Universität Göttingen
Department of Computer Science

Julian Kunkel

Exercise Week 4
HPDA / 2021
240 Minutes Total
Discussion in Week 5: 2021-11-22

---

1. The tasks described in this worksheet are part of the formative assessment. They serve the purpose to prepare you for the examination. We will discuss the solutions during the next **interactive session** after they are handed out – while they fit to the lecture of the week they are handed out, they might be discussed in two weeks time due to the bi-weekly exercise schedule.

2. Make sure to plan your time for the whole sheet carefully. The complete exercise should represent approximately three hours of independent study. The time limit indicates how much time you should spend on each task, and not how much time you may actually need; it is important that you engage with the material and not that you complete all tasks perfectly. Feel free to collaborate and team up.

3. The exercises are designed to challenge you and train you further as guided self-study. The time limit might be too ambitious for you; you may team up with colleagues. It is not an issue as long as you manage to at least partially resolve each task within the time budget. If you (and team) are struggling, reach out for help in Teams! You may also share your thoughts on the channel.

4. We recommend that you create a (private) GIT repository where you store your findings and outcomes while processing the exercises. This portfolio of work could be useful in the future.

Contents

Task 1: MapReduce: Formulating of SQL-queries (60 min)

MapReduce is a powerful programming modell that has been utilized for various applications including to emulate the execution of SQL on data bases. For example, the data warehouse system Apache HIVE [1] is built on top of Hadoop and allows to access data that is stored in files using schema on read.

As part of this task think about how you would implement the following SQL-alike commands into MapReduce jobs. Sketch the map and reduce functions for each query.

1. A simple selection:

```
SELECT fieldA FROM "input.csv" WHERE fieldA == something
```

2. A summation:

```
SELECT groupfield, SUM(fieldA) AS mysum FROM "input.csv" GROUP BY groupfield
```

3. A join:

```
SELECT a.f1, b.f2 FROM "tbl1.csv" AS a JOIN "tbl2.csv" AS b ON a.id = b.id
```

Portfolio (directory: `4/sqlMapReduce`)

`4/sqlMapReduce/selection.txt`	For the simple selection
`4/sqlMapReduce/summation.txt`	For the summation
`4/sqlMapReduce/join.txt`	For the join

Hints

- Maybe thinking about some examples can help.
- The Map phase can read in multiple files.
- Some (more complex) SQL queries may use multiple map reduce jobs that are chained together.

Further Reading

1. `https://en.wikipedia.org/wiki/Apache_Hive`

Task 2: MapReduce: Processing of Text Data (Python) (180 min)

By using MapReduce, we will parallelize the word count problem and apply it to a set of textfiles. Feel free to use/upload any number of textfiles you like to Hadoop. The output should be a list of Article id, list of tuples (words, frequency).

We will build this task in different stages, please start with Step 1 and Step 2 this week, we will complete the task next week if you didn't manage Step 3.

Instead of just outputting the words, we will normalize the words by applying alternatively 1) stemming and 2) lemmatisation (see hints).

2.1 Step 1: Python Surrogate

First, build a Python[1] program that reads files listed on the command line and generate the following output: The output should be formatted as follows[2]:

```
"<textfile>",<wordcount>,"[<stemmed word1>:<count>, ...]", "[lemma of word1:3, lemma of word2: ...]"
```

The last pieces are lists of word and count tuples. `Wordcount` is the total number of words in the textfile.

Also, for each word, calculate the sum of all occurrences across all textfiles and save them as a CSV in such a format:

```
S,StemmedWord,4
L,LemmaWord,3
...
```

[1] If you have no experience with Python, spend some time to learn the basics of Python instead of performing all tasks on this sheet!

[2] You may define the escaping or choose to format the word list as JSON instead.

For example for the sentence: “Translating the Force will be with you. Always.”, the result should be (order of the tokens doesn't quite matter):

```
S,translat,1
S,the,1
S,forc,1
S,will,1
S,be,1
S,with,1
S,you,1
S,alway,1
L,translate,1
L,the,1
L,force,1
L,will,1
L,be,1
L,with,1
L,you,1
L,always,1
```

2.2 Step 2: Emulation using a map and a reduce program

Next, split your program into two programs, a map.py and a reduce.py that read data from stdin and output results to stdout. It shall compute the sum of all occurrences only. You should be able to launch your program from the shell emulating the behavior of MapReduce as follows:

```
cat Input.csv | ./map.py | sort | ./reduce.py
```

Note that the semantics of reduce.py isn't quite the same as of an original map-reduce program as it is expected to be called with multiple keys but that is quite easy to emulate.

2.3 Step 3: Implementation using Hadoop streaming - this task is scheduled for next week

In this setup, create an independent Python program for the map and the reduce phase by considering Hadoop streaming (see below in Hints). Typically, such a program can be tested using the shell pipes. Try to produce a comparable output, albeit the exact output could be difficult to obtain.

2.4 Step 4: Analysis

Measure the runtime of the three programs, your Python version, with the execution using a pipe and using Hadoop. Why is the result as you measure it?

2.5 Hints

2.5.1 Stemming and Lemmatisation

Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form[3]. For example, the word students is based on student and the root of studying is study.

Stemming can be used to normalize words and aggregate all derived words instead of counting them separately.

We can use NTLK[4] in Python for word stemming. There are several implementations of stemmers provided, we will use the SnowballStemmer.

[3] https://en.wikipedia.org/wiki/Stemming
[4] http://www.nltk.org/

```python
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("english")
print(stemmer.stem("studying"))
# prints "studi"; well the world is not perfect
```

Lemmatisation is similar but reduces the word to the word's lemma that is its dictionary form. Check the NTLK documentation for further information how to apply Lemmatisation.

2.5.2 Hadoop streaming

Hadoop streaming is available via hadoop-streaming.jar, it allows for running arbitrary programs as map and reduce functions. External programs are called with keys and values as input, the standard output of the program is used as output of the reducer or mapper.[5]

Running a program with streaming is simple:

```
# Start Hadoop and Yarn
start-dfs.sh
start-yarn.sh
# Prepare Hadoop file system, need to do this once
hadoop fs -mkdir /user/
hadoop fs -mkdir /user/hpda
hadoop fs -mkdir /user/hpda/example
hadoop fs -copyFromLocal /etc/passwd /user/hpda/example/example.csv

# Streaming example, what does this do?
yarn jar /home/hpda/hpda-samples/install/hadoop-3.3.1/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar
    ↪ -mapper $(which echo) -reducer $(which wc -l) -input /user/hpda/example -output /user/hpda/out
# Getting the output
hadoop fs -getmerge /user/hpda/out out
# Print output
cat out
```

Embedding Python is very simple using this functionality:[6]

```
yarn jar $HADOOP/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar \
-Dmapred.reduce.tasks=1 -Dmapred.map.tasks=11 \
-mapper $PWD/my-map.py -reducer $PWD/my-reduce.py -input <inputDir> -output <outputDir>
```

This runs the program `my-map.py` as mapper and `my-reduce.py` as reducer.

A trivial Python example works as follows:

```python
#!/usr/bin/python3
import sys

# This program outputs for every input tuple (line for mapper, key/value pair for reduce)
# the fixed tuple ("key", "value"). It is possible to filter and aggregate tuples (in reduce).
for line in sys.stdin:
   print("key\tvalue\n")
```

Portfolio (directory: `4/mapreduce-wordcount`)

`4/mapreduce-wordcount/Input.csv`	The chosen input data
`4/mapreduce-wordcount/map.py`	The mapper as python program
`4/mapreduce-wordcount/reduce.py`	The reducer as python program

[5] `http://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html`
[6] http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

Hints

- To setup the required Python packages run in the shell: `$ pip3 install --user nltk`

Further Reading

- `https://princetonits.com/hadoop-mapreduce-streaming-using-bash-script/`