



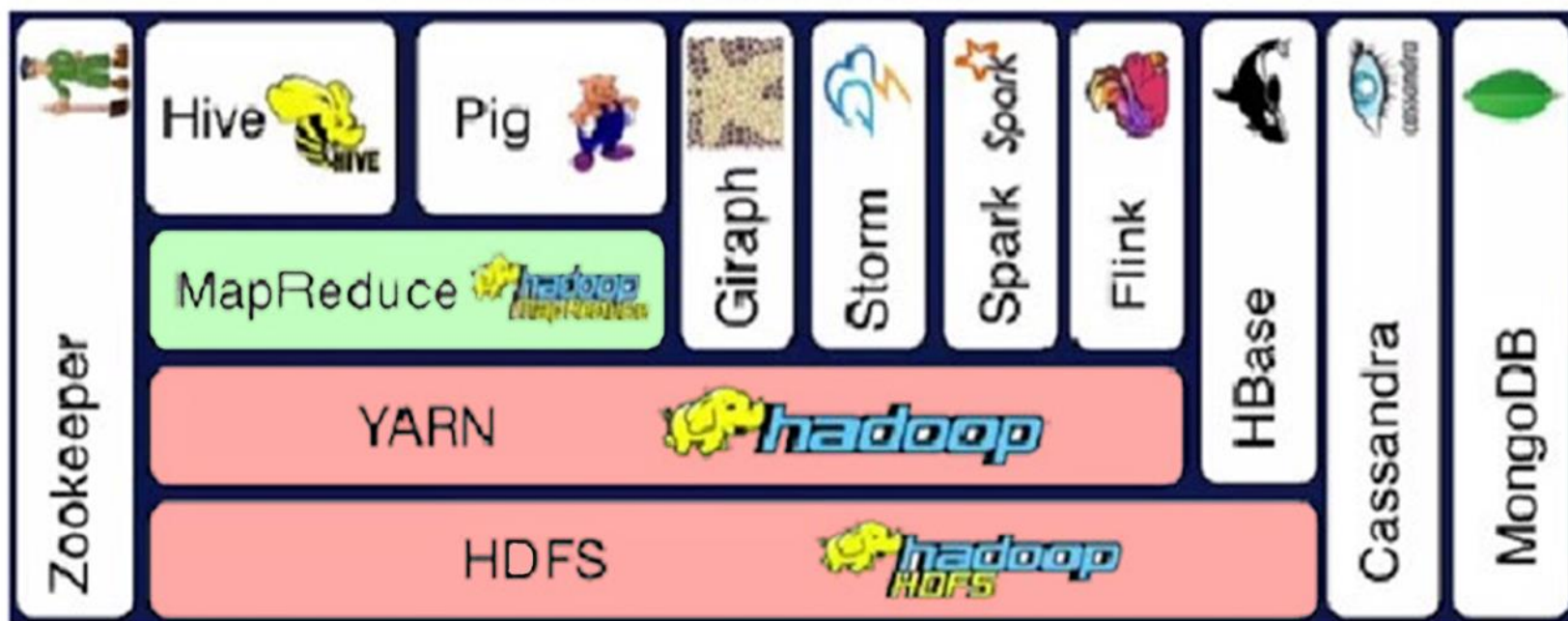
Odisee
DE CO-HOGESCHOOL

Big Data - MapReduce



Jens Baetens

Hadoop Ecosystem





Wat is map reduce?



Geschiedenis

- ▣ 2004: Jeffery Dean and Sanjay Ghemawat (Google) introduceerden het begrip in de paper: MAPREDUCE: SIMPLIFIED DATA PROCESSING ON LARGE CLUSTERS
- ▣ Essentieel onderdeel van Google Search en heeft Google groot gemaakt
 - ▣ Gebruikt tot 2014
- ▣ Originele data-processing layer van het Hadoop Ecosysteem
- ▣ Heel schaalbaar door big data te verdelen in kleinere delen
- ▣ Basis-idee van heel veel distributed toepassingen

▣ Programmeermodel

- Map Reduce van functioneel programmeren
- Split-Apply-Combine Data analysis





Voordelen

- ▣ Eenvoudig model
 - Geen zorgen om parallelisatie, data passing, race conditions, ...
- ▣ Scalable
- ▣ Structured en unstructured data
- ▣ Fault Tolerance
 - Master/Slave architectuur maakt het mogelijk om crashes te omzeilen (Automatic recovery)



Nadelen/kritieken

▣ Performance

- ▬ Niet het snelste door steeds alles van disk te lezen (HDFS)
- ▬ Wel heel schaalbaar

▣ Low-level programmeermodel

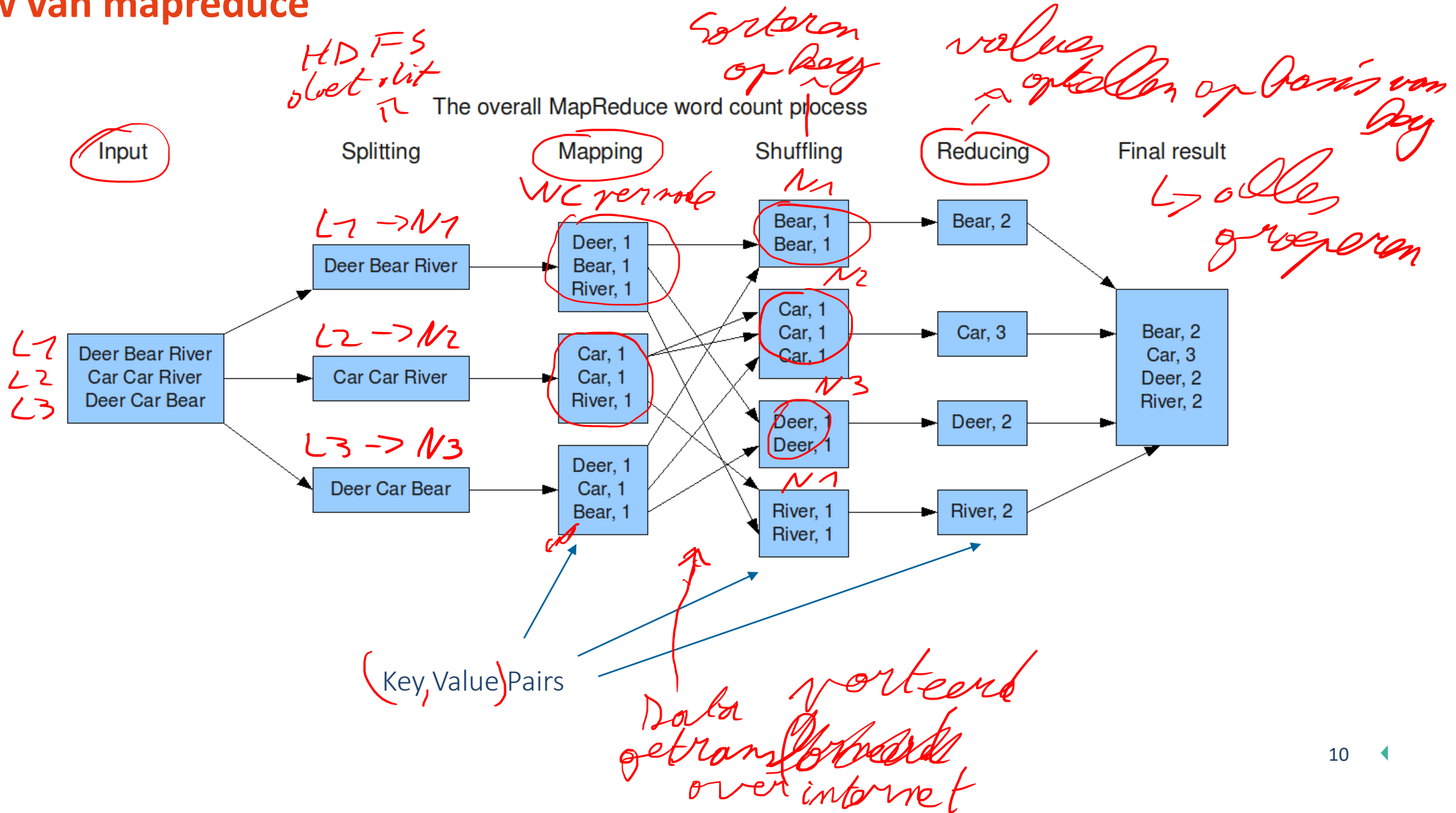
▣ Restricted framework

- ▬ Acyclic dataflow program zonder state
- ▬ Herhaaldelijk opvragen van dezelfde datasets is traag (nadelig voor ML)



Onderdelen

Flow van mapreduce



3 Operaties

Map

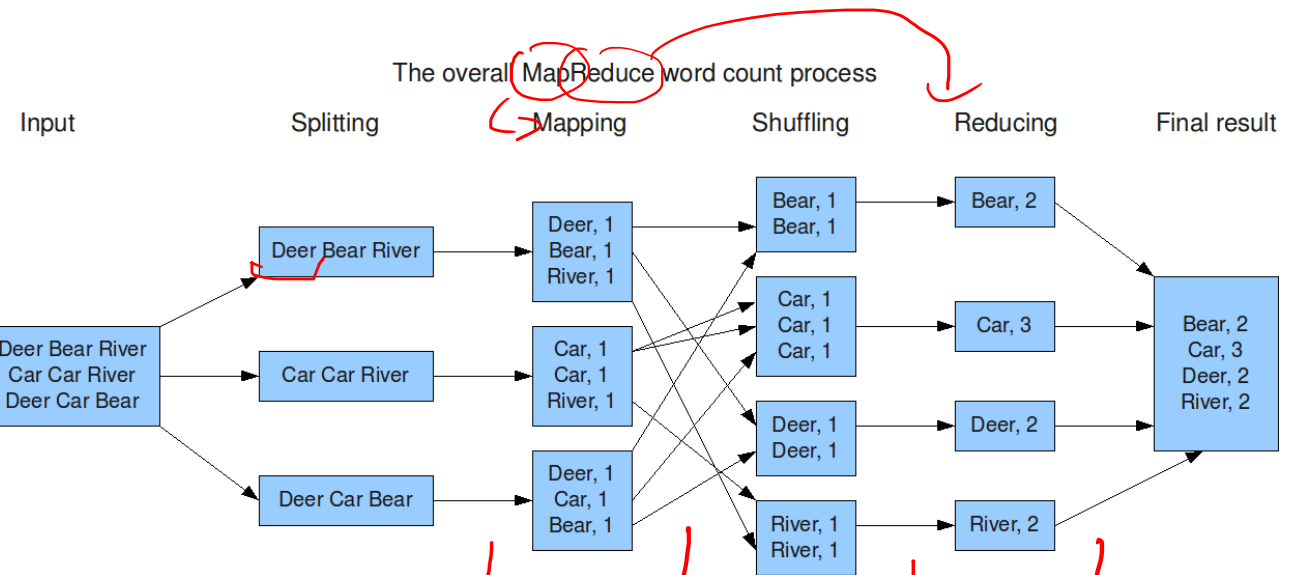
- Elke node voert dit uit op zijn data
- Lijn per lijn wordt file uitgelezen
- Master node vermijdt replica's
- (key, value) worden ~~uitgestuurd~~ en opgeslagen in tijdelijke opslag

Shuffle

- Groep alle (key, value) paren met dezelfde key op 1 node → *wordt gedaan*

Reduce

- Verwerk elke key in parallel



Wat zijn de (k, v)?
Wat doe ik met de (k, v)?

Deer (4, 1) en (Deer, 1)
Bear (4, 1) en (Bear, 1)
River (5, 1) en (River, 1)

MAP

Deer → (key, 4)
Bear → (key, 4)
River → (key, 5)

Reduce

Max(values) → 5
4, 4, 5



5 stappen

- ▣ Prepare the input (YARN)
- ▣ Run the Map() code (Zelf gecodeerd)
- ▣ Shuffle Map to Reduce (Doet MapReduce)
- ▣ Run the Reduce() code (Zelf gecodeerd)
- ▣ Produce final output (Doet MapReduce)



Performance

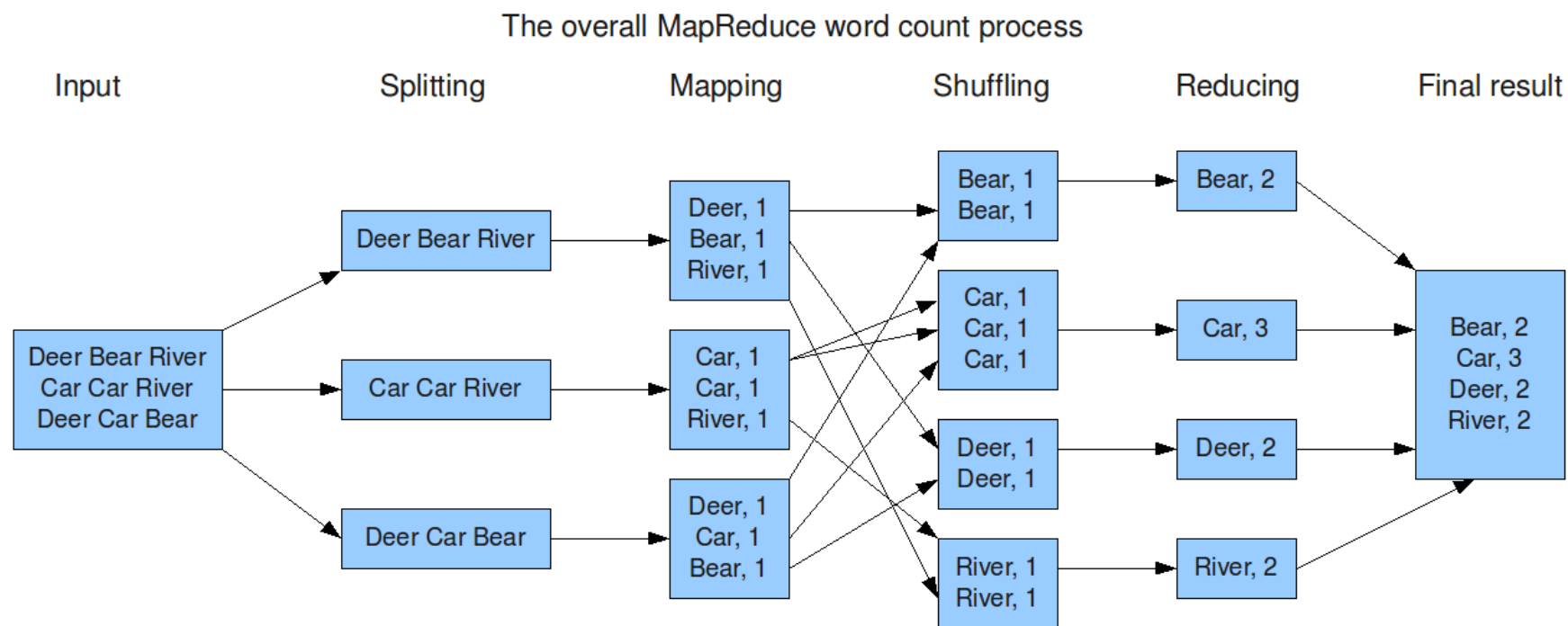
- ▣ Niet gegarandeerd snel maar wel in staat om heel veel data te verwerken
- ▣ Output of Map() wordt naar disk geschreven
- ▣ Enkel Map() en Reduce() worden parallel uitgevoerd
- ▣ Keys worden gesorteerd (kan lang duren)
- ▣ Communicatie tussen nodes zorgt ook voor delay



Fault Tolerance

- ▣ Voor taken die snel uitgevoerd worden
 - ▬ Herstarten van een enkele map() of reduce() is mogelijk
- ▣ Voor taken die langer duren kunnen tussentijdse resultaten gebruikt worden
 - ▬ Deze worden standaard opgeslagen

Typisch voorbeeld: Word Count





Inputs - Outputs

- ▣ Alles geïnterpreteerd als key-value pairs



Applicaties

Voorbeeld - wordcount

File information - input.txt



[Download](#)

[Head the file \(first 32K\)](#)

[Tail the file \(last 32K\)](#)

Block information --

Block 0 ▾

Block ID: 1073741912

Block Pool ID: BP-1244815758-127.0.1.1-1636028879278

Generation Stamp: 1088

Size: 111

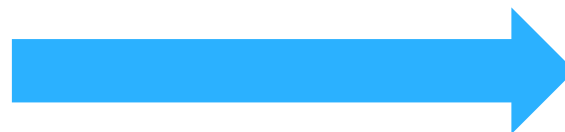
Availability:

- bigdata-VirtualBox

File contents

```
Hello World,  
hello world,  
hello world,
```

```
Dit is een voorbeeld file om het Wordcount voorbeeld te testen !
```



```
Dit 1  
Hello 1  
Wordcount 1  
World, 1  
een 1  
file 1  
hello 2  
het 1  
is 1  
om 1  
te 1  
testen. 1  
voorbeeld 2  
world, 2
```

Close



Wordcount

- ▣ Standaard geïncludeerd bij installatie met Hadoop.

```
!hadoop jar ~/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.0.jar wordcount 05_MapReduce/input.txt 05_MapReduce/output
```

Use hadoop

Execute jar

Jar containing all the examples

Which example

Input op HDFS

Output op HDFS

- ▣ Andere applicaties die reeds bestaan:

```
!hadoop jar ~/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.0.jar
```

WordCount zelf implementeren - JAVA

▣ Mapper

```
// Mapper de classes na extends Mapper<> zijn input-key, input-value, output-key, output-value
public static class TokenizerMapper extends Mapper<Object, String, String, Integer>{

    public void map(Object key, String value, Context context) throws IOException, InterruptedException {
        // value bevat de tekst in de huidige blok
        // de StringTokenizer maakt het mogelijk om over elk woord te lopen
        StringTokenizer itr = new StringTokenizer(value);
        // while-loop over alle woorden in de blocks op elke node
        while (itr.hasMoreTokens()) {
            // voeg een entry toe voor elk woord dat gezien wordt (deze worden in de reduce opgeteld)
            context.write(itr.nextToken(), 1);
        }
    }
}
```

WordCount zelf implementeren - JAVA

▣ Reducer


```
// Reducer: de classes na extends Reducer<> zijn input-key, input-value, output-key, output-value
public static class IntSumReducer extends Reducer<String,Integer,String,Integer> {

    public void reduce(String key, Iterable<Integer> values, Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (int val : values) {
            sum += val;
        }
        context.write(key, sum);
    }
}
```

WordCount zelf implementeren - JAVA

▣ Configuration

```
// configure the MapReduce program
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count java");
    job.setJarByClass(WordCount.class);
    // configure mapper
    job.setMapperClass(TokenizerMapper.class);
    // configure combiner (combiner is een reducer die op de mapping-node draait voor performantie te verbeteren)
    job.setCombinerClass(IntSumReducer.class);
    // configure reducer
    job.setReducerClass(IntSumReducer.class);
    // set output key-value classes
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    // set input file (first argument passed to the program)
    FileInputFormat.addInputPath(job, new Path(args[0]));
    // set output file (second argument passed to the program)
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    // In this case, we wait for completion to get the output/logs and not stop the program to early.
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

- 
- ▣ Voor het maken van deze applicaties heb je de volledige java programmeertaal tot je beschikking.
 - ▣ Meer informatie over de specifieke Mapper/Reduce/ Combiner klassen vind je hier:
<https://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapred/package-summary.html>


Track application state

- Yarn tracked de uitgevoerde applicaties en houdt hun logs bij.

bigdata-virtualbox:8088/cluster/apps

90% ☆

🔒



Cluster

About

Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Res
2	0	0	2	0	0 B	4 GB	0 B

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes
1	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=M), vcores, vram (unit=G)]	<memory:1024, vCores:1>	<memory:4096, vCores:4, vram: 4G>

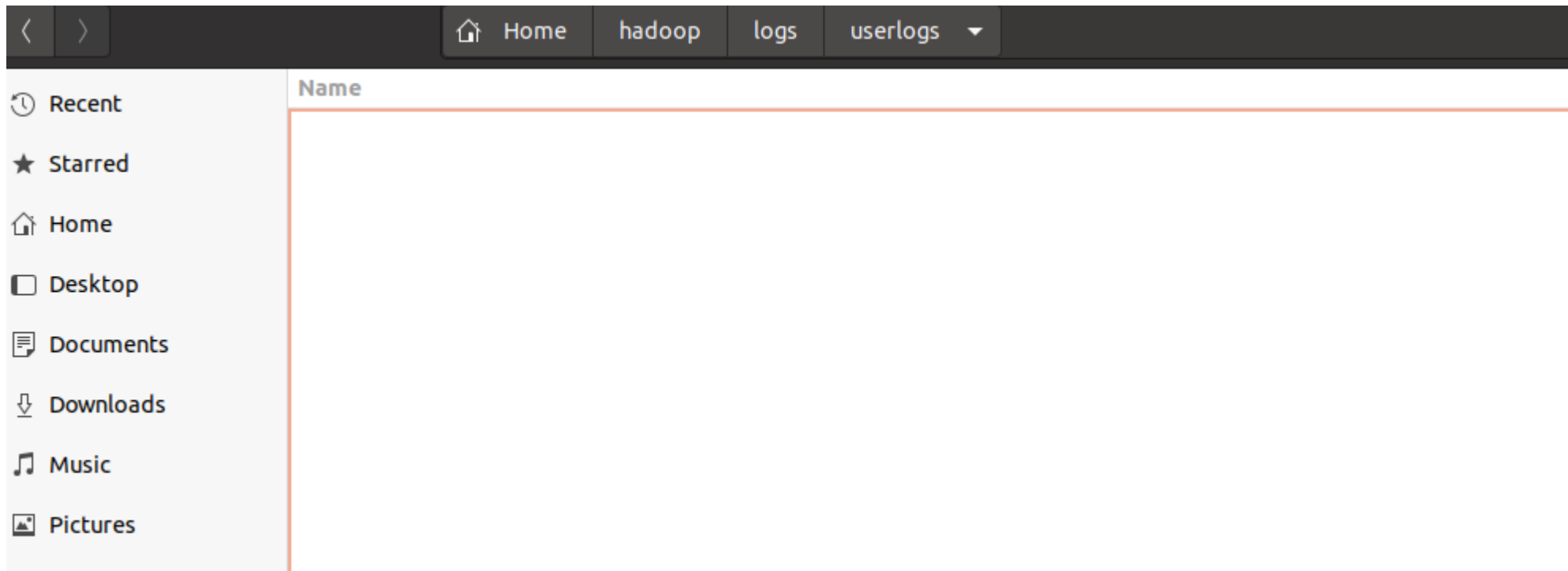
Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCo	Allocated Memory MB
application_1636041221550_0002	bigdata	word mean	MAPREDUCE		default	0	Thu Nov 4 17:10:51 +0100 2021	Thu Nov 4 17:10:51 +0100 2021	Thu Nov 4 17:11:17 +0100 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1636041221550_0001	bigdata	word count	MAPREDUCE		default	0	Thu Nov 4 16:53:58 +0100 2021	Thu Nov 4 16:54:01 +0100 2021	Thu Nov 4 16:54:39 +0100 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A

Showing 1 to 2 of 2 entries

Logs

- ▣ Of je kan ze ook in je Hadoop – folder vinden onder logs/userlogs





Mapreduce coderen in Java

- ▣ Maximale flexibiliteit
- ▣ Snelste uitvoering
- ▣ Beste integratie met Hadoop

API's voor andere programmeertalen

▣ Hadoop Streaming API

- ▬ Door gebruik te maken van stdin en stdout om te communiceren
- ▬ MrJob, dumbo, hadooppy

▣ Hadoop Pipes

- ▬ C++ interface voor MapReduce, communicatie via sockets
- ▬ pydoop

▣ Non-Hadoop

- ▬ Disco, octopy



Mrjob (Hadoop Streaming)

▣ Voordelen

- ▬ Uitgebreide documentatie
- ▬ Code kan lokaal uitgevoerd worden
- ▬ Veel management automatisch
- ▬ Werkt met cloud toepassingen
 - Google Dataproc
 - Amazon Elastic

▣ Maar beschikt niet over de volledige Hadoop API

Mrjob (Hadoop Streaming)

```
from mrjob.job import MRJob

class MRWordCount(MRJob):
    def mapper(self, _, line):
        for word in line.split():
            yield (word, 1)

    def reducer(self, word, counts):
        yield (word, sum(counts))

if __name__ == '__main__':
    MRWordCount.run()
```

```
!python wordcount_mrjob.py -r hadoop hdfs:///user/bigdata/05_MapReduce/input.txt > output.txt
```



Pydoop

- ▣ Meer informatie / API / Examples:
<https://crs4.github.io/pydoop/index.html>
- ▣ Beschikt in tegenstelling tot Mrjob wel over de volledige Hadoop Interface.
- ▣ Kan ook aanpassen hoe de bestanden ingelezen worden (bvb niet lijn per lijn)
- ▣ Pydoop script om python code uit te voeren

Pydoop script

```
def mapper(_, text, writer):  
    for word in text.split():  
        writer.emit(word, 1)  
  
def reducer(word, icounts, writer):  
    writer.emit(word, sum(icounts))
```

```
!pydoop script wordcount_script.py 05_MapReduce/input.txt 05_MapReduce/output_python_script
```



<https://youtu.be/bcjSe0xCHbE?list=PLQ4IP5IBsAQdCdLCKyUTyOQMR7Nyv3Wfk>