)
13 #max_temperatures = year_temperature.reduceByKey(max)
14 max_temperatures = year_temperature.reduceByKey(max_temperature)
15 max_temperaturesSorted = max_temperatures.sortBy(ascending = False, keyfunc=lambda k: k[1])
16 max_temperaturesSorted.saveAsTextFile("max_temperature")
```

- line 7: create SparkContext object
- line 8: get the file on hdfs, default home path '/user/USERNAME/'
- line 9: transform the data by splitting each line
- line 10: transform the data by extracting year and temperature as tuple
- line 11: filter data by a time period
- line 14: reducer, to get the max temperature,
 - line 12, line 13, line 14 show the different ways of passing functions to Spark
- line 15: sort result by temperature
- line 16: save result in a directory

*How to work on Sigma

- Connection

Thinlinc connection (sigma.nsc.liu.se)

`ssh -X username@sigma.nsc.liu.se`

- Submit, monitor, cancel jobs at Sigma

`sbatch, squeue, scancel` commands

- Demo on sigma

[`/software/sse/manual/spark/BDA_demo/`](#)

[`/software/sse/manual/spark/examples/pyspark_on_hdfs/`](#)

- The script for interacting with HDFS and running pyspark code

`run_local.q, run_local_with_historyserver.q, run_yarn.q,
run_yarn_with_historyserver.q`

* How to work on Sigma

```

huali50 — x_huali@sigma:~/BDA_demo — ssh -X x_huali@sigma.nsc.liu.se — 114x50
((base) mac00242:~ huali50$ ssh -X x_huali@sigma.nsc.liu.se
x_huali@sigma.nsc.liu.se's password:
Last failed login: Mon Apr 20 15:19:02 CEST 2020 from 2001:6b0:17:fc09:d154:c210:2f5:697c on ssh:notty
There was 1 failed login attempt since the last successful login.
Last login: Mon Apr 20 15:17:31 2020 from 2001:6b0:17:fc09:d154:c210:2f5:697c
Welcome to NSC and Sigma!

**** Project storage directories available to you:
/proj/tdde31_2020/users/x_huali
/proj/roarc/users/x_huali
/proj/tddd43/users/x_huali

**** Documentation and getting help:
https://www.nsc.liu.se/support/systems/sigma-getting-started/
https://www.nsc.liu.se/support

**** Useful commands
To see your active projects and CPU time usage: projinfo
To see available disk storage and usage: snicquota
To see your last jobs: lastjobs
Login to compute node to check running job: jobsh

To tweak job priorities, extend timelimits and reserve nodes: see
https://www.nsc.liu.se/support/batch-jobs/boost-tools/

(Run "nsc-mute-login" to not show this information)

[x_huali@sigma ~]$ cp -r /software/sse/manual/spark/BDA_demo/ ./
[x_huali@sigma ~]$ cd BDA_demo/
[x_huali@sigma BDA_demo]$ ls
demo.py  input_data  run_local.q  run_local_with_historyserver.q  run_yarn.q  run_yarn_with_historyserver.q
[x_huali@sigma BDA_demo]$ listreservations
Reservations available to user:x_huali / project(s):liu-compute-2020-3
  devel  from NOW to INF  (everyone)

Note: set one of the above as default by running:
  userreservation RESERVATIONNAME
Or without the userreservation alias:
  source /software/tools/bin/userreservation.sh RESERVATIONNAME
[x_huali@sigma BDA_demo]$ sbatch -A liu-compute-2020-3 --reservation devel run_yarn_with_historyserver.q
Submitted batch job 899450
[x_huali@sigma BDA_demo]$ squeue -u x_huali
      JOBID PARTITION    NAME             USER  ST       TIME  NODES NODELIST(REASON)
      899450      sigma  run_yarn      x_huali PD          0:00        2 (Resources)
[x_huali@sigma BDA_demo]$ ls
demo.py  output      run_local_with_historyserver.q  run_yarn_with_historyserver.q  slurm-899469.out
input_data  run_local.q  run_yarn.q                      slurm-899450.out              spark
[x_huali@sigma BDA_demo]$ vi slurm-899469.out
[x_huali@sigma BDA_demo]$ vi slurm-899450.out
[x_huali@sigma BDA_demo]$ cd output/

```

Step 1

Step 2

Step 3

Step 3

Step 4


```
#!/bin/bash
#SBATCH --time=10:00
#SBATCH --nodes=2
#SBATCH --exclusive

module add spark/.2.4.3-hadoop-2.7-nsc1

# Cleanup and start from scratch
rm -rf spark

echo "START AT: $(date)"

hadoop_setup

echo "Prepare output and input directories and files..."
# The following command will make folders on your home folder on HDFS, the input and output folders should be corresponding to the parameter you give to textFile and saveAsTextFile functions in the code
hadoop fs -mkdir -p "BDA" "BDA/input"
hadoop fs -test -d "BDA/output"
if [ "$?" == "0" ]; then
    hadoop fs -rm -r "BDA/output"
fi

hadoop fs -copyFromLocal ./input_data/temperature-readings-small.csv "BDA/input/"
# Remove the comment when you need specific file below
#hadoop fs -copyFromLocal ./input_data/temperature-readings.csv "BDA/input/"
#hadoop fs -copyFromLocal ./input_data/precipitation-readings.csv "BDA/input/"
#hadoop fs -copyFromLocal ./input_data/stations.csv "BDA/input/"
#hadoop fs -copyFromLocal ./input_data/stations-Ostergotland.csv "BDA/input/"

# Run your program
echo "Running Your program..."
exec 5>&1
APPLICATION_ID=$(spark-submit --conf spark.eventlog.enabled=true --deploy-mode cluster --master yarn --num-executors 9 --driver-memory 2g --executor-memory 2g --executor-cores 4 demo.py 2>&1 | tee >(cat - >&5) | awk '!found && /INFO.*Yarn.*Submitted application/ {tmp=gensub(/^.*/Submitted application (.*)$/,"\\1","g");print tmp; found=1}')

echo "===== FINAL OUTPUT ====="
hadoop fs -cat "BDA/output"/*
echo "===== "
echo "Application id: $APPLICATION_ID"
echo "===== "
echo "stderr"
echo "===== "
yarn logs -applicationId "$APPLICATION_ID" | awk -F: '/^LogType/ {if($2=="stderr") {output=1} else {output=0}} output==1 {print}'
echo "===== "
echo "result"
echo "===== "
yarn logs -applicationId "$APPLICATION_ID" | awk -F: '/^LogType/ {if($2=="stdout") {output=1} else {output=0}} output==1 {print}' | grep -v "WARN\\|INFO"

rm -rf output
hadoop fs -copyToLocal 'BDA/output' ./
hadoop_stop

echo "END AT: $(date)"
```

www.liu.se