DataSys 2013 - Tutorial

November 17 - November 22, 2013 - Lisbon, Portugal

The Hadoop Core – Understanding Map Reduce and the Hadoop Distributed File System

Daniel Kimmig[1], Andreas Schmidt[1,2]

(1)
Institute for Applied Sciences
Karlsruhe Institute of Technologie
PO-box 3640
76021 Karlsruhe
Germany

(2)
Department of Informatics and Business Information Systems
University of Applied Sciences Karlsruhe
Moltkestraße 30
76133 Karlsruhe
Germany

Outline

- Motivation
- What is Hadoop ?
- Hadoop Core Components
  - Hadoop Distributed File System (HDFS)
  - MapReduce Framework
- Hadoop Extensions
  - Hive
  - Pig
  - Further extensions

+ three practical exercises

Motivation

- 2010: 1.2 Zetabyte of available data ($1.2 * 10^{21}$) [1]
- 2011: $1.8 * 10^{21}$
- 2020: Expected $40 * 10^{21}$ of data
- actually we generate $2.5 * 10^{18}$ bytes of data daily[2]
- 90% of all available data was generated in the last two years

1. IDC-Study („Extracting value from Chaos“)
2. IBM-Study

Scale Out vs. Scale Up

- Scale-Up (scale vertically):
  - Add resources to computer (CPU, RAM, Disks)
  - Buy a more powerful computer
  - grow in performance is not linear in price
  - Moore’s law can’t compete with grow of data

- Scale-Out (scale horizonally)
  - Add more computers to cluster
  - Software layer can handle the addition or removal of nodes from cluser
  - grow of performance is linear to price
  - Cloud computing - rent nodes only on demand

Karlsruhe Institute of Technology

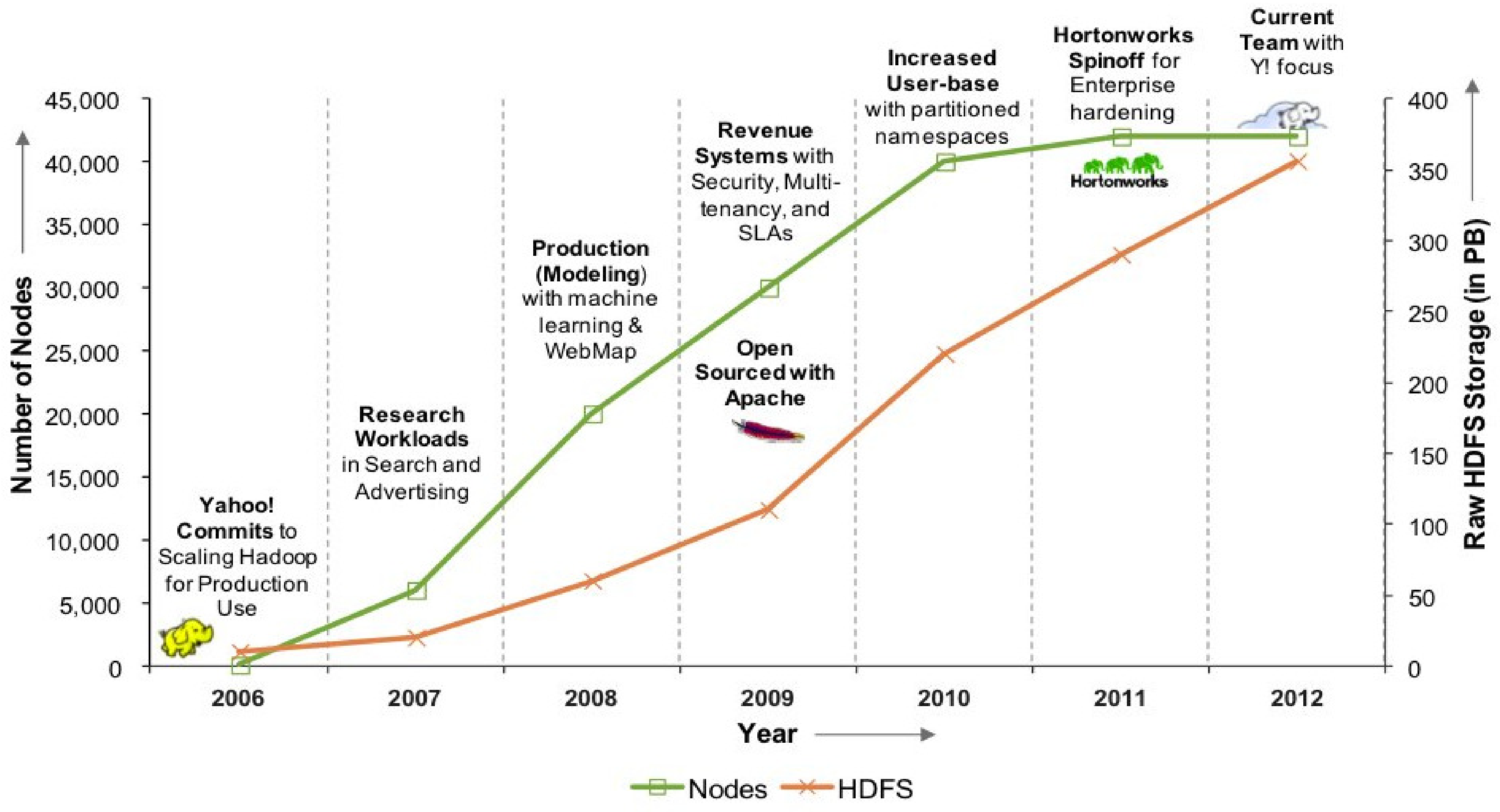
Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

What is Hadoop ?

- Platform for distributed storage and computation of massive amount of data
- Core Components:
  - HDFS (Hadoop Distributed File System)
    - Fault tolerant
    - High Troughput
    - Files of arbitrary size
  - Map Reduce Framework
    - Shared nothing architecture
    - Massive parallelisation of tasks
    - Basic data structure is Key/value pair

- used by: Yahoo, Facebook, Amazon, eBay, Twitter, Rackspace, AOL, LinkedIn, ...
- currently largest cluster: over 40000 machines by Yahoo!

Hadoop at Yahoo!



Source: http://developer.yahoo.com/blogs/ydn/hadoop-yahoo-more-ever-54421.html (Feb 2013)

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

What is Hadoop

- Key Features
  - Accessible - Runs on large cluster off commodity hardware or cloud computing services
  - Robust - can handle most of hardware malfunctions
  - Scalable - Scales linearly by adding more nodes
  - Simple - Easy to use
  - Move code to data philosophy (data locality)
  - Data is broken into chunks and distributed across cluster
  - Ideal for offline processing (batch)
  - Write once - read many

Hadoop is not for ...

- random access (i.e. process transactions)
- work that can not be parallelized
- low latency data access
- many small files
- intensive calculation with only little data

Use The Right Tool For The Right Job

Relational Databases:

When to use?

- Interactive Reporting (<1sec)
- Multistep Transactions
- Lots of Inserts/Updates/Deletes

Hadoop:

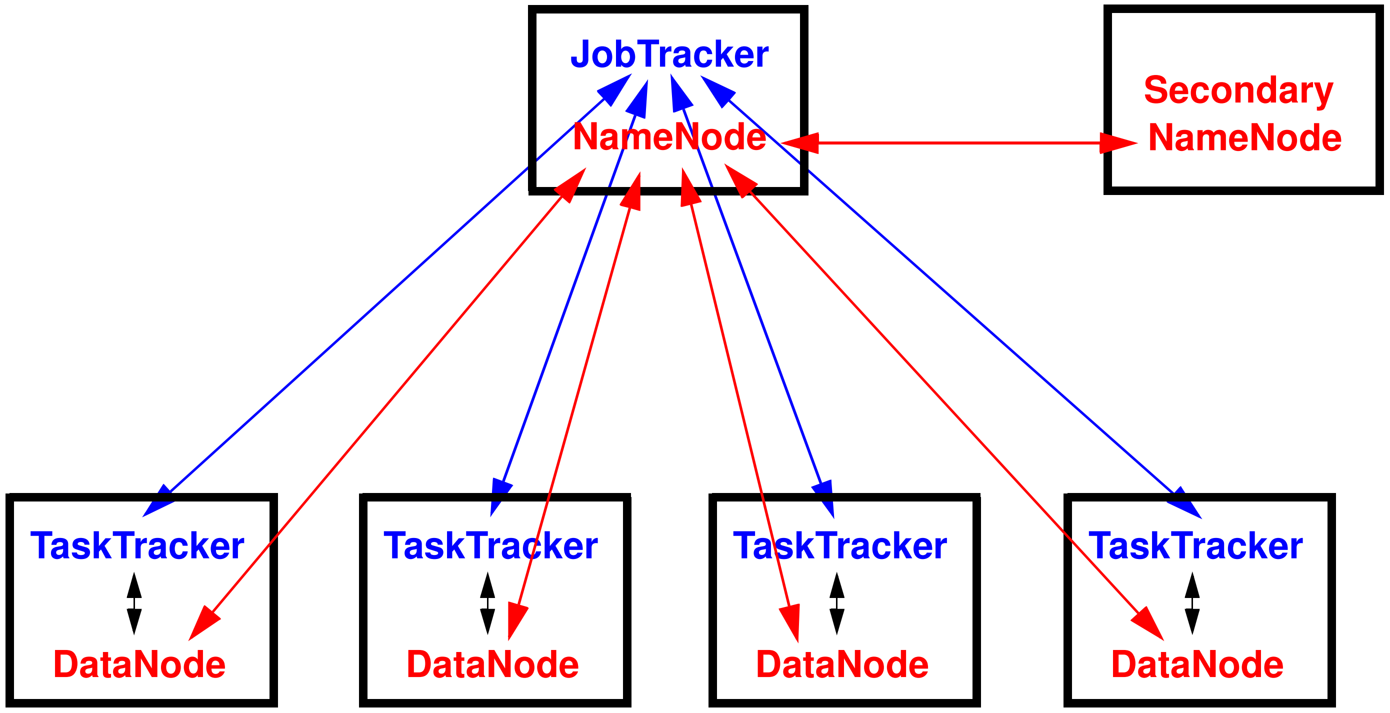
When to use?

- Affordable Storage/Compute
- Structured or Not (Agility)
- Resilient Auto Scalability

Amr Awadallah, Cloudera Inc

Source: http://de.slideshare.net/awadallah/introducing-apache-hadoop-the-modern-data-operating-system-stanford-ee380

Typical Topology (Version 1.x)



Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Hadoop Distributed File System

HDFS

HDFS

- Distributed file system on top of existing OS filesystem
- Runs on a cluster of commodity hardware
- Can automatically handle node failures
- Streaming data access pattern
- Files are fragmented in blocks
- Blocksize typically 64/128 MB
- Blocks are unit of replication (default: 3 replicas)
- Relaxed POSIX standard implementation
    - Append only write access
    - Only single writer (but multiple readers)

Data Node and Name Nodes

Name Node:

- HDFS master node
- Controls distribution of files into blocks
- Controls which nodes store which blocks
- controls overall health of filesys-tem
- Responsible for balancing
- POSIX like authorization model
- Single point of failure (Version 1.x)

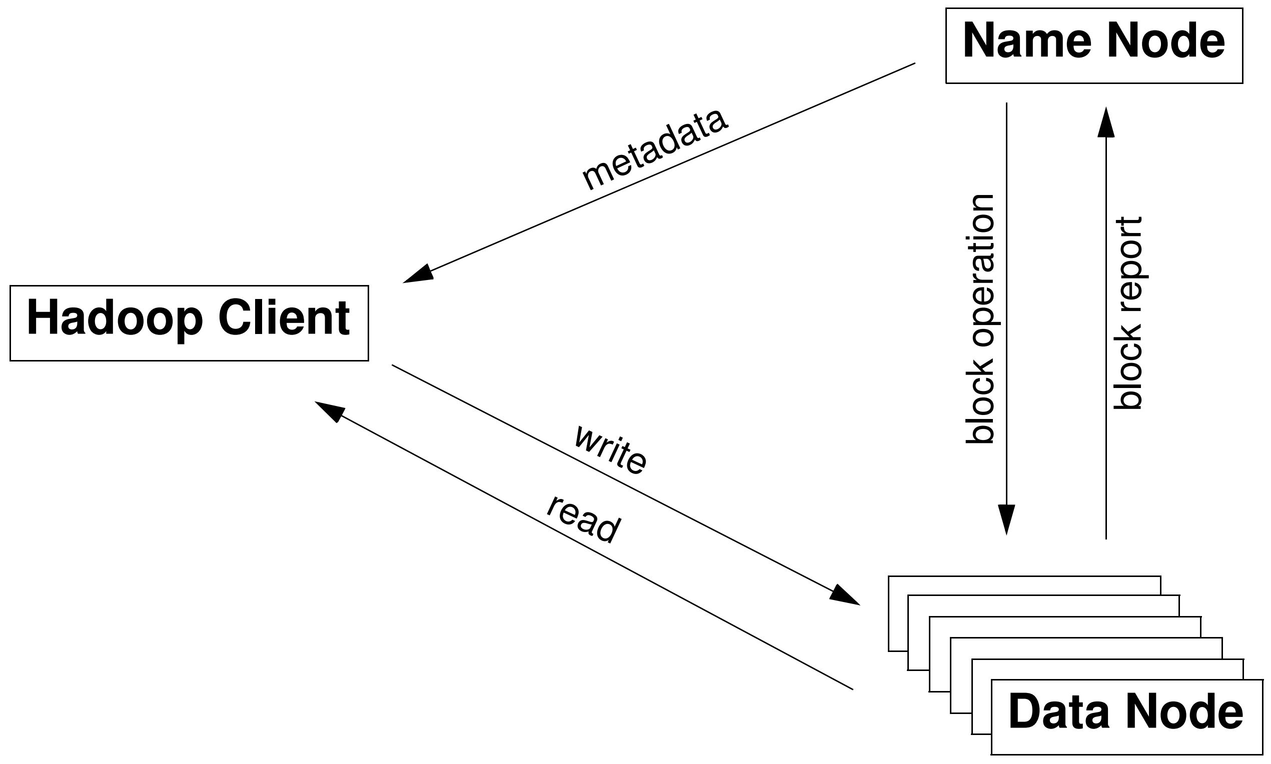
Data Node:

- HDFS slave node
- Reads and writes HDFS blocks
- Direct communication with client
- Report awareness to Name Node

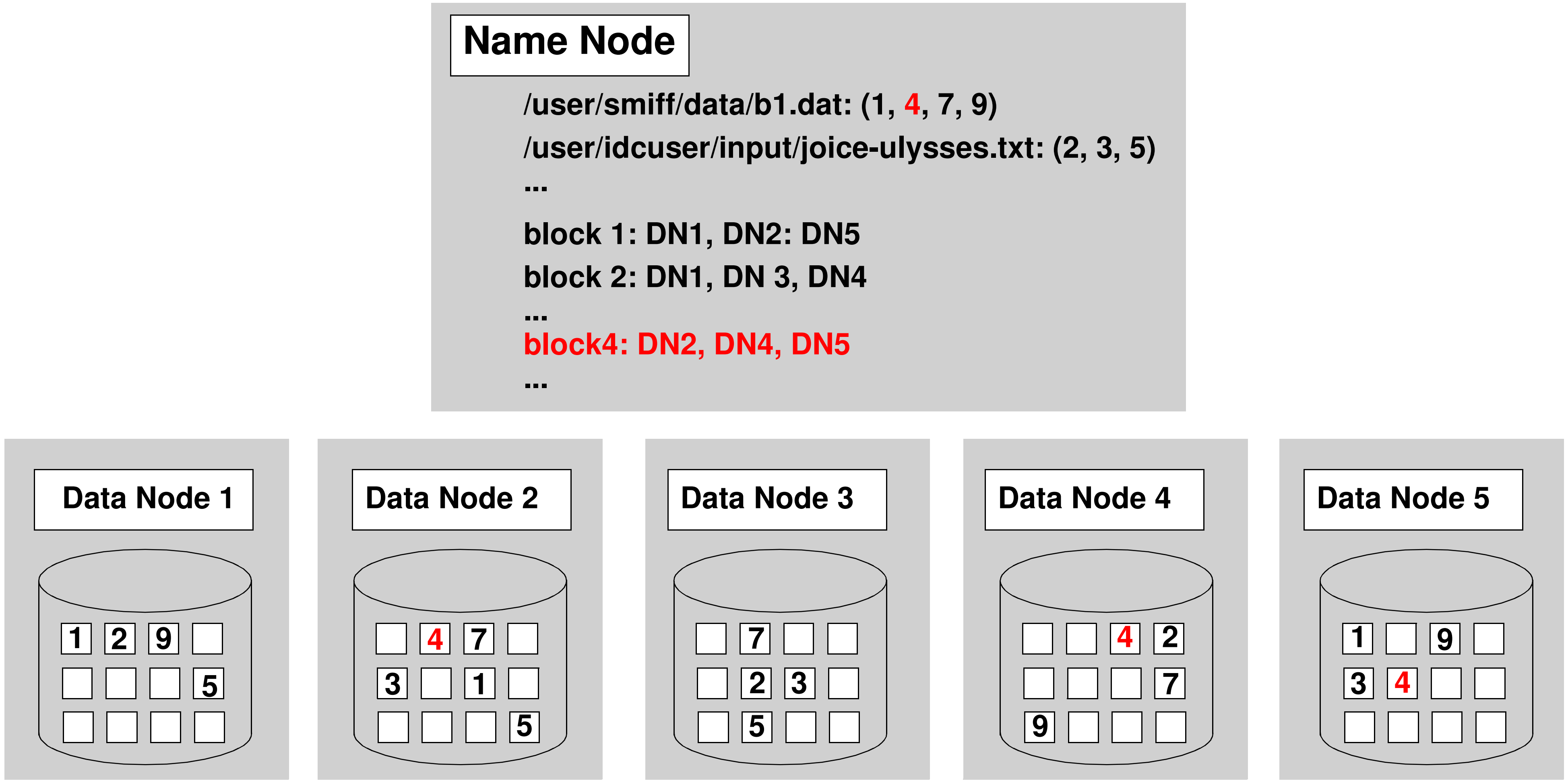
Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft UNIVERSITY OF APPLIED SCIENCES

HDFS



Data Node and Name Nodes



Name Node

/user/smiff/data/b1.dat: (1, 4, 7, 9)
/user/idcuser/input/joice-ulysses.txt: (2, 3, 5)
...

block 1: DN1, DN2: DN5
block 2: DN1, DN 3, DN4
...
block4: DN2, DN4, DN5
...

HDFS Interfaces

- Command Line Interface
- Java API
- Web Interface
- REST Interface (WebHDFS REST API)
- Mounting HDFS: There exist a number of projects like fuse-dfs, fuse-j-hdfs, hdfs-fuse

HDFS Command Line Interface

- Create a directory

```
$ hadoop fs -mkdir /user/idcuser/data
```

- Copy a file from the local filesystem to HDFS

```
$ hadoop fs -copyFromLocal cit-Patents.txt /user/idcuser/data/.
```

- List all files in the HDFS file system

```
$ hadoop fs -ls data/*
```

- Show the end of the specified HDFS file

```
$ hadoop fs -tail /user/idcuser/data/cit-patents-copy.txt
```

- Append multiple files and move them to HDFS (via stdin/pipes)

```
$ cat /data/ita13-tutorial/pg*.txt | hadoop fs -put - data/all_gutenberg.txt
```

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

HDFS commands

- File/Directory Commands:

  `copyFromLocal, copyToLocal, cp, getmerge, ls, lsr (recursive ls), moveFromLocal, moveToLocal, mv, rm, rmr (recursive rm), touchz, mkdir`

- Status/List/Show Commands:

  `stat, tail, cat, test (checks for existence of path, file, zero length files), du, dus`

- Misc Commands:

  `setrep, chgrp, chmod, chown, expunge (empties trashfolder)`

HDFS Java API

- Listing files/directories (globbing)
- Open/close inputstream
- Copy bytes (IOUtils)
- Seeking
- Write/append data to files
- Create/rename/delete files
- Create/remove directory
- Reading Data from HDFS

```
org.apache.hadoop.fs.FileSystem (abstract)
org.apache.hadoop.hdfs.DistributedFileSystem
org.apache.hadoop.fs.LocalFileSystem
org.apache.hadoop.fs.s3.S3FileSystem
```

Example: List content of files in HDFS

```
 public static void main (String [] args) throws Exception{
    try {
        Path hdfsPath = new Path(args[0]);
        Configuration conf = new Configuration();
        FileSystem fs = hdfsPath.getFileSystem(conf);
        FileStatus[] status = fs.listStatus(hdfsPath);
        for (int i=0;i<status.length;i++){
            BufferedReader br = new BufferedReader(
                                    new InputStreamReader(fs.open(status[i].getPath())));
            String line;
            line = br.readLine();
            while (line != null){
                System.out.println(line);
                line = br.readLine();
            }
        }
      } catch(Exception e){
          e.printStackTrace();
      }
    }
}
```

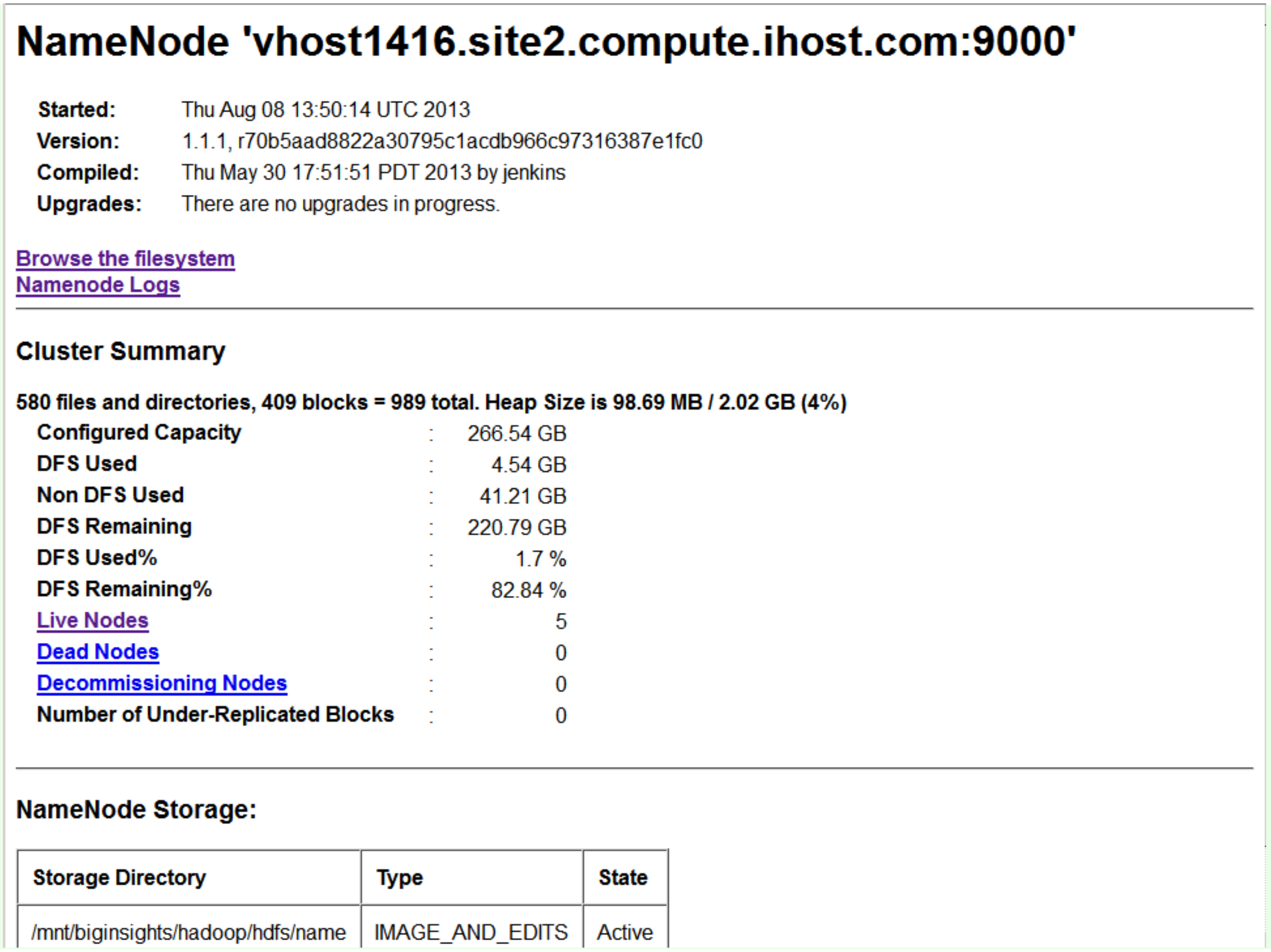
Example: PutMerge

```
// Copies all file from a local directory (args[0]) to a file (args[1]) in hdfs
       Configuration conf = new Configuration();
       Path hdfsPath = new Path(args[1]);
        FileSystem hdfs = hdfsPath.getFileSystem(conf);
        FileSystem local = FileSystem.getLocal(conf);
        try {
            FileStatus[] inputFiles = local.listStatus(new Path(args[0]));
            FSDataOutputStream out = hdfs.create(hdfsPath);
            for (int i=0; i<inputFiles.length; i++) {
                System.out.println("reading " + inputFiles[i].getPath().getName());
                FSDataInputStream in = local.open(inputFiles[i].getPath());
                byte buffer[] = new byte[1024];
                int bytesRead = 0;
                while( (bytesRead = in.read(buffer)) > 0) {
                    out.write(buffer, 0, bytesRead);
                }
                in.close();
            }
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft UNIVERSITY OF APPLIED SCIENCES

HDFS WEB Interface



NameNode 'vhost1416.site2.compute.ihost.com:9000'

Started:	Thu Aug 08 13:50:14 UTC 2013
Version:	1.1.1, r70b5aad8822a30795c1acdb966c97316387e1fc0
Compiled:	Thu May 30 17:51:51 PDT 2013 by jenkins
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

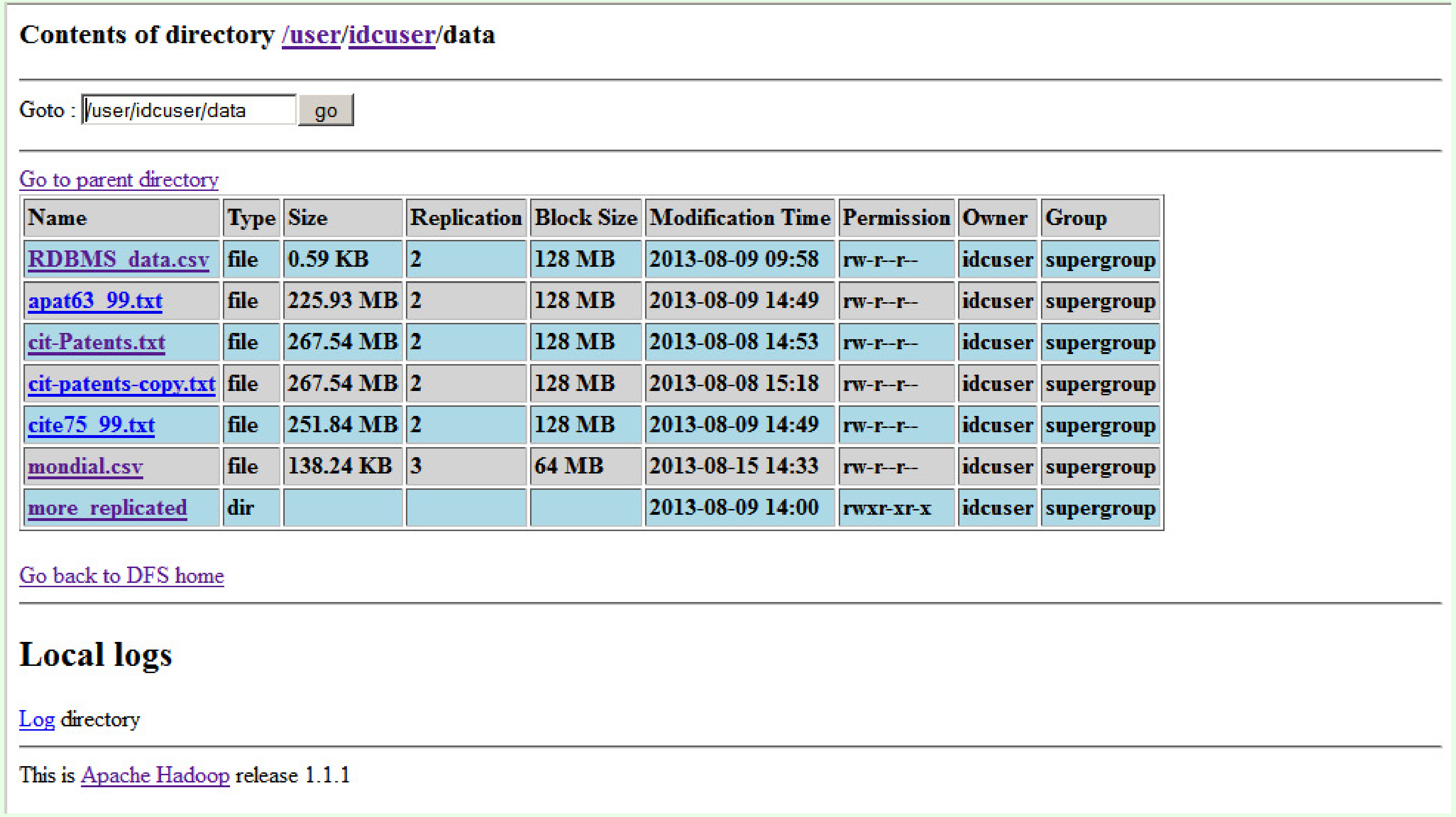
580 files and directories, 409 blocks = 989 total. Heap Size is 98.69 MB / 2.02 GB (4%)

Configured Capacity	:	266.54 GB
DFS Used	:	4.54 GB
Non DFS Used	:	41.21 GB
DFS Remaining	:	220.79 GB
DFS Used%	:	1.7 %
DFS Remaining%	:	82.84 %
Live Nodes	:	5
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

NameNode Storage:

Storage Directory	Type	State
/mnt/biginsights/hadoop/hdfs/name	IMAGE_AND_EDITS	Active

HDFS WEB Interface



Contents of directory /user/idcuser/data

Goto : /user/idcuser/data go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
RDBMS_data.csv	file	0.59 KB	2	128 MB	2013-08-09 09:58	rw-r--r--	idcuser	supergroup
apat63_99.txt	file	225.93 MB	2	128 MB	2013-08-09 14:49	rw-r--r--	idcuser	supergroup
cit-Patents.txt	file	267.54 MB	2	128 MB	2013-08-08 14:53	rw-r--r--	idcuser	supergroup
cit-patents-copy.txt	file	267.54 MB	2	128 MB	2013-08-08 15:18	rw-r--r--	idcuser	supergroup
cite75_99.txt	file	251.84 MB	2	128 MB	2013-08-09 14:49	rw-r--r--	idcuser	supergroup
mondial.csv	file	138.24 KB	3	64 MB	2013-08-15 14:33	rw-r--r--	idcuser	supergroup
more_replicated	dir				2013-08-09 14:00	rwxr-xr-x	idcuser	supergroup

Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.1.1

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Practical Exercise

- Exercise:

Appendix A, Part 1 - HDFS

- Preparation steps:
  - Install putty.exe
    (can be downloaded from here: http://www.smiffy.de/immm-2013/)
  - WinSCP.exe
  - Ask me for account information ;-)

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

MapReduce

Motivation Example

Task to perform: Take the book *Ulysses* from James Joice and count the number of appearances for each word.

1.) Start with the first word on the first side.

2.) Write the word on a sheet of paper and make a bar behind the word[1]

3.) for every next word ...

    Look if the word is already on the list of your paper ...

    if yes, make another bar behind the word

    if not, write the word, together with a single bar on the paper

4.) sum up the bars behind each word - finished !!

1. ... or use sorted file cards

Motivation Example

What would change if you have 75 friends who can help you ?

Distributed solution:

1.) Give every friend ten pages of the book

2.) Every friend now performs the algorithm on the previous slide for only her page

3.) You collect all the sheets of paper (or file cards) and aggregate the results.

Motivation Example

What would change if you have 75 friends who can help you ?

Distributed solution:

1.) Give every friend ten pages of the book

2.) Every friend now performs the algorithm on the previous slide for only her page

3.) You collect all the sheet of papers (or file cards) and aggregate the results.

Instead of collecting all the 75 peace of papers and aggregate te final result, your friends can help you again !!!

Motivation Example

- Appoint 10 of your friends who should receive the pieces of paper from about 7 or 8 persons and aggregate these results.
- Then, you collect the ten preliminary aggregated results and build the final result

Karlsruhe Institute of Technology

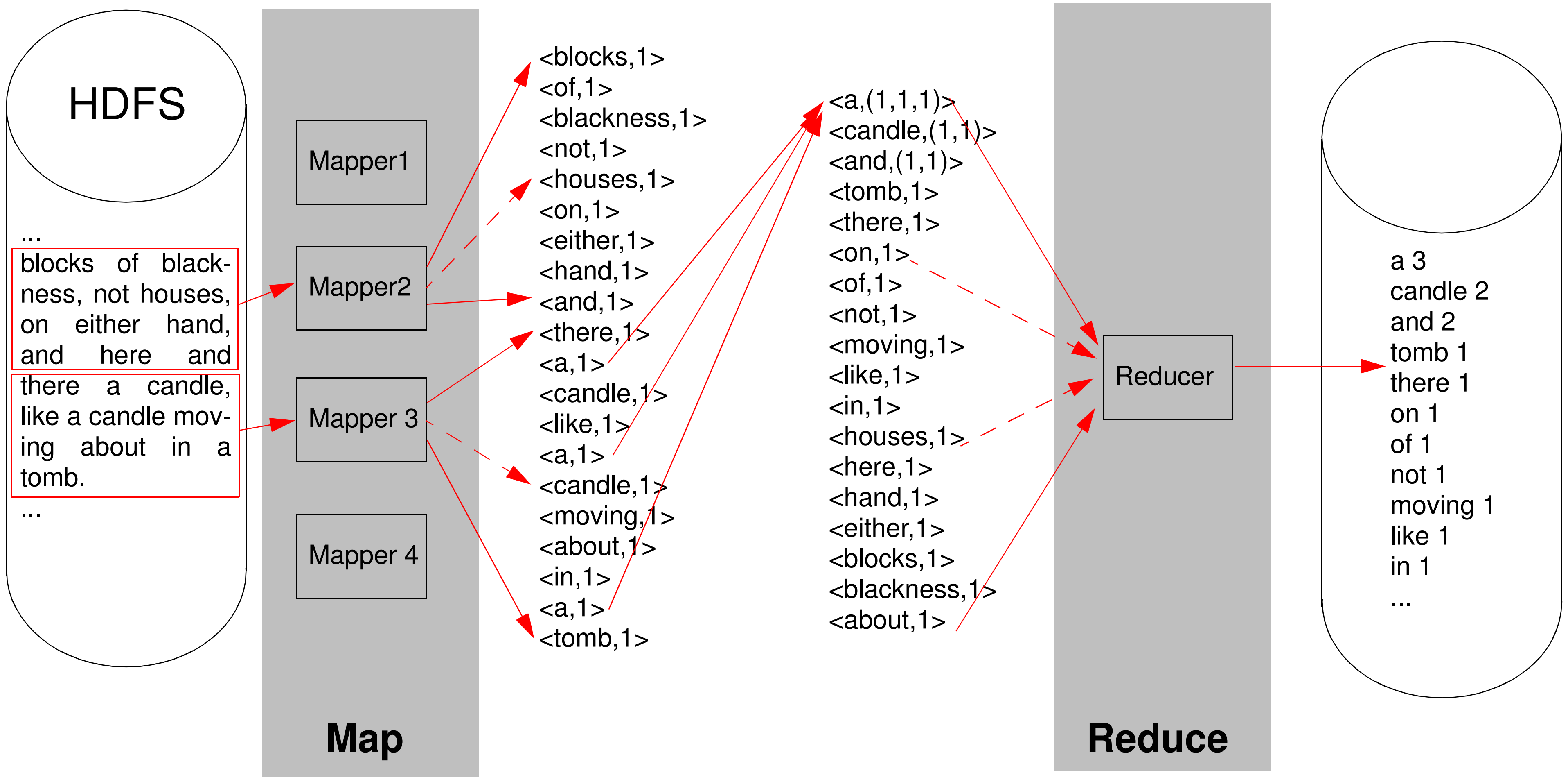
Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

MapReduce

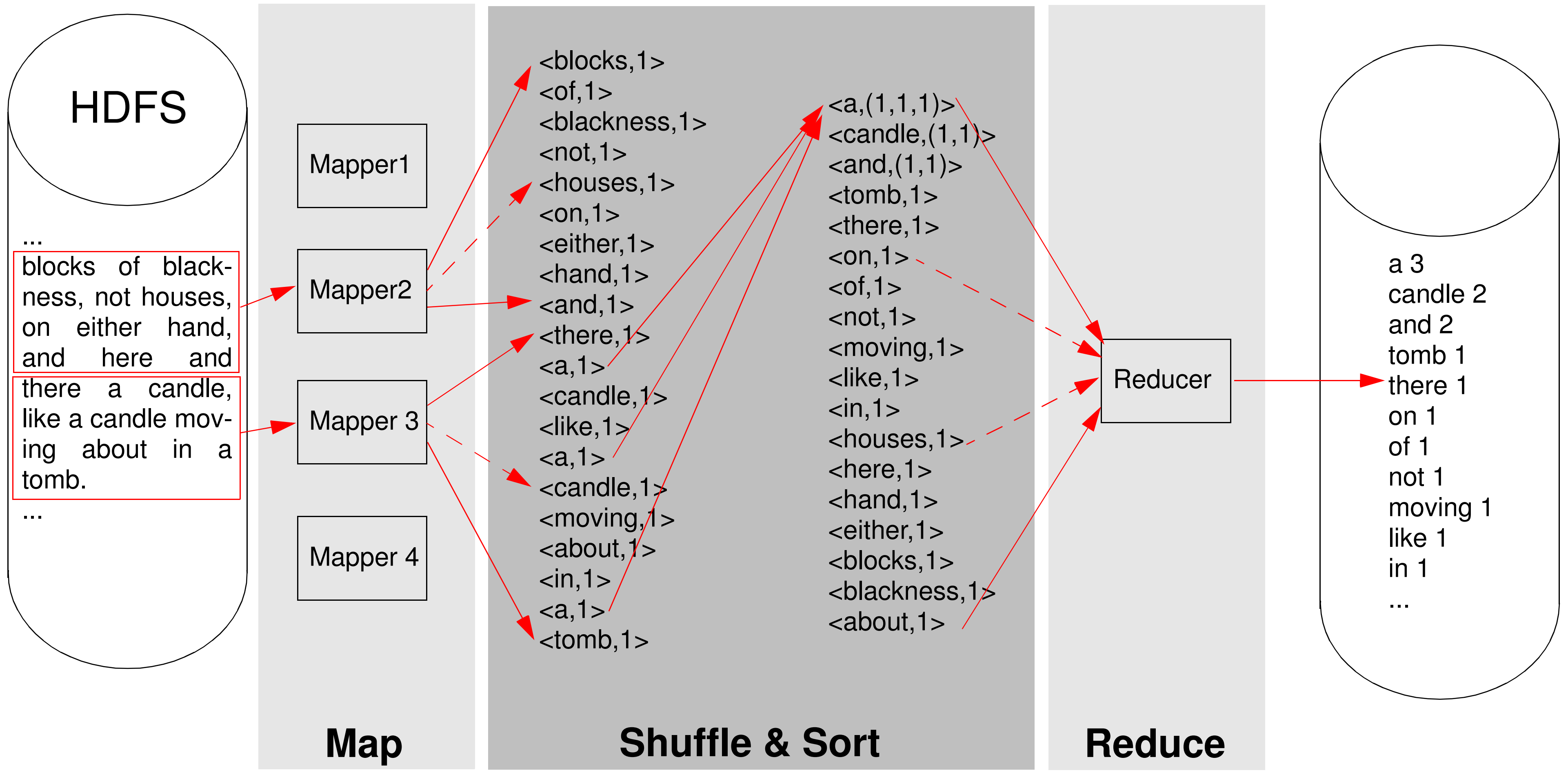
How can this manual task be performed by Hadoop ?

- Every friend is a task node
- The book *Ulysses* is a HDFS file
- Every page of the book is a block on a data node
- The counting of words is a mapper job
- The aggregation(s) is/are reducer job(s)

MapReduce - Programming Model



MapReduce - Programming Model



Map-Reduce Example (Java)

```
public class WordCount {

    public static class Map extends MapReduceBase
                 implements Mapper<LongWritable, Text, Text, IntWritable> {

      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();

      public void map(LongWritable key, Text value,
                      OutputCollector<Text, IntWritable> output,
                      Reporter reporter) throws IOException {

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
               word.set(tokenizer.nextToken());
               output.collect(word, one);
            }
      }
   }
```

$(\mathtt{key}_1, \mathtt{value}_1)$

$\mathtt{list}(\mathtt{key}_2, \mathtt{value}_2)$

$(\mathrm{key}_2,\ \mathrm{list}(\mathrm{value}_2))$

```
    public static class Reduce extends MapReduceBase
                        implements Reducer<Text, IntWritable, Text, IntWritable> {

          public void reduce(Text key, Iterator<IntWritable> values,
                             OutputCollector<Text, IntWritable> output,
                             Reporter reporter) throws IOException {

              int sum = 0;
              while (values.hasNext()) {
                  sum += values.next().get();
              }
              output.collect(key, new IntWritable(sum));
          }
    }

} // class WordCount
```

$\mathrm{list}(\mathrm{key}_3,\ \mathrm{value}_3)$

```
public static void main(String[] args) throws Exception {

            JobConf conf = new JobConf(WordCount.class);
            conf.setJobName("wordcount");


            conf.setInputFormat(TextInputFormat.class);
            conf.setOutputFormat(TextOutputFormat.class);

            conf.setOutputKeyClass(Text.class);
            conf.setOutputValueClass(IntWritable.class);

            conf.setMapperClass(Map.class);
            conf.setReducerClass(Reduce.class);

            FileInputFormat.setInputPaths(conf, new Path(args[0]));
            FileOutputFormat.setOutputPath(conf, new Path(args[1]));

            JobClient.runJob(conf);
          }
       }
```

```
runWordCount: wordcount.jar
        hadoop fs -copyFromLocal ulysses.txt input/.
        hadoop fs -rmr output
        hadoop jar wordcount.jar org.myorg.WordCount input/ulysses.txt output
        hadoop fs -cat output/*000



HADOOP_CLASSPATH=${HADOOP_HOME}/hadoop-core.jar

wordcount.jar: WordCount.java
        mkdir -p wordcount_classes
        javac -classpath ${HADOOP_CLASSPATH} -d wordcount_classes WordCount.java
        jar -cvf wordcount.jar -C wordcount_classes/ .
```

WordCount Example - Output

```
hadoop jar wordcount.jar org.myorg.WordCount input/MobyDick.txt output

13/09/02 15:06:41 INFO mapred.FileInputFormat: Total input paths to process : 1
13/09/02 15:06:41 INFO mapred.JobClient: Running job: job_201308081351_0206
13/09/02 15:06:42 INFO mapred.JobClient:  map 0% reduce 0%
13/09/02 15:06:47 INFO mapred.JobClient:  map 50% reduce 0%
13/09/02 15:06:53 INFO mapred.JobClient:  map 100% reduce 0%
13/09/02 15:07:02 INFO mapred.JobClient:  map 100% reduce 33%
13/09/02 15:07:04 INFO mapred.JobClient:  map 100% reduce 100%
13/09/02 15:07:05 INFO mapred.JobClient: Job complete: job_201308081351_0206
...
13/09/02 15:24:38 INFO mapred.JobClient:     Reduce input groups=33782
13/09/02 15:24:38 INFO mapred.JobClient:     Combine output records=0
13/09/02 15:24:38 INFO mapred.JobClient:     Map output records=215133
13/09/02 15:24:38 INFO mapred.JobClient:     Map input records=22108
13/09/02 15:24:38 INFO mapred.JobClient:     Reduce shuffle bytes=2521615
13/09/02 15:24:38 INFO mapred.JobClient:     Combine input records=0
13/09/02 15:24:38 INFO mapred.JobClient:     Spilled Records=430266
13/09/02 15:24:38 INFO mapred.JobClient:     Reduce input records=215133
13/09/02 15:24:38 INFO mapred.JobClient:     Reduce output records=33782
13/09/02 15:24:38 INFO mapred.JobClient:     Map output materialized bytes=2521615
```

WordCount Improvement

- Each mapper outputs a lot of (<word>,1) tuples.
- Each result tuple has to be transported to the reducer (over the network)
- Add for each mapper a local reducer, which counts for each word the number of tuples:

$$\left.\begin{array}{l}\texttt{<the,1>}\\\texttt{<the,1>}\\\texttt{<the,1>}\end{array}\right\}\ \texttt{= <the,3>}$$

- How to do ?
  - Implement an additional Combiner class (with a reduce method) using the Reducer interface
  - Use the Reducer class also as combiner, i.e.:

```
conf.setMapperClass(Map.class);
conf.setCombinerClass(Reduce.class);
conf.setReducerClass(Reduce.class);
```

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

WordCountWithCombiner Example - Output

without combiner

```
hadoop jar wordcount.jar org.myorg.WordCountWihCombiner input/MobyDick.txt output

13/09/02 15:23:44 INFO mapred.FileInputFormat: Total input paths to process : 1
13/09/02 15:23:44 INFO mapred.JobClient: Running job: job_201308081351_0210
13/09/02 15:23:45 INFO mapred.JobClient:  map 0% reduce 0%
13/09/02 15:23:54 INFO mapred.JobClient:  map 50% reduce 0%
13/09/02 15:23:56 INFO mapred.JobClient:  map 100% reduce 0%
13/09/02 15:24:02 INFO mapred.JobClient:  map 100% reduce 100%
13/09/02 15:24:03 INFO mapred.JobClient: Job complete: job_201308081351_0210
...
13/09/02 15:24:03 INFO mapred.JobClient:     Reduce input groups=33782
13/09/02 15:24:03 INFO mapred.JobClient:     Combine output records=42269              (0)
13/09/02 15:24:03 INFO mapred.JobClient:     Map output records=215133
13/09/02 15:24:03 INFO mapred.JobClient:     Map input records=22108
13/09/02 15:24:03 INFO mapred.JobClient:     Reduce shuffle bytes=614624         (2521615)
13/09/02 15:24:03 INFO mapred.JobClient:     Combine input records=215133              (0)
13/09/02 15:24:03 INFO mapred.JobClient:     Spilled Records=84538                (430266)
13/09/02 15:24:03 INFO mapred.JobClient:     Reduce input records=42269           (215133)
13/09/02 15:24:03 INFO mapred.JobClient:     Reduce output records=33782
13/09/02 15:24:03 INFO mapred.JobClient:     Map output materialized bytes=614624 (2521615)
...
```

WordCount Improvement

- Using a single reducer can be a bottleneck - why ?
  - Work can be distributed over multiple nodes (better workbalance)
  - All the input data has to be sorted before processing
  - Question: Which data should be send to which reducer ?
- Possible Solutions:
  - Arbitrary distributed, based on a hash function (default mode)
  - Partitioner Class, to determine for every output tuple the corre-sponding reducer

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

User provided Partitioner Class

- Example:

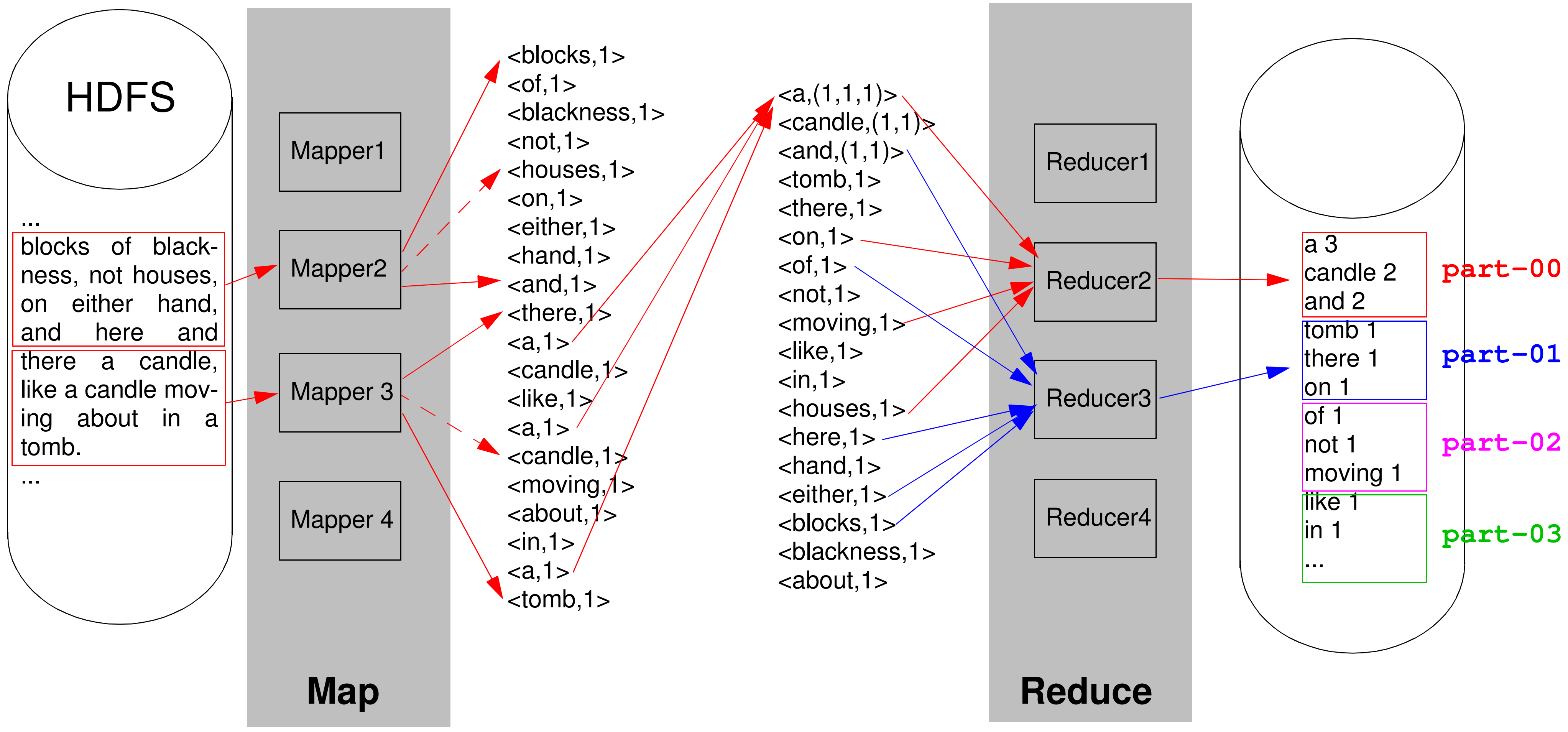
```
public class MyPartitioner implements Partitioner<Text, Writable> {
    @Override
    public void configure(JobConf job) {}

    @Override
    public int getPartition(Text key, Writable value, int numPartitions) {
        return ((int) key.charAt(0)) % numPartitions;
    }
}
```

group by first character of word

- in main program:

```
public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
        conf.setNumReduceTasks(4);
        conf.setPartitionerClass(MyPartitioner.class);
        ...
}
```


HDFS
...
blocks of blackness, not houses, on either hand, and here and
there a candle, like a candle moving about in a tomb.
...
Mapper1
Mapper2
Mapper 3
Mapper 4
Map
<blocks,1>
<of,1>
<blackness,1>
<not,1>
<houses,1>
<on,1>
<either,1>
<hand,1>
<and,1>
<there,1>
<a,1>
<candle,1>
<like,1>
<a,1>
<candle,1>
<moving,1>
<about,1>
<in,1>
<a,1>
<tomb,1>
<a,(1,1,1)>
<candle,(1,1)>
<and,(1,1)>
<tomb,1>
<there,1>
<on,1>
<of,1>
<not,1>
<moving,1>
<like,1>
<in,1>
<houses,1>
<here,1>
<hand,1>
<either,1>
<blocks,1>
<blackness,1>
<about,1>
Reducer1
Reducer2
Reducer3
Reducer4
Reduce
a 3
candle 2
and 2
tomb 1
there 1
on 1
of 1
not 1
moving 1
like 1
in 1
...
part-00
part-01
part-02
part-03

MapReduce Streaming API

- Write Map and Reduce functions in any language you like, i.e. unix commands, python-scripts, php-scripts, perl-scripts, ...
- Very good for quick prototyping
- Input, Output via STDIN, STDOUT
- Data must be text, each line is a record (Key/value-pair)
- Performance is not so good as with the Java-API
- Input to reducer is not (key_2, list($value_2$)) but list(key_2, $value_{2<x>}$)

Example dataset:

```
Aachen,D,Nordrhein Westfalen,247113,NULL,NULL
Aalborg,DK,Denmark,113865,10,57
Aarau,CH,AG,NULL,NULL,NULL
Aarhus,DK,Denmark,194345,10.1,56.1
Aarri,WAN,Nigeria,111000,NULL,NULL
Aba,WAN,Nigeria,264000,NULL,NULL
Abakan,R,"Rep. of Khakassiya",161000,NULL,NULL
Abancay,PE,Apurimac,NULL,NULL,NULL
Abeokuta,WAN,Nigeria,377000,NULL,NULL
Aberdeen,GB,Grampian,219100,NULL,NULL
Aberystwyth,GB,Ceredigion,NULL,NULL,NULL
Abidjan,CI,"Cote dIvoire",NULL,-3.6,5.3
Abilene,USA,Texas,108476,-99.6833,32.4167
Abu Dhabi,UAE,United Arab Emirates,363432,54.36,24.27
Abuja,WAN,Nigeria,NULL,NULL,NULL
Acapulco,MEX,Guerrero,515374,NULL,NULL
Acarigua,YV,Portuguesa,116551,NULL,NULL
Accra,GH,Ghana,867459,-0.2,5.55
Acheng,TJ,Heilongjiang,197595,NULL,NULL
Achinsk,R,Krasnoyarskiy kray,123000,NULL,NULL
```

Streaming API with unix commands

- Question: How many cities has each country ?

```
hadoop jar /mnt/biginsights/opt/ibm/biginsights/pig/test/e2e/pig/lib/hadoop-streaming.jar \
                   -input input/city.csv  \
                   -output output \
                   -mapper "cut -f2 -d," \
                   -reducer "uniq -c"\
                   -numReduceTasks 5
```

- Explanation:

```
    cut -f2 -d,    # Extracts the second column in a comma (,) separated list of values

    uniq -c        # Filter adjacent matches matching lines from INPUT,
                   #    -c: prefix lines by the number of occurrences

    additional remark: # numReduceTasks=0: no shuffle & sort phase !!!
```

Output(s)

```
After mapper:          After shuffle & sort:          After reduce:

D                      A                              3 A
DK                     A                              2 AL
CH                     A                              1 AND
DK                     AL                             4 ANG
WAN                    AL                             9 AUS
WAN                    AND                            1 AZ
R                      ANG                            9 B
PE                     ANG                            8 BD
WAN                    ANG                            1 BDS
GB                     ANG                            1 BI
GB                     AUS                            1 BOL
CI                     AUS                            70 BR
USA                    AUS                            1 BRU
UAE                    AUS                            1 BZ
WAN                    AUS                            6 C
MEX                    AUS                            7 CAM
YV                     AUS                            12 CDN
GH                     AUS                            13 CH
..                     ...                            1 CI
```

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Discussion

- Same operation as:

```
select count(*), code
  from city
 group by code
```

- Same operation as ... ?

```
cut -f 2 -d , ../data/Mondial/city.csv | uniq -c

cut -f 2 -d , ../data/Mondial/city.csv | sort | uniq -c
```

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft UNIVERSITY OF APPLIED SCIENCES

Streaming API with perl scripts - Mapper

Mapper script:

```
#! /usr/bin/perl -w

while (my $line = <STDIN>) {
        @words = split(" ", $line);
        for my $word (@words) {
            if ($word) {
              print "$word\t1\n";
            }
        }
}
```

Call:

```
hadoop jar /mnt/.../lib/hadoop-streaming.jar \
    -input input/MobyDick.txt  \
    -output output \
    -mapper "perl_mapper_example.pl" \
    -file perl_mapper_example.pl\
    -numReduceTasks 0
```

Hadoop copies this script to every task tracker node

KIT Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft UNIVERSITY OF APPLIED SCIENCES

Output (mapper, shuffle & sort)

after mapper:

```
The         1
Project     1
Gutenberg   1
EBook       1
of          1
Moby        1
Dick        1
or          1
The         1
Whale       1
by          1
Herman      1
Melville    1
This        1
eBook       1
is          1
for         1
the         1
use         1
of          1
anyone      1
anywhere    1
...
```

setting `-numReduceTasks=1` (after shuffle & sort):

```
A           1
A           1
...
A           1
ABOUT       1
ACCOUNT     1
ACCOUNT     1
ACTUAL      1
ADDITIONAL  1
ADVANCING   1
ADVANCING   1
ADVENTURES1
AFFGHANISTAN1
AFRICA      1
AFTER       1
AGAINST     1
AGREE       1
AGREE       1
AGREEMENT   1
AHAB        1
AHAB        1
```

compared with Java-API

```
A              (1,1,...,1)
ABOUT          (1)
ACCOUNT        (1,1)
ACTUAL         (1)
ADDITIONAL     (1)
ADVANCING      (1,1)
ADVENTURES     (1)
AFFGHANISTAN   (1)
AFRICA         (1)
AFTER          (1)
AGAINST        (1)
AGREE          (1,1)
AGREEMENT      1
AHAB           (1,1,..,1)
...
```

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft UNIVERSITY OF APPLIED SCIENCES

Streaming API with perl scripts - Reducer

Reducer script:

```
#! /usr/bin/perl -w

my $last_key="";
my $sum;
while (my $line = <STDIN>) {
    my ($key, $value) = split("\t", $line);
    if ($key ne $last_key) {
        if ($last_key) {
            print "$last_key\t$sum\n";
        }
        $last_key = $key;
        $sum = 1;
    } else {
        $sum++;
    }
}
print "$last_key\t$sum\n"; # last entry
```

key has changed

Call:

```
hadoop jar /mnt/.../hadoop-streaming.jar \
    -input input/MobyDick.txt  \
    -output output \
    -mapper "perl_mapper_example.pl" \
    -reducer "perl_reducer_example.pl" \
    -file perl_mapper_example.pl\
    -file perl_reducer_example.pl \
    -numReduceTasks 1
```

Output

```
A              168
ABOUT            1
ACCOUNT          2
ACTUAL           1
ADDITIONAL       1
ADVANCING        2
ADVENTURES       1
AFFGHANISTAN     1
AFRICA           1
AFTER            1
AGAINST          1
AGREE            2
AGREEMENT        1
AHAB            10
AK               1
ALFRED           1
ALGERINE         1
ALIVE            1
ALL              9
ALONE            2
AM               2
AMERICA          1
AMONG            1
```

Hadoop Distributed Cache

- Files, specified to be in the *Distributed Cache*, are distributed to all TaskTracker nodes
- How to add files to distributed cache ?
  - With Java-API: `DistributedCache.addCacheFile(<filename>, conf)`
  - Use the `-file` option with the hadoop command line call (as seen before)
- Purpose:
  - Send scripts to TaskTracker nodes (as used with streaming api)
  - Configuration files (i.e. stopword list)
  - Data for mapper-side joins
  - Additional libraries
  - ...

Complex Workflows

- Chaining multiple MapReduce Jobs
  - complete MapReduce jobs in sequence (using multiple `JobClient.run-Job(conf)`-method calls in sequence)

    ```
    mapreduce1 | mapreduce2 | ...
    ```
  - Chaining Mappers and Reducers in arbitrary order (using ChainMapper, Chain-Reducer-classes)

    ```
    map1 | map2 | reduce1 | map3 | map4 | reduce 2
    ```
  - with complex dependencies

    ```
    i.e. job2.addDependencyJob(job1); // job2 waits until job1 finishes
    ```

Lämmel, 2007: Five Simple Concepts

1. Iteration over the Input data
2. Computing key/value pairs from every piece of input
3. Grouping all intermediate results by key
4. Iterating over resulting groups
5. Reducion of each group

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Lämmel, 2007: Five Simple Concepts

1. Iteration over the Input data
2. Computing key/value pairs from every piece of input

Mapper

3. Grouping all intermediate results by key

Shuffle & Sort

4. Iterating over resulting groups
5. Reducion of each group

Reducer

Practical Exercise

Appendix A, Part 2 and 3 - MapReduce

Hadoop Extensions

Pig

- High level data processing language which abstracts from Map-Reduce paradigma
- Yahoo! runs about 40% of all Hadoop jobs with Pig
- Language: Pig latin
  - operation on complex, nested data structures
  - optional schema
  - Support for complex datatypes like bags, tuples (also nested)
  - Support of user defined functions (UDF)
  - Number of relational operators
  - Compiler translates Pig Latin to Hadoop MapReduce Jobs (optimizer)
- Application fields: logfile analysis, natural language processing, analyzing network graphs, ...

Pig

- How to run Pig ?
  - Script (batch) - with named parameters
  - interactively using `grunt>` command line
  - embedded in Java
- datatypes:
  - simple: int, long, float, double, chararray, bytearray, ...
  - complex: tuple, bag, map
- Commands:
  - HDFS file commands: cat, ls, copyFromLocal, ...
  - Data read/write: load, store, dump, limit
  - Utility: kill, exec[1], run, help, quit, set, describe, illustrate, explain
  - Relational: split, union, filter, distinct, sample, foreach, group, join, cogroup, ...

1. execute in separate space

Pig Examples

```
cities = LOAD 'data/mondial/city.csv' using PigStorage(',') as (
                                          name:chararray,
                                          country:chararray,
                                          province:chararray,
                                          population:long,
                                          longitude:float,
                                          latitude:float);

countries = LOAD 'data/mondial/country.csv' using PigStorage(',') as (
                                          name:chararray,
                                          code:chararray,
                                          capital:chararray, x,y,z);

population_cities = FILTER cities by population is not null;

SPLIT population_cities INTO french_cities IF country=='F',
                             us_cities IF country=='USA',
                             other_cities OTHERWISE;

country_cities = JOIN french_cities BY country, countries BY code;
STORE contry_cities INTO 'output/cities;
```

```
grp_country_cities = GROUP population_cities BY country;

avg_pop = FOREACH grp_country_cities GENERATE group,
                                              AVG(population_cities.population),
                                              COUNT(population_cities);

rm output;
STORE avg_pop into 'output/avg_population';

/* give some information about the datatypes */
describe population_cities;
describe grp_country_cities;
describe avg_pop;

-- illustrate: sample run with a small sample to show step by step wat happended
illustrate avg_pop;

-- output to stdout
dump avg_pop;
```

Pig - Summary

- Data driven language, with optional schema (schema on read)
- `Store` and `dump` statements trigger the evaluation of a script/interactive commands
- Pig compiler translates the commands in optimized MapReduce commands
- Greatly simplifies joining of datasets and job chaining compared to pure MapReduce programs

Hive

- Facebook development
- Data warehouse infrastructure build on top of Hadoop
- SQL-like language (HiveQL)
- Converts SQL queries to MapReduce jobs
- Works on structured data (Schema required - Metastore)
- Metastore resides in a relational database
- Supports concepts tables, rows, columns, schema
- Tables (many data formats and sources are supported):
  - managed tables
  - external tables

Hive example queries

- Restriuctions of HiveQL: No support for sql update, delete, transactions, indexes, correlated subquery, subquery outside from, stored procedures
- Batch process oriented (low latency)
- Supports table partitioning (create clause), [sorted] bucketing (clustering), sam-pling
- Storage format:
  - delimited text (default)
  - SerDe (serialized/desrialized)
  - Binary SerDe
    - Row oriented (Sequence file)
    - column oriented (RCFile)

Hive

- Hive command line interface, web interface, Clients likeJDBC/ODBC, Eclipse
- Supported datatypes:
  - primitive datatypes:
    - tinyint,smallint, int, bigint, float, double
    - boolean
    - string, binary
    - timestamp
  - Collections:
    - array
    - struct (i.e. struct<f1: int, f2:arry<string>>)
    - map (i.e. map<int, string>)
    - union (i.e. uniontype <int, string, dounle>)

Hive

- Partitioning based one one or more columns
- Bucketing (grouping of data based on hashing).
- Table are regular HDFS files
- support of „external tables“
- DDL for creating schemas/tables
- Different storage formats allowed:
    - Textfile,
    - SequenceFile,
    - RCFile,
    - Hadoop specific input/output formats

Hive: database/table creation

```
drop database if exists mondial cascade;
create database mondial location '/user/idcuser/data/mondial.db';
use mondial;
grant create on database mondial to user idcuser;

create table country (
   name string,
   code string,
   capital string,
   province string,
   area int,
   population int
) row format delimited fields terminated by ',' stored as textfile;

load data local inpath '../data/Mondial/country.csv'
overwrite into table country;

select count(*)
from country;
```

Table creation/datatypes

```
create table country_with_cities (
   name string,
   code string,
   capital string,
   province string,
   area int,
   population int,
   cities array<string>
)
row format delimited
          fields terminated by ','
          collection items terminated by ':'
stored as textfile;

-- Attention: Source HDFS file is moved to Hive location !!
load data inpath 'data/mondial/country_with_cities.csv'
overwrite into table country_with_cities;
```

- File: countries-with_cities.csv:

```
Germany,D,Berlin,Berlin,356910,83536115,Karlsruhe:Berlin:Stuttgart:Koeln:Potsdam:Muenchen
Austria,A,Vienna,Vienna,83850,8023244,Vienna:Insbruck:Klagenfurt:Graz
Denmark,DK,Copenhagen,Denmark,43070,5249632,Alborg:Copenhagen:Esbjerg:Odense
```

Hive querying

```
select con.name, count(cit.name), avg(cit.population) avg
from country con join city cit on (cit.country=con.code)
group by con.name
having count(cit.name) > 10
sort BY avg desc
;
Total MapReduce jobs = 3
Launching Job 1 out of 3
Starting Job = job_201308081351_0186, Tracking URL = http://vhost1416.site2.com-
pute.ihost.com:50030/jobdetails.jsp?jobid=job_201308081351_0186
Kill Command = /mnt/biginsights/opt/ibm/biginsights/IHC/libexec/../bin/hadoop job  -
Dmapred.job.tracker=vhost1416.site2.compute.ihost.com:9001 -kill job_201308081351_0186
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2013-08-19 10:12:35,737 Stage-1 map = 0%,  reduce = 0%
2013-08-19 10:12:38,750 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 0.64 sec
2013-08-19 10:12:39,773 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 0.64 sec
...
Egypt   22      895400.0
India   38      847594.0
China   81      660571.8518518518
Iraq    15      635249.4545454546
```

Further Hadoop Subprojects

- HBase: Distributed, column-store database (à la Googles BigTable)
- ZooKeeper: Relable coordination system for managing state between applications
- Mahout: Scalable machine learning library (clustering, classification)
- Flume: Distributed service for gathering a set of log-files on different machines in a cluster and aggregate them to a persitent store like HDFS.

Practical Exercise

Appendix A, Part 4- Pig and Hive

Sources

- Chuk Lam, Hadoop in Action, Manning, 2011
- Alex Holmes, Hadoop in Practice, Manning, 2012
- Thomas Kiencke: Hadoop Distributed File System (HDFS). http://media.itm.uni-luebeck.de/teaching/ws2012/sem-sse/thomas-kiencke-hdfs-ausarbeitung.pdf
- Lämmel, R. Google's mapreduce programming model — revisited. Sci. Comput. Program., 68(3):208–237.

Appendix

- A - Exercises
- B - Hive Exercise Guide
- C - Pig Exercise Guide

DataSys-2013 Tutorial: The Hadoop Core – Understanding Map Reduce and the Hadoop Distributed File System

Appendix A: Exercises

Daniel Kimmig[1], Andreas Schmidt[1,2]
(1) Karlsruhe Institute of Technology / (2) Karlsruhe University of Applied Sciences
(daniel.kimmig@kit.edu, andreas.schmidt@kit.edu)

To perform the tasks in this exercises you need a computer with internet connection and a ssh-client (on windows you can use putty.exe or the ssh command from cygwin). See the appendix for links. To connect to the Hadoop cluster, execute the following command:

```
ssh -l immm-user<x> 129.35.213.141
```

or (if you are using putty)

```
putty.exe -ssh -l immm-user<x> 129.35.213.141
```

Part 1 (HDFS):

a) Enter the command `hadoop fs` and look at the output. It lists the possible co-mands for the Hadoop Distributed File System (HDFS) command line tool (see also [1]). To get more info about a special command you can type:

```
hadoop fs -help setrep
```

b) List all the files in your HDFS root directory.

c) Because the commands are quite long, we define an alias. Type

```
alias hfs='hadoop fs '
```

at the command line and then try the following short notation:

```
hfs -ls /
```

d) Create a new directory ’data’ in your HDFS home directory:

e) Copy from the local /data/immm-2013-tutorial/Mondial directory the file country2.csv to the newly created data directory in your HDFS file system.

f) Also copy the file city.csv from the same directory to the HDFS data directory.

g) Take a look at the content of the new created directory (country.csv and city.csv should be inside).

h) Set the replication factor of the file country.csv to 5: Check if the operaton was performed successfully (with the hadoop fs -ls command). Browse to http://vhost1416.site2.compute.ihost.com:50070/dfshealth.jsp. Klick onto the Link Browse the filesystem, browse to the 'data' directory and look at the files and replication factors.

i) Create a new directory under 'data' with the name 'imprtant', copy some files in it and look at the replication factor. Now change the replication factor of the directory and look again of the replication factor of the files. What happend when you copy another file in this directory ?

j) Look with the browser at the content of a file under the 'data' directory in HDFS. Here you can also see on which nodes the replications reside.

k) Remove the entire 'important' directory with its content.

Part 2 (MapReduce - Java-API):

a) Copy the file /data/immm-2013-tutorium/pg2701.txt to input/MobyDick.txt and list the content of the file, than change to the local (not HDFS) directory ~/HadoopTutorial. Here you can find the file 'makefile'. The makefile contains a number of commands for compiling and executing different MapReduce jobs. Use the command `less`[1] and take a look at the content of the file (`less makefile`). Inside the file you can find the target „runWordCount“:

```
runWordCount: wordcount.jar                                          (1)
        $(call remove_if_dir_exists,output)                          (2)
        hadoop jar wordcount.jar org.myorg.WordCount \
                -D mapred.reduce.tasks=2 input/MobyDick.txt output   (3)
```

The target does the following things:

Line 1: Before executing the commands starting with line 1, the target calls another target (wordcount.jar). This target checks if the Library `wordcount.jar` is up to data, and if not, recompiles the sourcefile `WordCount.java`. After the depended target has been checked it runs the following commands:

Line 2: removes the hdfs directory `output` (if exists)[2]
Line 3: executes the hadoop job *org.myorg.WordCount*. Input is the File MobyDick.txt inthe input directory, results are written to the directory output. The number of reducer tasks is 2.

b) Execute the following command: `make runWordCount`. After the program finished, take a look at the content of the result file(s) in the `output` directory. Add these com-

1. or an editor like xemacs, vi or vim.
2. `remove_if_file_exists` and `remove_if_dir_exists` are makefile macro, defined in the lower part of the makefile. They check, if the file/directory already exist, and if they exist, remove them. They use hadoop HDFS commands to execute this job.

mands at the end of the `runWordCount` target.

c) Change the value of mapred.reduce.tasks to 20 and run the make command again. Again take a look at the content of the hdfs `output` directory.

d) Load the file WordCount.java and uncomment the line

```
//      conf.setPartitionerClass(MyPartitioner.class);
```

as well as the `MyPartitionerClass`. Save the file and run the `make runWordCount` command again. What happend and why? Take a look at the method `getPartition(...)` in the `MyPartitioner class`.

While running the MapReduce program, take a look at http://vhost1416.site2.compute.ihost.com:50030/jobtracker.jsp and look what is displayed (you have to press reload to see changes).

e) Copy also the files `/data/immm-2013-tutorial/pg100.txt` and `pg2600.txt` to the `input` directory. Change the input parameter to the hadoop command in the makefile from `input/MobyDick.txt` to `input/*.txt`. What happend ?

Part 3 (MapReduce - Streaming API):

a) Issue the command `less makefile` and go to the target „runGroupByStreaming“

```
runGroupByStreaming:
    $(call remove_if_file_exists,input/city.csv)                    (1)
    hadoop fs -copyFromLocal /data/immm-2013-tutorial/Mondial/city.csv input/ci-
ty.csv                                                              (2)
    $(call remove_if_dir_exists,output)                             (3)
    hadoop jar /mnt/biginsights/opt/ibm/biginsights/pig/test/e2e/pig/lib/ \
hadoop-streaming.jar \                                              (4)
           -input input/city.csv  \
           -output output \
           -mapper "cut -f2 -d," \
           -reducer "uniq -c" \
           -numReduceTasks 1
```

The first three lines copy the `input` to the right destination and remove the `output` directory (this is automatically created from the mapReduce job).
In line 4, the MapReduce job is startet. The streaming api implementation is found in the `hadoop-streaming.jar` library. The streaming api expects the input file(s) or directory, the output directory, as well as the mapper and reducer commands. Optionally, you can specify the number of reducers to run.

If you aren’t familar with the cut and uniq command, execute the following commands to get an idea of what these command are doing:

```
$ man cut
$ less /data/immm-2013-tutorial/Mondial/city.csv
$ cut -f2 -d, /data/immm-2013-tutorial/Mondial/city.csv
$ cut -f2,1 -d, /data/immm-2013-tutorial/Mondial/city.csv
```

and

```
$ man uniq
$ less  /data/immm-2013-tutorial/unique-testdata.txt
$ uniq  /data/immm-2013-tutorial/unique-testdata.txt
$ uniq  -c /data/immm-2013-tutorial/unique-testdata.txt
```

b) Analyze the above hadoop streaming command with the given mapper and reducer. What do you expect as result?

c) Execute `make runGroupByStreaming`

   Inspect the output and then change the value of the parameter -numReduceTasks in the makefile to 0. Rerun the `make runGroupByStreaming` command and inspect the output. What happend ?

   Next, change the value numReduceTasks back to a value greater zero but change the reducer parameter value to '`cat`' (the `cat` command only displays the result of the prior mapper phase, but do not change anything). What is the difference to the previous output and why? How does the number of numReduceTasks affects the result?

d) For profis: Copy the file `/data/immm-2013-tutorial/cite75_99.txt` to your hdfs `input` directory. The file contains the information, which patent is cited by which other patent (the cited patent is in the second column). Write a MapReduce program (i.e. inside the makefile), that returns for each patent, the number of patents, that cites it.

e) Extend this program, so that the 100 most cited patents are returned. Hint: take a look at the `tail` or `head` command (`man tail, man head`).

Part 4 (Pig and Hive):

a) Copy the directory `/data/immm-2013-tutorial/Mondial` to the hdfs directory data. Write a pig script, that retuns all cities with more than 5 Million inhabitants, ordered by population descendant.

b) Write a HiveQL Statement, which does the same job.

Resources:

[1] hadoop hdfs commands: http://hadoop.apache.org/docs/stable/file_system_shell.html

[2] Hadoop Hive, Language manual: https://cwiki.apache.org/confluence/display/Hive/LanguageManual

[3] Pig Latin, Reference manual: http://pig.apache.org/docs/r0.7.0/piglatin_ref2.html

[4] Pig cookbook: http://pig.apache.org/docs/r0.7.0/cookbook.html

[5] Programming in Pig. Sample chapter from the Book: Hadoop in Action, Manning, 2010: http://www.manning.com/lam/SampleCh10.pdf

Links:

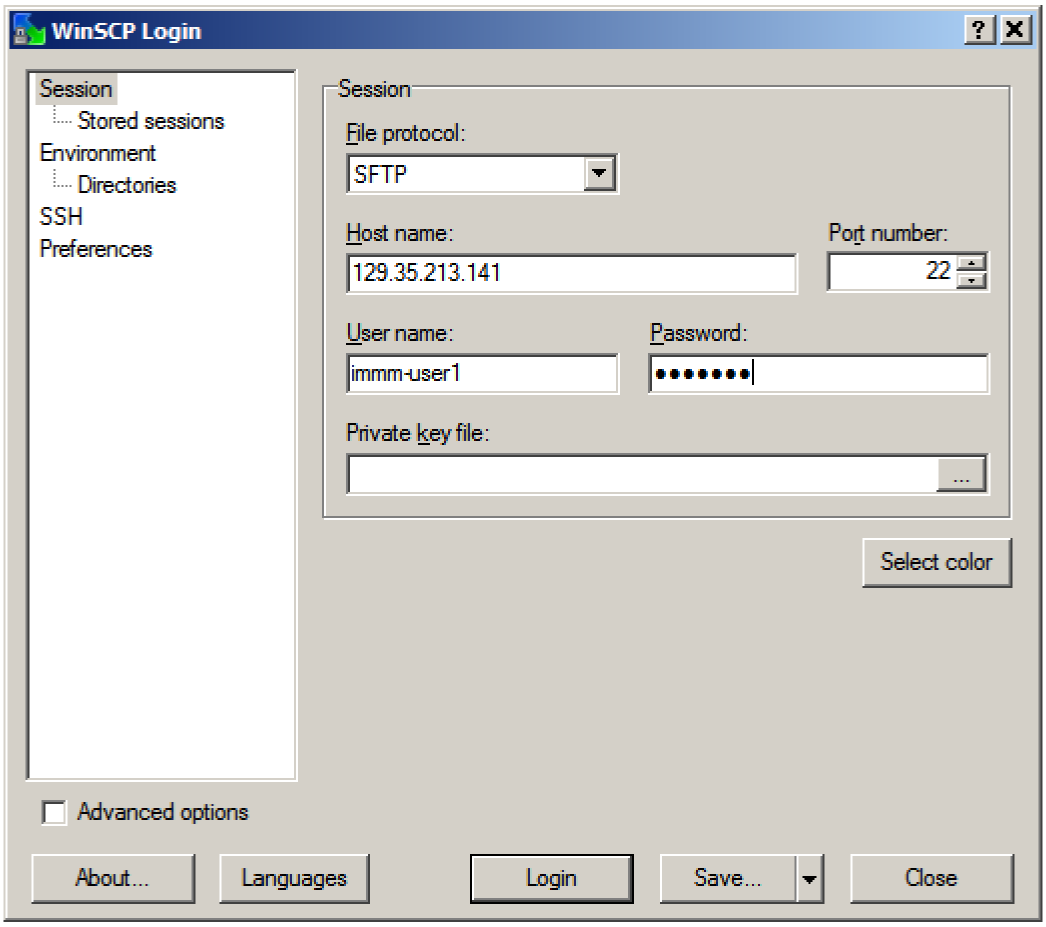
putty: `http://www.putty.org/`

cygwin: `http://www.cygwin.com/`

vncviewer: `http://www.realvnc.com/download/viewer/`

Bundle of useful programs: `http://www.smiffy.de/immm-2013/`

Configuration WinSCP::



Session

File protocol: SFTP

Host name: 129.35.213.141

Port number: 22

User name: immm-user1

Password: •••••••

Private key file:

After that, type „Login“ and you should see a window with the directorory structure of the Hadoop node on the right side. Change to the „HadoopTutorial“ directory, click on the File makefile, open the context menu (right click) and chose „Edit“.

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Appendix B: Hive - Exercise Guide

0) **Loading data into HDFS**
copyFromLocal ‘filsystem-path‘ ‘hdfs-file-path‘;

1) Creating a table:
CREATE TABLE tbl_name (var1 type, var2 type)ROW FORMAT DELIMITED FIELDS TERMI-NATED BY ',' STORED AS TEXTFILE;
Supported datatypes: STRING, INT, DOUBLE, ...

2) Loading data into a table:
LOAD DATA INPATH ‘hdfs-file' OVERWRITE INTO TABLE tbl_name;

3) Looking at the the structure of a table:
DESCRIBE tbl_name;

4) Writing queries:
Any SQL statement supported by HiveQL will do ?
example: SELECT * from tbl_name WHERE name == ‘SMITH‘;

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft UNIVERSITY OF APPLIED SCIENCES

Appendix C: Pig - Exercise Guide

0) Loading data into HDFS:
copyFromLocal ‘filsystem-path‘ ‘hdfs-file-path‘;

1) Loading data into a relation:
alias = LOAD ‘hdfs-file' USING PigStorage(‘,') AS (var1:type, var2:type);
Supported datatypes: chararray, int, double, ...

2) Looking at the the structure of a relation:
DESCRIBE alias;
ILLUSTRATE alias;

3) Filtering a relation based on a predicate:
alias_filtered = FILTER alias BY expression;
expression example: name == ‘SMITH‘

4) Looking a the content of a relation:
DUMP alias;