University of Reading
Department of Computer Science

Julian Kunkel

Exercise week 2
CS3DP / Autumn 2019
30 Marks Total
Due Date: 2019-10-14l

---

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 tutorial/exercise session on the next Monday they are handed out.

2. Make sure to carefully plan your time for the whole sheet, the complete exercise should represent approximately 5 hours of independent study. Each mark attained should correspond to roughly 10 minutes of your time and includes the time to check the lecture notes. The time limit might be too ambitious for you. It indicates how much time you should spent on each task not how much time you may actually need.
You should get it at least partially resolved with the time budget, if you are struggling then reach out for help at `http://cs-reading.slack.com`. Join here.

3. 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 (10 Marks)

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

Submission (directory: `2/sqlMapReduce`)

`2/sqlMapReduce/fb.txt`	The feedback file for this exercise.
`2/sqlMapReduce/selection.txt`	For the simple selection
`2/sqlMapReduce/summation.txt`	For the summation
`2/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) (20 Marks)

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).
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 article.

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

```
Wort1,4
Wort2,3
...
```

2.2 Step 2: Implementation using streaming

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.

2.3 Step 3: Analysis

Measure the runtime of the three programs, your Python version, with the execution using a pipe and using Hadoop.

2.4 Hints

2.4.1 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.

[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.
[3] `https://en.wikipedia.org/wiki/Stemming`

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

```
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.

2.4.2 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]
Embedding Python is very simple using this functionality:[6]:

```
yarn jar $HADOOP/share/hadoop/tools/lib/hadoop-streaming-3.2.1.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:

```
#!/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")
```

For simpler debugging you can launch your program from the shell emulating the behavior of MapReduce as follows:

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

Submission (directory: `2/mapreduce-wordcount`)

`2/mapreduce-wordcount/fb.txt`	The feedback file for this exercise.
`2/mapreduce-wordcount/Input.csv`	The chosen input data
`2/mapreduce-wordcount/map.py`	The mapper as python program
`2/mapreduce-wordcount/reduce.py`	The reducer as python program

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/`

[4] `http://www.nltk.org/`
[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/