NDBI040: PRACTICAL CLASS 1

MAPREDUCE

Based on NDBI040 practical class materials created by Martin Svoboda; Tutor: Pavel Čontoš; October 14th 2020

(RECOMMENDED) REQUIREMENTS

- Fair programming knowledge, object-oriented programming
- Java (advanced knowledge)
- Maven (it exists)
- Java 8 JDK or newer installed
- NetBeans IDE (or another IDE of yours choice)
- macOS / Linux command line or PuTTy / WinSCP on Windows

MAPREDUCE MODEL

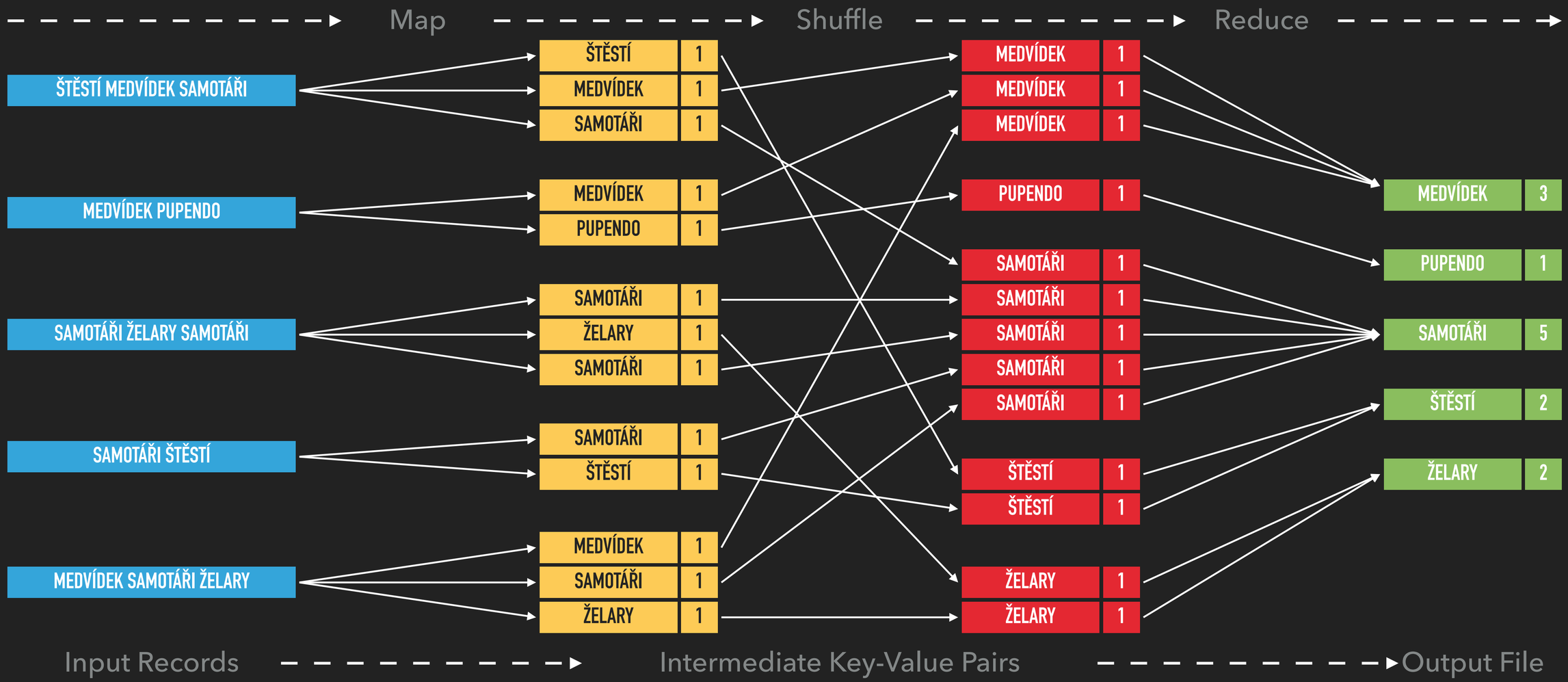
MAP FUNCTION

- Input: an input key-value pair (input record)
- Output: a set of intermediate key-value pairs
  - Usually from a different domain
  - Keys do not have to be unique
- $(key, value) \rightarrow list\ of\ (key, value)$

REDUCE FUNCTION

- Input: an intermediate key + set of (all) values for this key
- Output: a possibly smaller set of values for this key
  - From the same domain
- $(key, list\ of\ values) \rightarrow (key, list\ of\ values)$

EXAMPLE: WORD FREQUENCY



EXAMPLE: WORD FREQUENCY

```
/**
 * Map function
 * @param key Document identifier
 * @param value Document contents
 */

map(String key, String value) {
  foreach(word in value):
    emit(word, 1);
}
```

EXAMPLE: WORD FREQUENCY

```
/**
 * Reduce function
 * @param key Particular word
 * @param values List of count values generated for this word
 */

reduce(String key, Iterator values) {
  int result = 0;
  foreach(value in values):
    result += value;

  emit(key, result);
}
```

SERVER ACCESS

CONNECT TO NOSQL SERVER

- `ssh` on macOS / Linux
- `PuTTy` on Windows

- nosql.ms.mff.cuni.cz:42222
- Login and password send by e-mail
- Change your initial password (if not yet changed) by `passwd`

TRANSFER FILES

- `scp` on macOS / Linux
- `WinSCP` on Windows

APACHE HADOOP

- Open-source framework

- Hadoop Common
- Hadoop Distributed File System (HDFS)
- Hadoop Yet Another Resource Negotiator (YARN)
- Hadoop MapReduce

- Hadoop Ecosystem (ZooKeeper, Avro, Thrift, Sqoop, OOZIE, Flume, Mahout, Pig, Hive, HBase, Accumulo, Storm, Kafka, Solr, Spark, Ambari, ...)

FIRST STEPS

BASIC HADOOP COMMANDS

- `hadoop`
  - Basic help for Hadoop commands
- `hadoop fs`
  - Distributed file system commands
- `hadoop jar`
  - Execution of MapReduce jobs

BROWSE THE HDFS NAMESPACE

- `hadoop fs -ls /`
- `haddop fs -ls /user/`
- `hadoop fs -ls /user/login/`

EXERCISE 1: WORD COUNT JOB (SOLVED)

- Create your working directory
- Copy sample Java source file into your working directory
- Compile WordCount implementation
- Create HDFS working directories
- Copy the sample input data
- Run the MapReduce job
- Explore the job result
- Clean the output directory

EXERCISE 1: WORD COUNT JOB

CREATE YOUR WORKING DIRECTORY

- `cd ~`
- `mkdir -p mapreduce/WordCount`
- `cd mapreduce/WordCount`

COPY SAMPLE JAVA SOURCE FILE INTO YOUR CURRENT FOLDER

- `cp /home/NOSQL/mapreduce/WordCount.java .`
- (notice symbols space and dot at the end of line as a "shortcut" for the current folder)

EXERCISE 1: WORD COUNT JOB

COMPILE OUR WORD COUNT IMPLEMENTATION

- `mkdir classes`

- `javac -classpath /home/NOSQL/mapreduce/hadoop-common-3.1.1.jar:/home/NOSQL/mapreduce/hadoop-mapreduce-client-core-3.1.1.jar -d classes/ WordCount.java`
- (notice no space between jar: and /.)
- (notice a space between classes/ and WordCount.java)

- `jar -cvf WordCount.jar -C classes/ .`
- (notice symbols space and dot at the end of line (we compile jar file to the current folder "."))

EXERCISE 1: WORD COUNT JOB

CREATE YOUR HDFS WORKING DIRECTORIES

- `hadoop fs -mkdir /user/login/WordCount`
- `hadoop fs -mkdir /user/login/WordCount/input1`

COPY THE LOCAL SAMPLE INPUT DATA

- `hadoop fs -copyFromLocal /home/NOSQL/mapreduce/input1/movies.txt /user/login/WordCount/input1`

EXERCISE 1: WORD COUNT JOB

RUN THE PREPARED MAPREDUCE JOB

- `hadoop jar WordCount.jar WordCount /user/login/WordCount/input1 /user/login/WordCount/output1`

- Make sure that the directory `/user/login/WordCount/output1` does not exist before execution of the job
- If the directory already exists and execution of the job fails, remove it (see the next page)

EXERCISE 1: WORD COUNT JOB

RETRIEVE AND EXPLORE THE JOB RESULT

- `hadoop fs -copyToLocal /user/login/WordCount/output1/part-r-00000 result.txt`
- `cat result.txt`

CLEAN THE OUTPUT HDFS DIRECTORY

- `hadoop fs -rm -r /user/login/WordCount/output1`

EXERCISE 2: BIGGER WORD COUNT JOB

- Run our MapReduce job on a bigger input file
- Create your `input2` HDFS directory
- Use a copy of the following input file /home/NOSQL/mapreduce/input2/RomeoAndJuliet.txt
- Run the MapReduce job
- Retrieve and browse the result
- Clean the output HDFS directory

USEFUL COMMANDS

- `mapred job -list all`
  - Lists identifiers of all the MapReduce jobs
- `mapred job -status job-id`
  - Prints status counters for a given MapReduce job (identified by job-id)
- `mapred job -kill job-id`
  - Kills a particular MapReduce job (identified by job-id)

MAPREDUCE PROJECT

- Make sure that Java 8 or newer is installed on your workstation
- Choose your preferred JAVA IDE
- Apache NetBeans IDE, IntelliJ IDEA, Eclipse, ... even Notepad works

- Download `ndbi040-invertedindex` project skeleton from practical class website
- `Open` project `or Import` downloaded (maven) project into your IDE's workspace

- Open pom.xml file and edit the following lines in order to match your Java JDK version (see `java -version` in command line):
  - <maven.compiler.source>14</maven.compiler.source>
  - <maven.compiler.target>14</maven.compiler.target>

- (Clean and) `Build` the project in order to download libraries (so code completion works)

JAVA INTERFACE

MAPPER CLASS

- Implementation of the `map function`
- Template parameters
  - `KEYIN`, `VALUEIN` - types of input key-value pairs
  - `KEYOUT`, `VALUEOUT` - types of intermediate key-value pairs
- Intermediate pairs are emitted via `context.write(k,v)`

```
public class MyMapper extends Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  @Override
  public void map(KEYIN key, VALUEIN value, Context context) throws IOException, InterruptedException {
    /* Implementation */
  }
}
```

JAVA INTERFACE

REDUCER CLASS

- Implementation of the `reduce function`
- Template parameters
  - `KEYIN`, `VALUEIN` - types of intermediate key-value pairs
  - `KEYOUT`, `VALUEOUT` - types of output key-value pairs
- Output pairs are emitted via `context.write(k, v)`

```
public class MyReducer extends Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  @Override
  public void reduce(KEYIN key, Iterable<VALUEIN> values, Context context) throws IOException, InterruptedException {
    /* Implementation */
  }
}
```

EXERCISE 3: INVERTED INDEX

- Implement an inverted index using MapReduce
- Use input files in `/home/NOSQL/mapreduce/input3/`
- Produce a list of `file:occurences` pairs for each word
  - E.g.: `Samotari file1:1 file3:2 file4:1 file5:1`

- `Inside map method:`
  - Use `((FileSplit)context.getInputSplit()).getPath().getName();` to access input file names
- Inside reduce method:
  - Use `Map<String, Integer> map = new HashMap<>();` to process intermediate key-value pairs
  - Use `map.entrySet()` to iterate over map entries

- Compile, deploy and run the job...

REFERENCES

- HDFS: File System Shell commands
  - https://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-common/FileSystemShell.html
- MapReduce: tutorial
  - https://hadoop.apache.org/docs/r3.1.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- MapReduce: shell commands
  - https://hadoop.apache.org/docs/r3.1.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredCommands.html
- MapReduce: JavaDoc
  - https://hadoop.apache.org/docs/r3.1.1/api/