NDBI040: **Big Data Management and NoSQL Databases**

http://www.ksi.mff.cuni.cz/~svoboda/courses/2016-1-NDBI040/

Practical Class 1

MapReduce

Martin Svoboda
svoboda@ksi.mff.cuni.cz

18. 10. 2016

Charles University in Prague, Faculty of Mathematics and Physics
Czech Technical University in Prague, Faculty of Electrical Engineering

MapReduce: Model Overview

Map function

- *Input*: a key-value pair
- *Output*: **a set of intermediate key-value pairs**
  - Usually from a different domain
  - Keys do not have to be unique
- $(k_1, v_1) \rightarrow \texttt{list}(k_2, v_2)$

Reduce function

- *Input*: **an intermediate key + a set of values** for this key
- *Output*: **a possibly smaller set of values** for this key
  - From the same domain
- $(k_2, \texttt{list}(v_2)) \rightarrow (k_2, \texttt{list}(v_2))$

MapReduce: Example

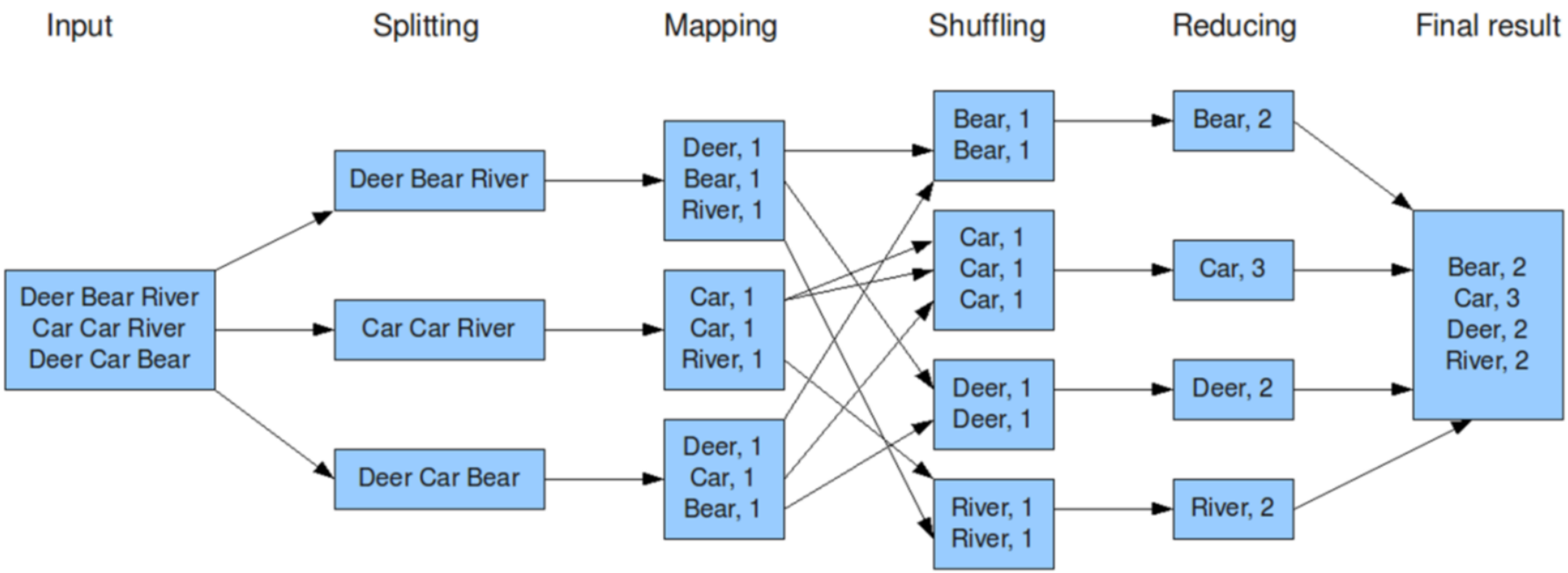
Word Frequency

```
/**
 * Map function
 * @param key   Document name
 * @param value Document contents
 */
map(String key, String value) {
  foreach word w in value: emit(w, 1);
}
```

```
/**
 * Reduce function
 * @param key    Particular word
 * @param values List of count values associated with the word
 */
reduce(String key, Iterator values) {
  int result = 0;
  foreach v in values: result += v;
  emit(key, result);
}
```

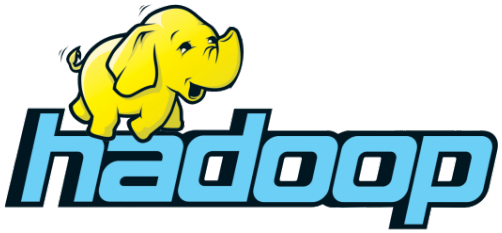
MapReduce: Example

Word Frequency



Apache Hadoop

Open-source software framework



- Hadoop Common
- Hadoop **Distributed File System** (HDFS)
  - Distributed, scalable, and portable file system
- Hadoop Yet Another Resource Negotiator (YARN)
- Hadoop **MapReduce**
  - Implementation of the MapReduce programming model

Tutorial: MapReduce

Server Access

NoSQL server

- SSH and SFTP access
- **nosql.ms.mff.cuni.cz:42222**
- Login and password: sent by e-mail

Useful tools for Linux

- `ssh -p port login@host`
- `sftp -P port login@host`
  - `put local remote`, `get remote local`, …

Useful tools for Windows

- **PuTTY** – http://www.chiark.greenend.org.uk/~sgtatham/putty/
- **WinSCP** – http://winscp.net/

General Information

Change your initial password (if not yet changed)

- `passwd`

Browse important directories

- `/home/login/` – personal directory with your data
- `/home/NOSQL/` – shared directory with course data

Learn how to submit your home assignments

- `submit name`
  - Submits everything located in a sub-directory with name `name`
  - This `name` parameter must also correspond to an assignment name (e.g. mapreduce, …)

First Steps

Get familiar with basic Hadoop commands

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

Browse the HDFS namespace

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

Word Count Example

Create your working directory

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

Make a copy of the sample java source file

- `cp /home/NOSQL/mapreduce/WordCount.java .`

Word Count Example

Compile our sample Word Count program

- `mkdir classes`
- `javac -classpath /usr/local/hadoop/share/hadoop/common/hadoop-common-2.7.3.jar:/usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.7.3.jar -d classes/ WordCount.java`
- `jar -cvf WordCount.jar -C classes/ .`

Word Count Example

Create your HDFS working directories

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

Prepare the sample input data

- `hadoop fs -copyFromLocal /home/NOSQL/mapreduce/input1/file1 /user/login/WordCount/input1`
- `hadoop fs -copyFromLocal /home/NOSQL/mapreduce/input1/file2 /user/login/WordCount/input1`

Word Count Example

Run the prepared MapReduce job

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

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 -rmr /user/login/WordCount/output1/`

Bigger Word Count Example

Run the job on a Shakespeare input file

- Create your `input2` HDFS directory
- Deploy a copy of the input file `/home/NOSQL/mapreduce/input2/shake.txt`
- Run the MapReduce job, retrieve and browse the result
- Clean the output HDFS directory

Useful Commands

Additional MapReduce commands that might be helpful

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

NetBeans Project

Launch NetBeans IDE and create a new project

- Select *Java application* as a project type
- Make local copies of the following Hadoop libraries
  - `/usr/local/hadoop/share/hadoop/common/hadoop-common-2.7.3.jar`
  - `/usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.7.3.jar`
- Add both the required libraries to the project
  - Use *Add JAR/Folder* in the project context menu
- Replace the WordCount source file with the sample one
  - `/home/NOSQL/mapreduce/WordCount.java`

Build the project to create a *jar* distribution

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)`

```
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)`

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

Inverted Index

Implement a new MapReduce job the *inverted index* problem

- Use `((FileSplit)context.getInputSplit()).getPath().getName();` to access the input file names
- Use `Map<String, Integer> map = new HashMap<>();` in the reduce function

Compile, deploy and run the job…

References

HDFS: File System Shell commands

- https://hadoop.apache.org/docs/r2.7.3/hadoop-project-dist/hadoop-common/FileSystemShell.html

MapReduce: tutorial

- https://hadoop.apache.org/docs/r2.7.3/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

MapReduce: shell commands

- https://hadoop.apache.org/docs/r2.7.3/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredCommands.html

MapReduce: JavaDoc

- https://hadoop.apache.org/docs/r2.7.3/api/