732A54/TDDE31 Big Data Analytics

Exercise Session

Huanyu Li

LINKÖPING UNIVERSITY

Agenda

- Aims of this exercise session
- Overview of Spark
- RECAP
  - 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

- An overview introduction of Spark (more details given in tomorrow's lecture)
- Give you an overview introduction of the labs
- Help you understand how to design and write code using Spark in python
- Exercises: start to solve the assignments in the labs (BDA1)

Agenda

- Aims of this exercise session
- Overview of Spark
- RECAP
  - ✓ 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

Spark – Overview

- A framework for parallel big data processing
- Distributed collection of data can be processed in parallel
- Basic programming interface:
  - Resilient Distributed Dataset (RDD), a fault-tolerant collection of elements that can be operated on in parallel
  - DataFrame, conceptually equivalent to a table in a relational database or a data frame in R/Python
- Higher-level APIs (PySpark, Spark SQL, MLib, etc.)
- For our lab assignments (BDA1, BDA2, BDA3)
  - First, solve them using Map/Reduce model (BDA1, RDD)
  - Second, solve BDA1's questions using Spark SQL (structured data processing, DataFrame)
  - BDA3: iterative processing (…keep intermediate results in memory across iterations, leading to faster convergence.)

Spark - Data Storage

- How large-scale data is stored and how Spark deals with data?

- One way is to:
  - Organize data in RDDs, divided into partitions,
    - units of parallelism that can be processed independently on different nodes of the cluster

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

- Parallelizing an existing collection given in your program

```
distFile = sc.textFile("data.txt")
```

- Referencing a dataset in an external storage system

✓ We use this for our labs, data in HDFS

Spark – Data Operation

- Manipulate and process a distributed collection of data elements
- Transformations:
  - Operations applied to RDDs to create a new RDD by transforming the data in some way (e.g., data pre-processing and cleaning)
    - map(), filter(), reduceByKey()
  - They are executed only when an action is called, triggering the actual computation of the data.
- Actions:
  - Operations trigger the execution of the Spark computation and return results to the program or write data to an external storage system.
    - count(), reduce(), saveAsTextFile()
- Spark triggers transformations until an action is called

Spark for MapReduce - RDD Operations

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$sample(fraction : \mathrm{Float})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
	$groupByKey()$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
	$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$union()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
	$join()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
	$cogroup()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
	$crossProduct()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$partitionBy(p : \mathrm{Partitioner[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
Actions	$count()$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{Long}$
	$collect()$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{Seq[T]}$
	$reduce(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{T}$
	$lookup(k : \mathrm{K})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{Seq[V]}$ (On hash/range partitioned RDDs)
	$save(path : \mathrm{String})$	:	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/3.5.1/api/python/reference/index.html
  - ✓ Spark 3.5.1 RDD programming guide: https://spark.apache.org/docs/latest/rdd-programming-guide.html

Agenda

- Aims of this exercise session
- Overview of Spark
- RECAP
  - ✓ 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

RECAP - 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 Python objects

    ('a', 3)
    (1, (3, 4))
    ((1,1), 2)
    (1, [1, 2, 3, 4])

- 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]

- Spark 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, describing the transformation

- 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]) *–get the first elements to generate the (key, value) pair*

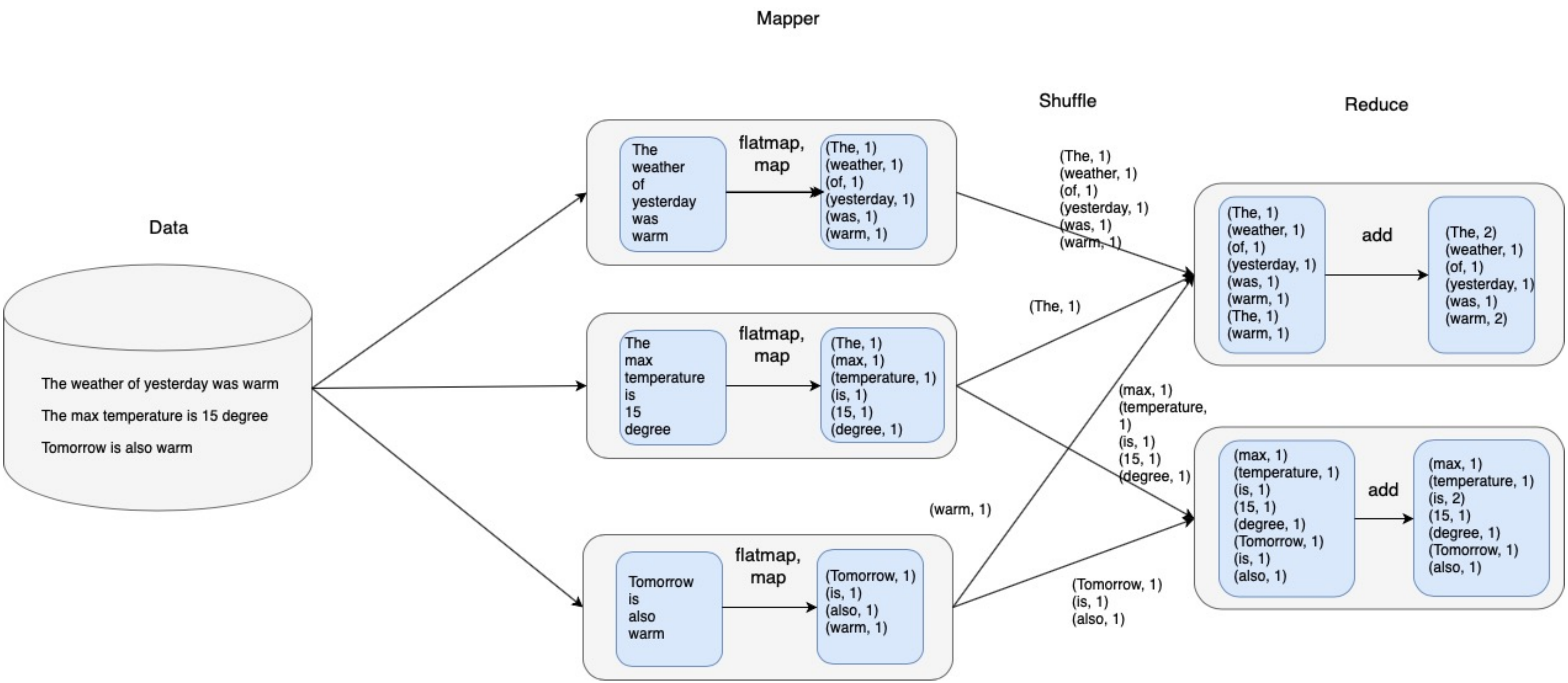
  lambda input_list : max(input_list) *–get the max element in a list*

Agenda

- Aims of this exercise session
- Overview of Spark
- RECAP
  - 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

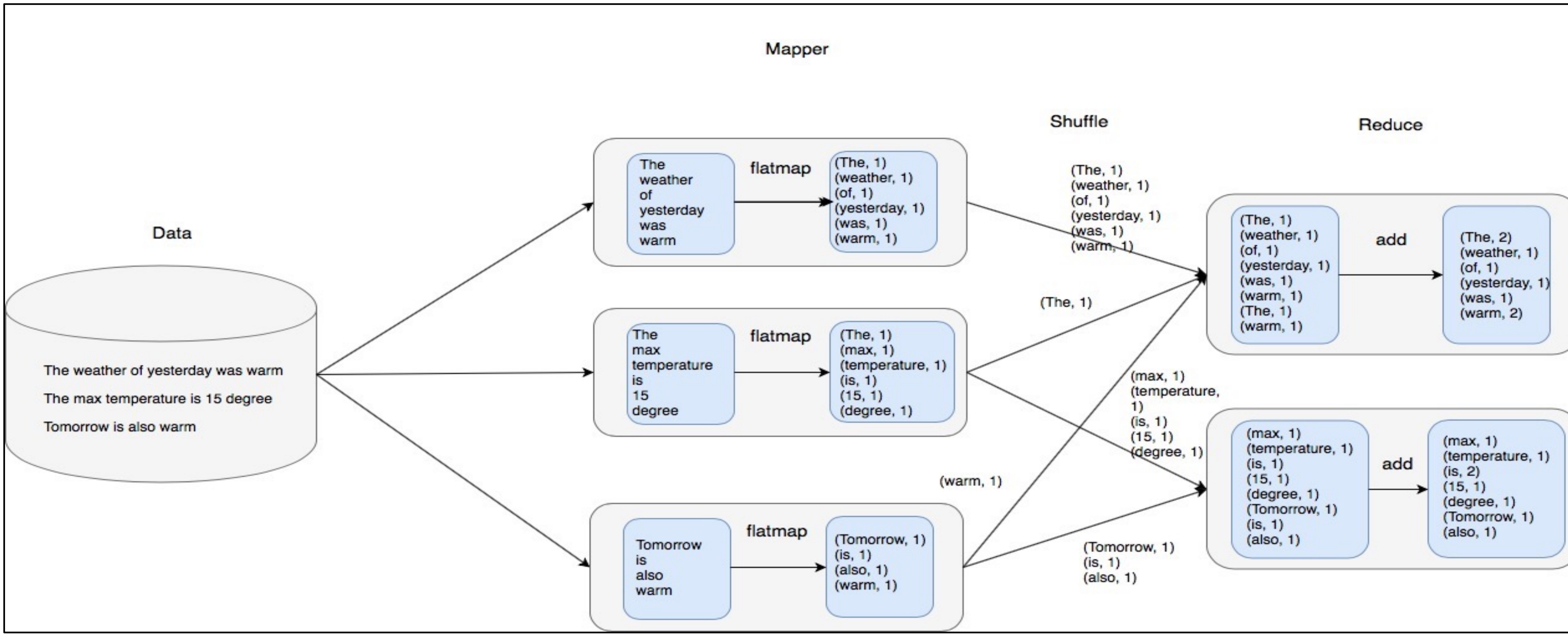
- In terms of map-reduce programming model, how to form key, value pair and what kind of transforms are needed, etc.
- During reduce process, what functions are needed on the values.



How to write PySpark code

- Pre-Step. Upload your data to Hadoop Distributed File System (HDFS) (in cluster mode)
- 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 operation: transformation, action

- Step 1. To create a SparkContext object
- Step 2. To create distributed datasets (RDD)
  - ✓ Use external datasets by local file system or HDFS
- Step 3. RDD operation: transformation, action




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

step 1

step 2

step 3: RDD transformation(s)

step 3: RDD action

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

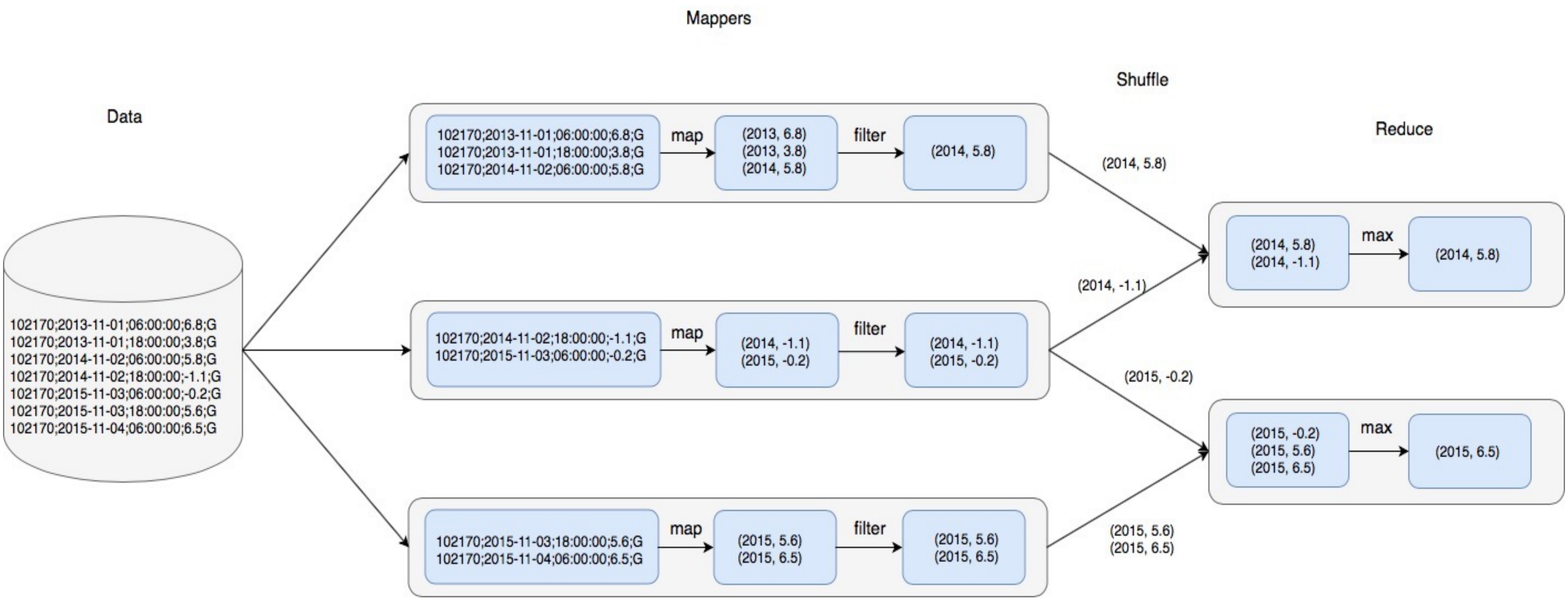
Headers for *temperature-readings.csv*

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
- Step 2. To create distributed datasets (RDD) from HDFS.
- Step 3. RDD operation: 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 (HDFS)

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
  - ✓ Conceptual design (how to form key, value pairs and what RDD operations are needed)
  - ✓ Write PySpark code (Pseudocode)

Headers for *temperature-readings.csv*

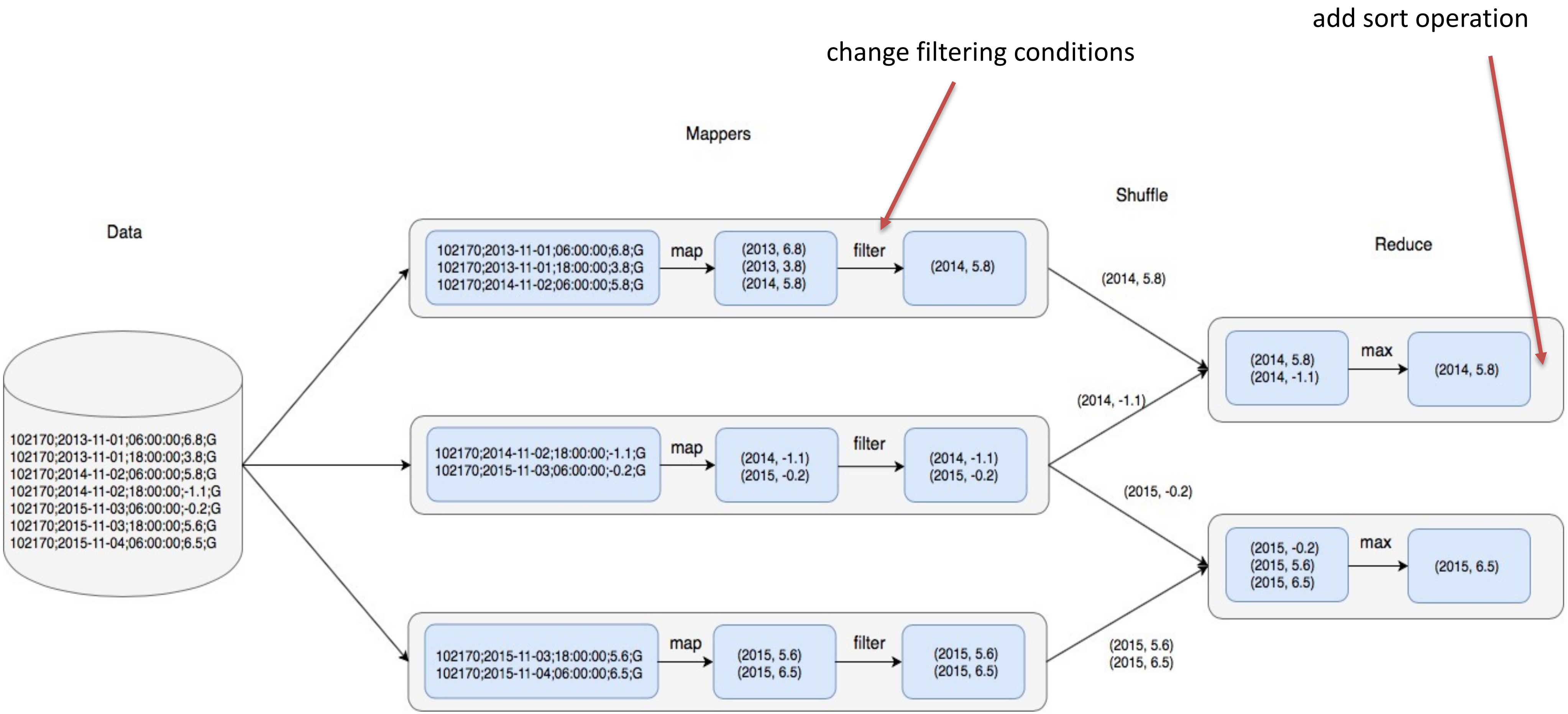
Station number	Date	Time	Air temperature (in °C)	Quality[3]

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 operation

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

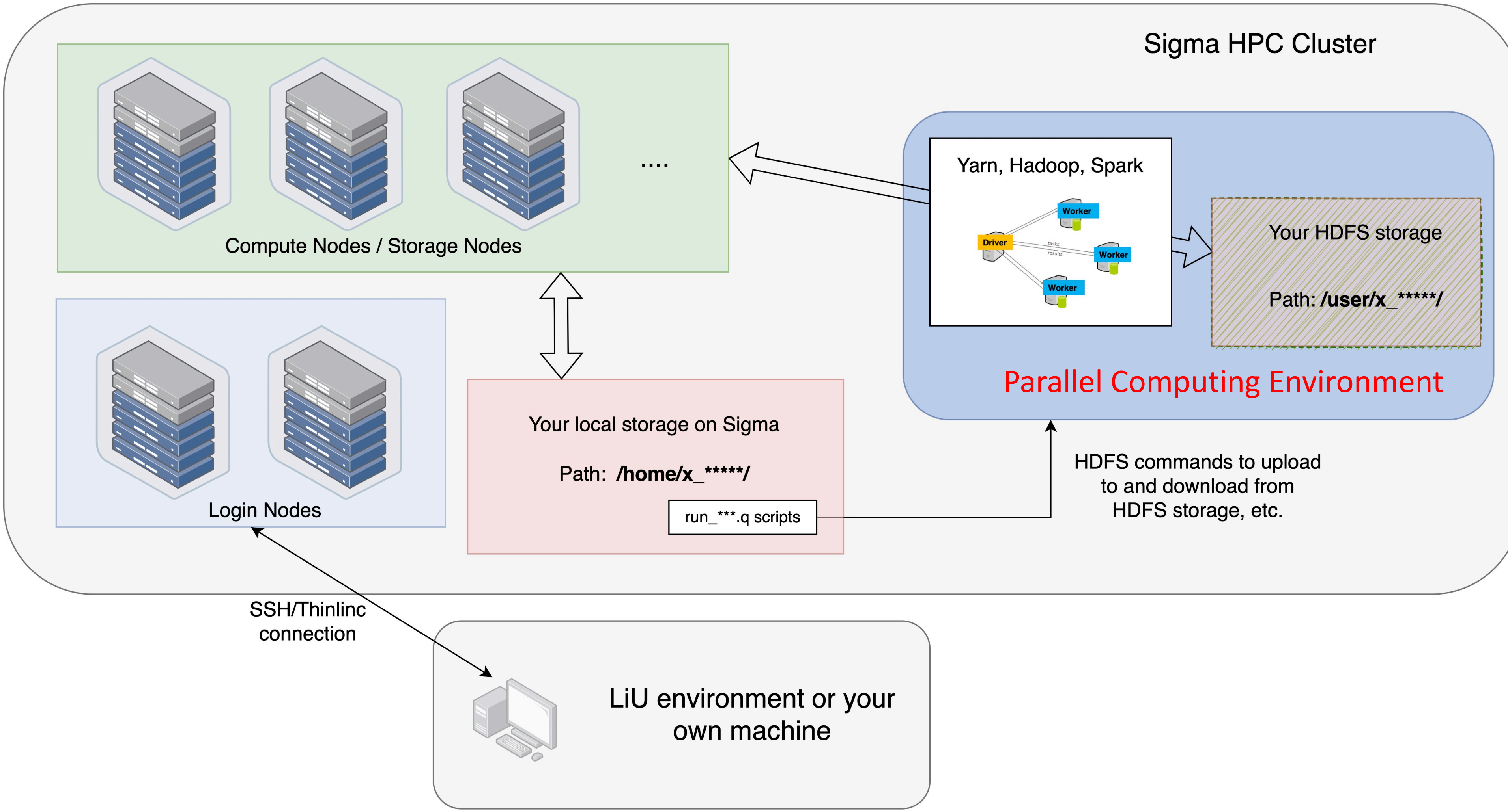
Solution

- Pre steps: Distribute your data
- Step 1. To create a SparkContext object
- 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")
```

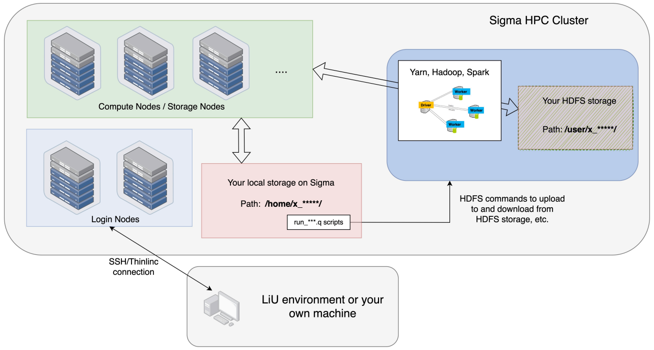
- line 7: create SparkContext object
- line 8: get the file on hdfs, default home path '/user/USERNAME/
- line 9: transform the data by splitting each line
- line 10: transform the data by extracting year and temperature as tuple
- line 11: filter data by a time period
- line 14: reducer, to get the 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 (HDFS)

How to work on Sigma



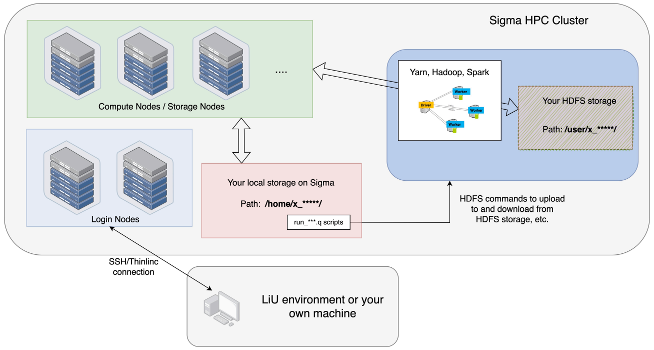
How to work on Sigma

- Connection
  - Option 1: ssh –X username@sigma.nsc.liu.se
    - 1.1: Connect the University environment first. You can use thinlinc to connect 'thinlinc.edu.liu.se', then use 'ssh -X' to connect sigma.
    - 1.2: With your own laptop, you need a X forwarding configuration.
      - An X server software installed on your computer.
        - If you run Linux, this is already taken care of.
        - If you run MacOS, you might need to install and start X11.app (XQuartz: https://www.xquartz.org) which is included in MacOS but not always installed.
        - If you run Windows, you need to find a third-party X server software (e.g Xming https://sourceforge.net/projects/xming/), as this is not normally included in Windows.



How to work on Sigma

- Connection
  - Option 2: Thinlinc connection to Sigma directly (sigma.nsc.liu.se)
    - If you are at a computer in an SU room at the university, to use thinlinc, you need to run following two commands first in a terminal:
      - ***module load course/732A54*** or ***module load course/TDDE31***
      - ***tlclient***
    - If you use your own computers, you just need to download thinlinc and then connect Sigma
    - Notice: During the lab sessions, for each group, please just use at most one thinlinc connection.



How to work on Sigma

- Submit, monitor, cancel jobs at Sigma

  sbatch, squeue, scancel commands
- Demo on sigma

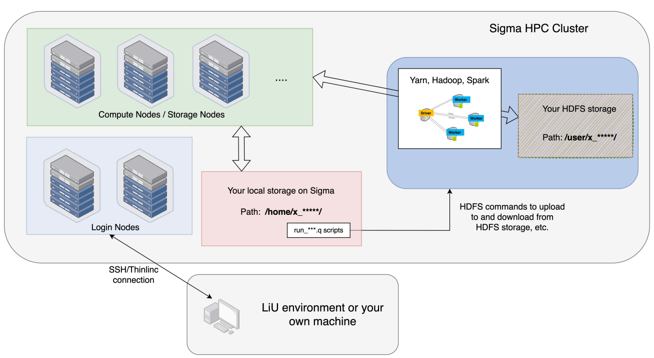
  /software/sse2/tetralith_el9/manual/spark/course-examples/BDA_demo/
- The script for running pyspark code
  - Client (local) or Cluster (yarn) mode
    - Input and output paths/files need to be changed accordingly
    - ! Client mode is just for testing with smaller size data
  - History server, log details for debugging

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

How to work on Sigma (run demo code)

- Step 1: Login Sigma
- Step 2: Copy the code to your home folder on Sigma

```
(py312) huali50@mac01048 ~ % ssh -X x_huali@sigma.nsc.liu.se
(x_huali@sigma.nsc.liu.se) Password:
(x_huali@sigma.nsc.liu.se) Verification code:
Last login: Tue Jan 16 15:32:45 2024 from 2001:6b0:17:fc08:bccc:1abe:3bd4:be6
Welcome to NSC and Sigma!

**** Project storage directories available to you:
/proj/liu-compute-2023-24/users/x_huali
/proj/theophys/users/x_huali
/proj/liu-compute-2024-2/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 ir

[x_huali@sigma ~]$ ls
2_java_wordcount_1.0   Desktop     MapReduc
4_pyspark_wordcount    Documents   Music
732A54-test            Downloads   Pictures
[x_huali@sigma ~]$ cp -r /software/sse2/tetralith_el9/manual/spark/course-examples/BDA_demo/ .
[x_huali@sigma ~]$ ls
2_java_wordcount_1.0   BDA_demo    Downloads
4_pyspark_wordcount    Desktop     MapReduceLab
732A54-test            Documents   Music
[x_huali@sigma ~]$ cd BDA_demo/
[x_huali@sigma BDA_demo]$ ls
demo.py  input_data  run_local.q  run_local
server.q
[x_huali@sigma BDA_demo]$ listreservations
Reservations available to user:x_huali / project(s):liu-compute-2024-2,liu-compute-2023-24,liu-2019
-26
 devel   from 2024-03-21T08:00:00 to 2
 liu-bda-2024-04-09  from 2024-04-09T
 liu-bda-2024-04-11  from 2024-04-11T08:00:00 to 2024-04-11T10:15:00  (project:liu-compute-2024-2)
 liu-bda-2024-04-12  from 2024-04-12T15:00:00 to 2024-04-12T17:15:00  (project:liu-compute-2024-2)
 liu-bda-2024-04-16  from 2024-04-16T13:00:00 to 2024-04-16T17:15:00  (project:liu-compute-2024-2)
 liu-bda-2024-04-18  from 2024-04-18T08:00:00 to 2024-04-18T10:15:00  (project:liu-compute-2024-2)
 liu-bda-2024-04-19
 liu-bda-2024-04-23
 liu-bda-2024-04-25
 liu-bda-2024-04-26
 liu-bda-2024-04-30
 liu-bda-2024-05-02
 liu-bda-2024-05-03
 liu-bda-2024-05-14
 liu-bda-2024-05-16
 liu-bda-2024-05-17  from 2024-05-17T15:00:00 to 2024-05-17T17:15:00  (project:liu-compute-2024-2)
 liu-bda-2024-05-21  from 2024-05-21T13:00:00 to 2024-05-21T17:15:00  (project:liu-compute-2024-2)

Note: set one of the above as default by running:
 usereservation RESERVATIONNAME
Or without the usereservation alias:
 source /software/sse2/generic/manual/ssetools/v1.9.5/bin/hpc_usereservation RESERVATIONNAME
```

How to work on Sigma

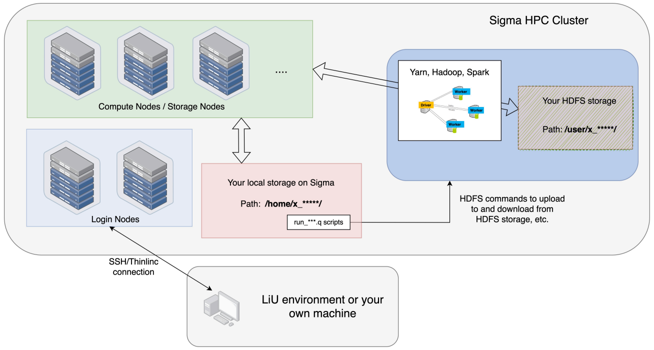
```
[x_huali@sigma BDA_demo]$ sbatch -A liu-compute-2024-2 --reservation devel run_yarn_with_historyserver.q
Submitted batch job 3760375
[x_huali@sigma BDA_demo]$ squeue -u x_huali
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
3760375     sigma run_yarn  x_huali  R       0:06      2 n[1153,1179]
[x_huali@sigma BDA_demo]$ squeue -u x_huali
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
[x_huali@sigma BDA_demo]$ ls
demo.py     output       run_local_with_historyserver.q  run_yarn_with_historyserver.q  spark
input_data  run_local.q  run_yarn.q                      slurm-3760375.out
[x_huali@sigma BDA_demo]$ cd output/
[x_huali@sigma output]$ ls
_SUCCESS  part-00000  part-00001
```

Step 3

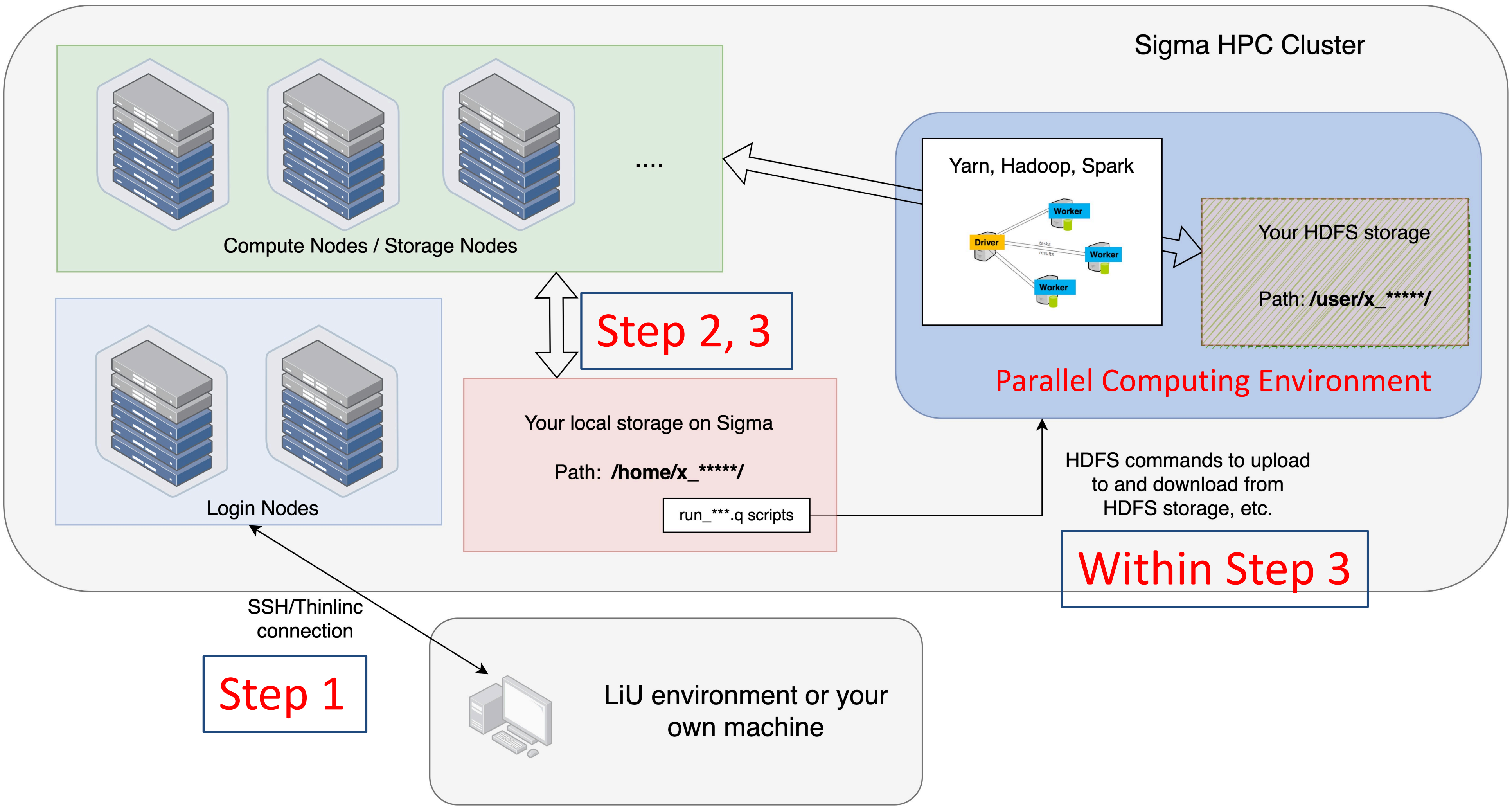
Step 4: check outputs

This indicats that the program runs successfully (without syntax errors).

- Step 3: Submit a job on Sigma
- Step 4: Check the result/output/history server log

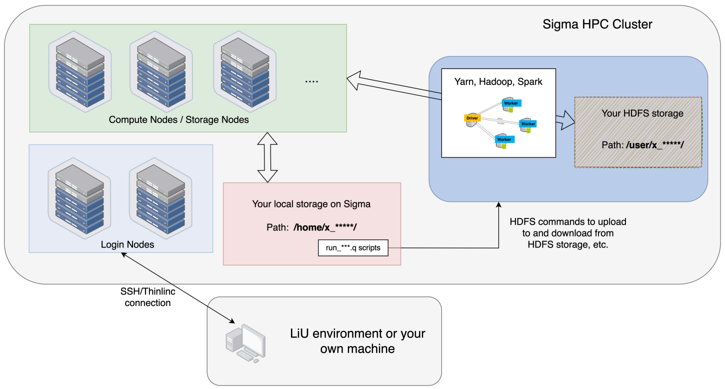


How to work on Sigma



How to work on Sigma

- You will need to make changes on this file, when run your own code.
- run_yarn_with_historyserver.q or one of the other run_*.q scripts



run_yarn_with_historyserver.q

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

echo "START AT: $(date)"

module load spark/3.5.1-hadoop-3.3.6-hpc1-bdist

# Cleanup and start from scratch
rm -rf spark

# Startup hadoop filesystem and yarn
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 fold
ers should be corresponding to the parameter you give to textFile and saveAsTextFile functions i
n 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 specifc 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 "Applicaton id: $APPLICATION_ID"
echo "===================================================================="
echo "=                               stderr                              ="
echo "===================================================================="
yarn logs -applicationId "$APPLICATION_ID" | awk -F: '/^LogType/ {if($2=="stderr") {output=1} el
se {output=0}} output==1 {print}'
echo "===================================================================="
echo "=                               result                              ="
echo "===================================================================="
yarn logs -applicationId "$APPLICATION_ID" | awk -F: '/^LogType/ {if($2=="stdout") {output=1} el
se {output=0}} output==1 {print}' | grep -v "WARN\|INFO"

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

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

Please read these comment lines, you need to change these hadoop commands.

Here you need to replace with your own code.

www.liu.se