



# 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
- $(key, value) \rightarrow \text{list of } (key, value)$

## Reduce function

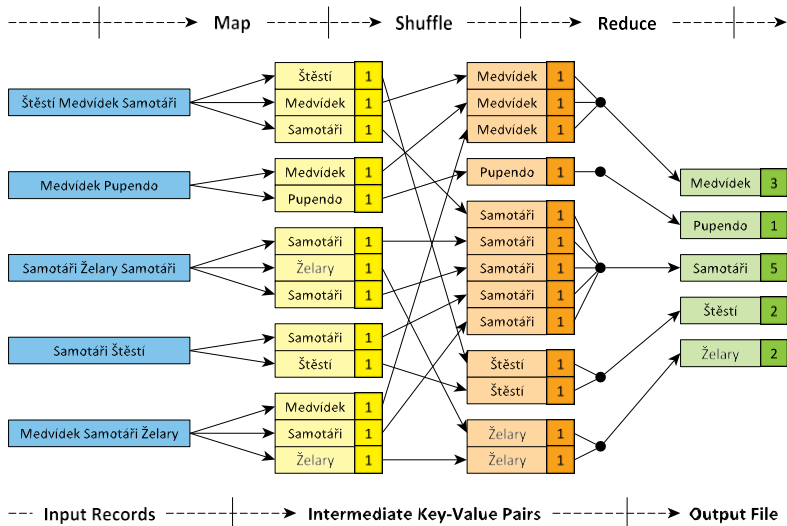
- **Input: an intermediate key + a set of (all) values** for this key
- **Output: a possibly smaller set of values** for this key
  - From the same domain
- $(key, \text{list of } values) \rightarrow (key, \text{list of } values)$

# Example: Word Frequency

```
/**
 * Map function
 * @param key Document identifier
 * @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 generated for this word
 */
reduce(String key, Iterator values) {
    int result = 0;
    foreach v in values: result += v;
    emit(key, result);
}
```

# Example: Word Frequency



# Apache Hadoop

Open-source framework



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

# Server Access

## Connect to our NoSQL server

- ssh and sftp on Linux
- **PuTTY** and **WinSCP** on Windows
- nosql.felk.cvut.cz

Login and password sent by e-mail

## Change your initial password (if not yet changed)

- passwd

# First Steps

## Get familiar with 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 /`
- `hadoop fs -ls /user/`
- `hadoop fs -ls /user/login/`

# Word Count Job

## Create your working directory

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

## Make a copy of the sample java source file

- `cp /home/DS2/mapreduce/WordCount.java .`

# Word Count Job

## Compile our Word Count implementation

- `mkdir classes`
- `javac -classpath \`  
`/home/DS2/mapreduce/hadoop-common-3.1.1.jar:\`  
`/home/DS2/mapreduce/\`  
`hadoop-mapreduce-client-core-3.1.1.jar \`  
`-d classes/ WordCount.java`
- `jar -cvf WordCount.jar -C classes/ .`

# Word Count Job

## 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/DS2/mapreduce/input1/movies.txt \`  
`/user/login/WordCount/input1`

# Word Count Job

## Run the prepared MapReduce job

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

# 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/`

# Bigger Word Count Job

## Run our MapReduce job on a bigger input file

- Create your input2 HDFS directory
- Deploy a copy of the following input file  
`/home/DS2/mapreduce/input2/RomeoAndJuliet.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

# MapReduce Project

**Download the following Hadoop libraries**

- `/home/DS2/mapreduce/  
hadoop-common-3.1.1.jar`
- `/home/DS2/mapreduce/  
hadoop-mapreduce-client-core-3.1.1.jar`

**Choose your preferred Java IDE**

- NetBeans
- IntelliJ IDEA

# NetBeans Project

## Launch NetBeans IDE and create a new project

- Select *Java application* as a project type
  - Let the main class to be created automatically
  - Do not use explicit packages
- Add both the Hadoop libraries into the project
  - Use *Add JAR/Folder* in the project context menu
- Replace the contents of the main class with our pattern
  - `/home/DS2/mapreduce/WordCount.java`
  - Change the name of the class appropriately

## Build the project to create a *jar* distribution

# IntelliJ IDEA Project

## Launch IntelliJ IDEA and create a new project

- Select *Java* as a project type
  - Do not include any additional libraries or frameworks
- Add both the Hadoop libraries into the project
  - Open *Project Structure* in the *File* menu
  - Select *Modules* at the left panel and then *Dependencies* tab
  - Click *+* on the right and select *JARs or directories*
- Add a new Java class into the project
  - Replace its contents with the sample pattern
  - `/home/DS2/mapreduce/WordCount.java`
  - Change the name of the class appropriately

## 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 an *inverted index* using MapReduce

- Use input files in `/home/DS2/mapreduce/input3/`
- Produce a list of *file:occurrences* pairs for each word
  - E.g.: `Samotari file1:1 file3:2 file4:1 file5:1`
- Use `((FileSplit)context.getInputSplit()).getPath().getName();` to access input file names
- 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/>