Big Data
02
Hadoop

Eric MSP Veith <veith@offis.de>

March 2, 2020

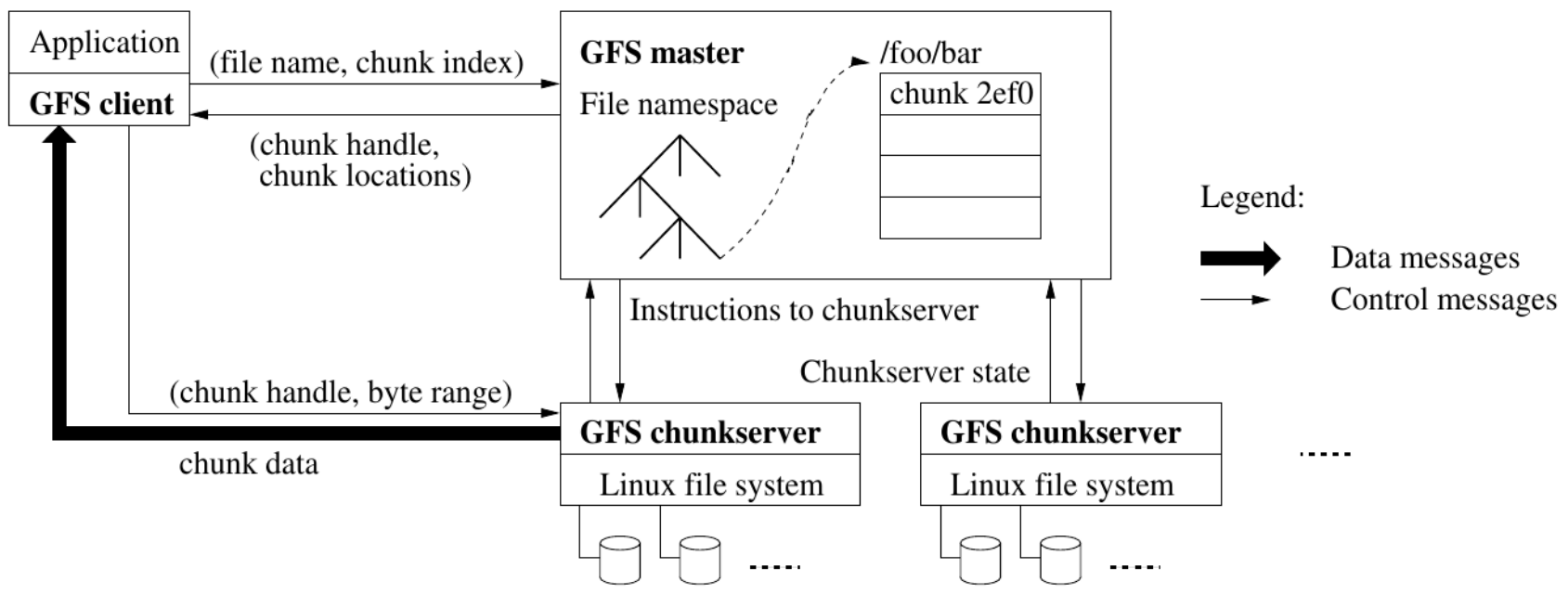
Organisatorisches

- Prüfungsleistung: Seminararbeit
  - 10 Seiten Ausarbeitung zum Thema: Arial o.ä., Schriftgröße 12pt, einfacher Zeilenabstand, 2cm Rand, keine separate Titelseite, Seitenzahl inkl. Literaturverzeichnis
  - 15 Minuten Vortrag zum Thema
- Seminarthemen & Folien online verfügbar:
  `https://vhome.offis.de/~eveith`

Geschichte

- Hadoop 0.1.0: April 2006
- Inspiration für Hadoop:
  - Das *Google-Dateisystem*
    Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003.
  - Der *Map-Reduce*-Algorithmus
    Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.
- Drei wesentliche Bestandteile:
  - HDFS: **Hadoop File System**
  - Hadoop **MapReduce**
  - YARN: **Yet Another Ressource Negotiator**
- („Hadoop" war der Name des Spielzeug-Elephanten des Sohnes eines der Entwickler)

Das *Google File System*

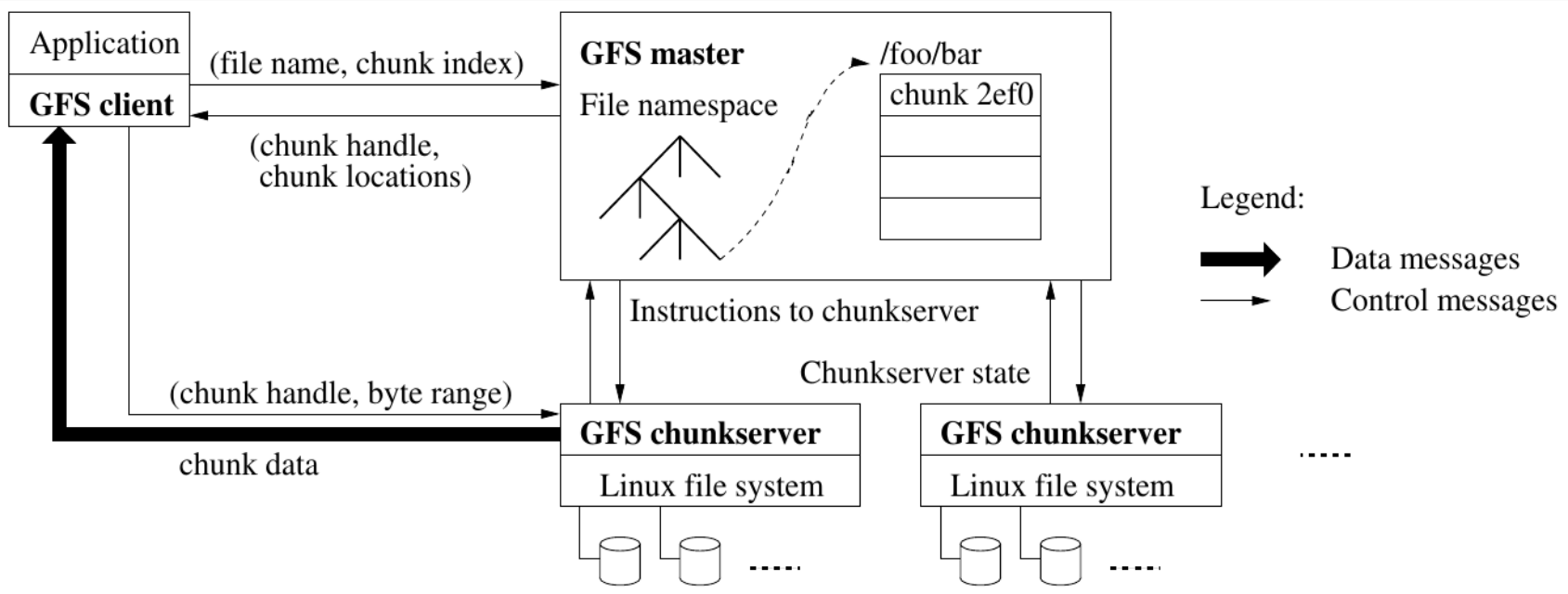


Ghemawat, Gobioff, & Leung. "The Google file system."

Annahmen für den Entwurf

- Schreiben: Hauptsächlich **Anhängen**, v.a. Streaming; gleichzeitiges Anhängen
- Lesen: Large streaming reads, small random reads
- Kleine Anzahl größere Dateien

Das *Google File System*

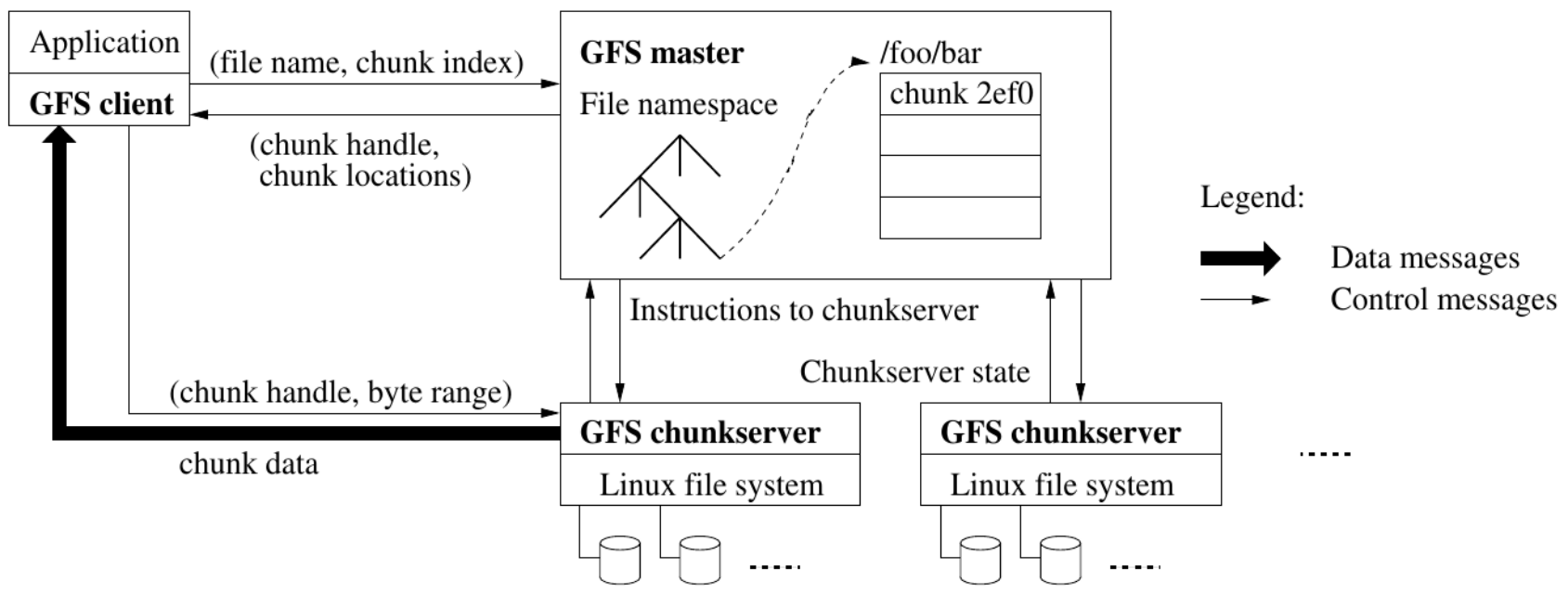


Ghemawat, Gobioff, & Leung. "The Google file system."

Ein *Master Server*, > 1 *Chunk Server*

Chunks: 64 MB

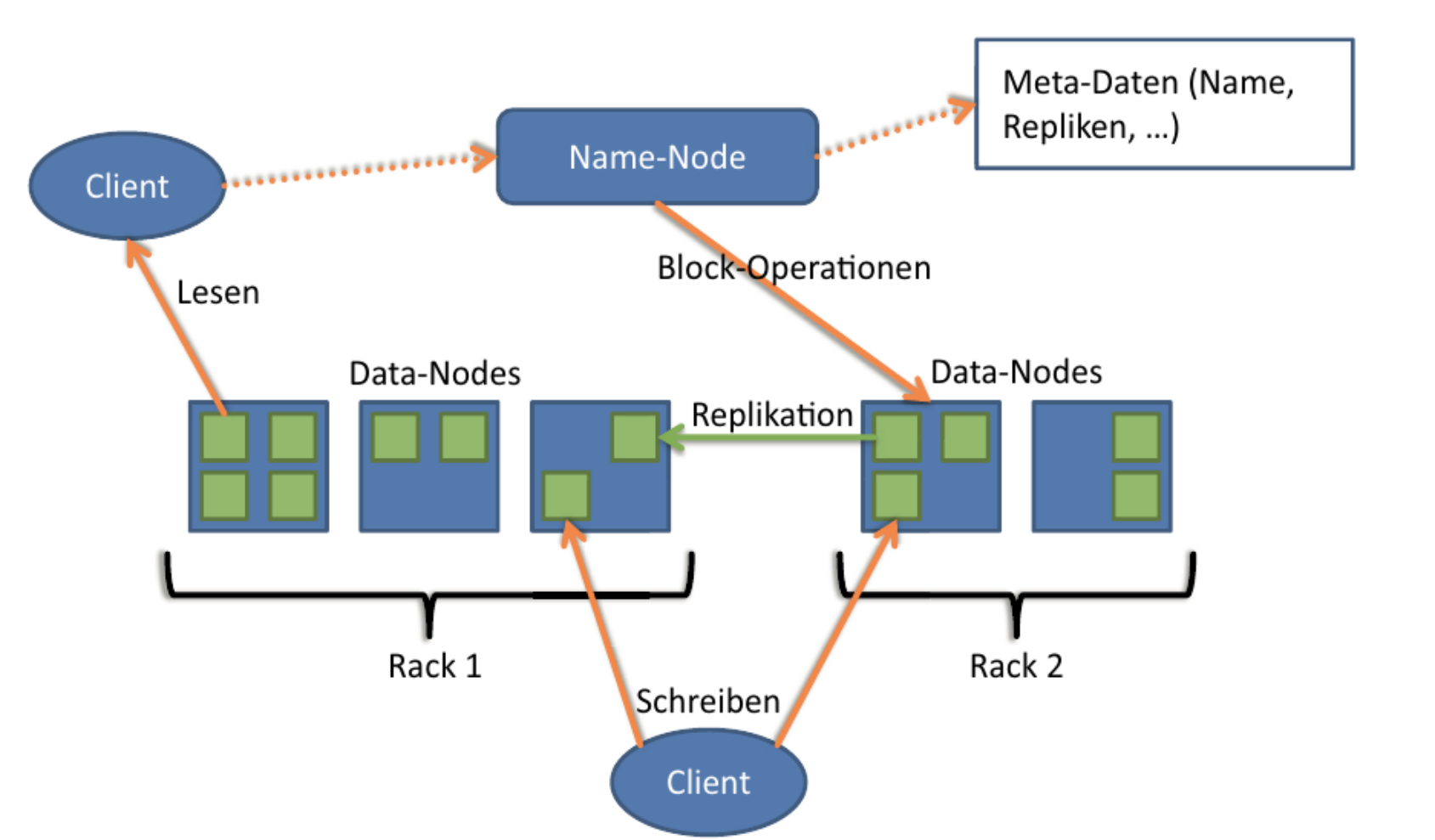
Das *Google File System*



Ghemawat, Gobioff, & Leung. "The Google file system."

- System aus handelsüblichen Geräten gebaut – Ausfälle vergleichsweise häufig
- *Fast Recovery* über *Operation Logs*
- Kein Unterschied zwischen normalem und abnormalem Prozessende

Hadoop File System



Freiknecht, & Papp. „Big Data in der Praxis.“

Umgang mit HDFS i

- Verzeichnis listen:

```
hdfs dfs -ls VERZEICHNIS
hdfs dfs -ls /
```

- Verzeichnis erstellen:

```
hdfs dfs -mkdir [-p] VERZEICHNIS
hdfs dfs -mkdir -p /input
```

- Dateien auf HDFS kopieren:

```
hdfs dfs -copyFromLocal QUELLE ZIEL
hdfs dfs -copyFromLocal ~/titanic.csv
↪  /input/titanic.csv
```

Umgang mit HDFS ii

- Dateien von HDFS kopieren:

```
hdfs dfs -copyToLocal QUELLE ZIEL
hdfs dfs -copyToLocal /output/survival.csv
↪  ~/survival.csv
```

- Kopieren innerhalb von HDFS:

```
hdfs dfs -cp QUELLE ZIEL
hdfs dfs -cp /input/titanic.csv /input/passengers.csv
```

- Verschieben/umbenennen innerhalb von HDFS:

```
hdfs dfs -mv QUELLE ZIEL
hdfs dfs -mv /input/titanic.csv /input/passengers.csv
```

- Löschen von Dateien und leeren Verzeichnissen, auch rekursiv:

```
hdfs dfs -rm [-r] ORDNER_ODER_DATEI
hdfs dfs -rm /input/titanic.csv
```

Umgang mit HDFS iii

- Erstellen einer leeren Datei:

  ```
  hdfs dfs -touchz DATEI
  hdfs dfs -touchz /hdfs/input/test3.txt
  ```

- Ausgeben des letzten Kilobytes einer Datei:

  ```
  hdfs dfs -tail [-f] DATEI
  hdfs dfs -tail -f /output/survival.csv
  ```

Alternative: Einhängen als reguläres Dateisystem

Map-Reduce

- Programmiermodel aus zwei Funktionen: `map(·)` und `reduce(·)`

$$\mathrm{map}(\mathit{KeyType}_1, \mathit{ValueType}_1) \mapsto \mathit{list}\,(\mathit{KeyType}_2, \mathit{ValueType}_2)$$

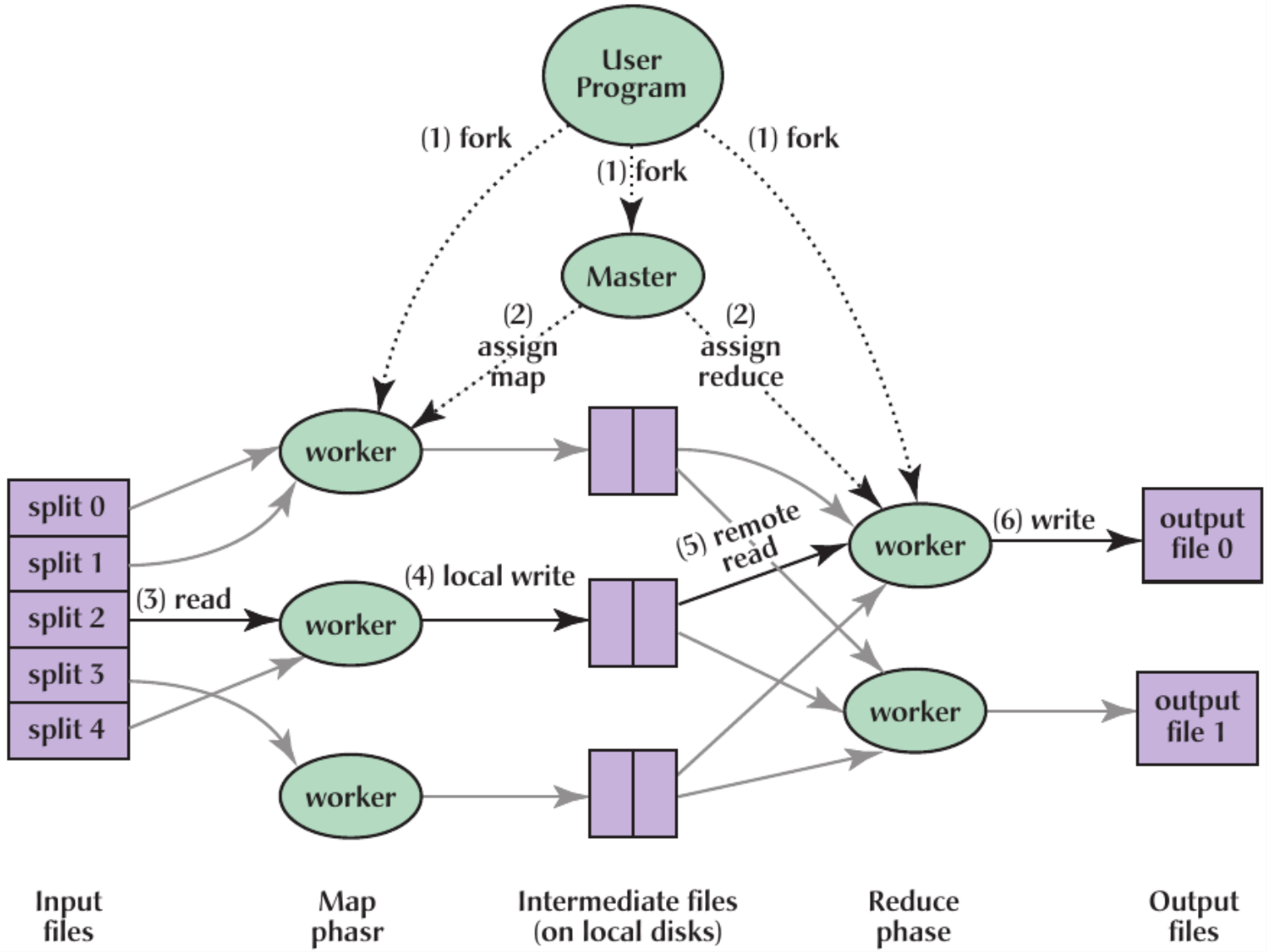
$$\mathrm{reduce}\,(\mathit{KeyType}_2, \mathit{list}(\mathit{ValueType}_2)) \mapsto \mathit{list}(\mathit{ValueType}_2)$$

- Paradigma erlaubt starke Parallelisierung
- Datenstruktur: *key*, *value* – keine andere möglich

Ablauf bei MapReduce

1. **Eingabedaten aufteilen:** *M Splits* von 16 MB to 64 MB
2. **Kopien starten:** So viele Kopien des Programms wie sinnvoll möglich auf den Clusterknoten starten
3. **Aufgaben verteilen:** Ein Master-Knoten verteilt *M Mapping*- und *R Reduce*-Jobs auf den Arbeitsknoten
4. **Map:** Jeder *Mapping*-Knoten liest einen *Split* und führt ihn der `map(·)`-Funktion des Nutzers zu. Die (*key*, *value*)-Paare werden im RAM gehalten
5. **Sichern:** Die Zwischenergebnisse werden periodisch auf die Festplatte geschrieben und in *R* Partitionen unterteilt
6. **Sortieren:** Ein *Reduce-Worker*-Knoten liest die Daten von der Festplatte eines *Mapping-Worker*-Knoten und sortiert sie nach dem Schlüssel (Werte gruppieren)
7. **Reduce:** Der Schlüssel und die zugehörigen Werte werden der `reduce(·)`-Funktion des Nutzers übergeben

MapReduce-Ausführungsschema



Dean, & Ghemawat. "MapReduce: simplified data processing on large clusters."

MapReduce zusammengefasst

Map

- Erzeugen von Schlüssel-Wert-Paaren

Combine

- Aggregieren der Schlüssel-Wert-Paare

Reduce

- Daten ausdünnen
- Ein Wert pro Schlüssel

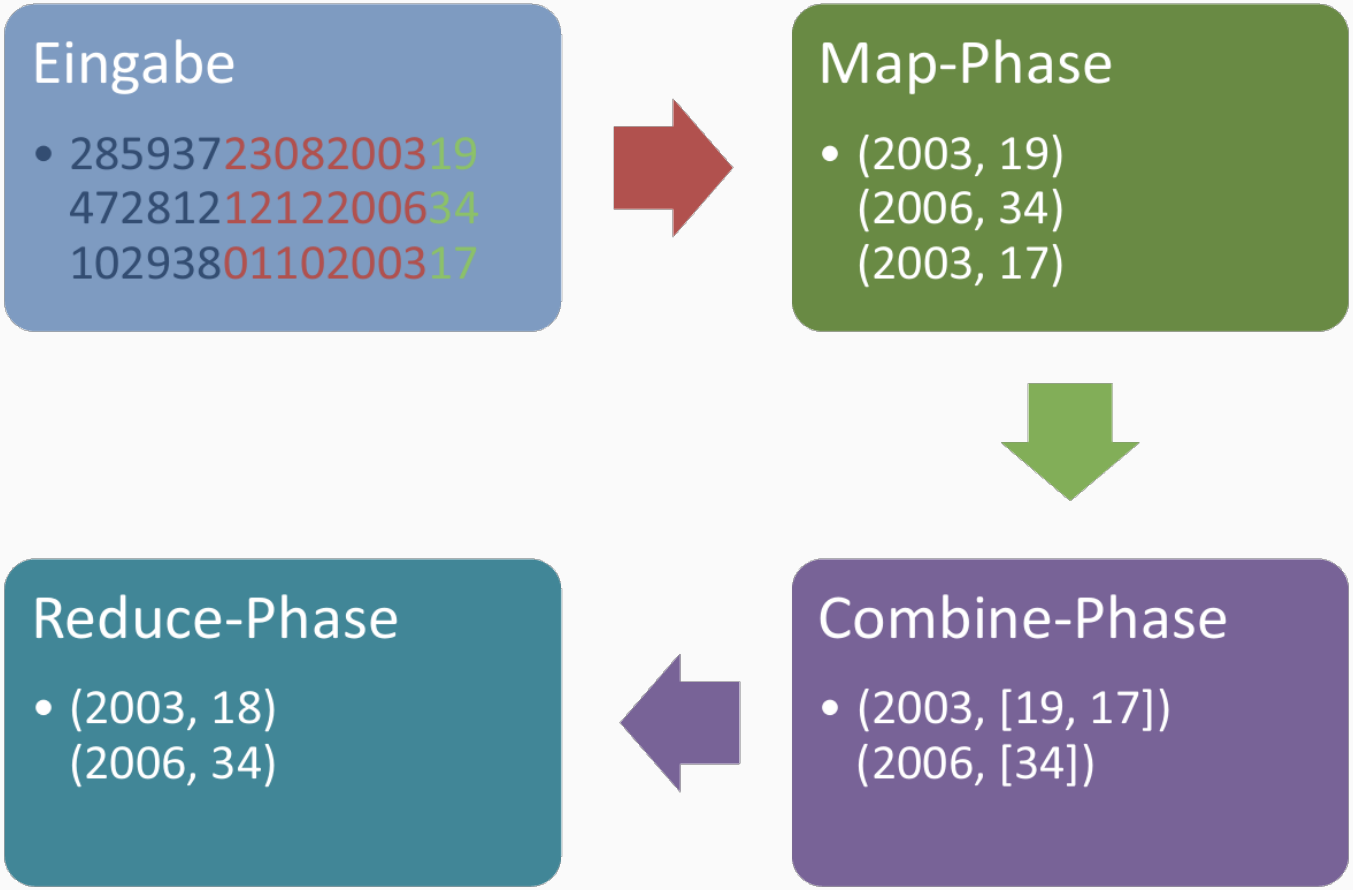
Beispiele für MapReduce

- **Verteiltes grep:** `map(·)` gibt die Zeilen mit Treffer aus
- **Zugriffe auf eine URL:** `map(·)` Durchforstet Regionen eines Logfiles/mehrere Logfiles und gibt $(\mathit{url}, 1)$ pro Zugriff aus; `reduce(·)` sortiert nach URL und summiert
- **Link-Graph:** Für eine Menge von Websites, emittiert `map(·)` $(\mathit{ziel}, \mathit{quelle})$; `reduce(·)` listet alle Quell-URLs zu einem Ziel: $(\mathit{ziel}, \mathit{list}(\mathit{quelle}))$

Code-Beispiel für MapReduce

- Aufgabenstellung: Berechnung des Notendurchschnitts der Studierenden der letzten zehn Jahre
- Datenquelle: Hochschulserver
- Datenformat: Semi-Strukturierte Abfolge von Matrikelnummer, Datum und Note:
  `2859372308200319472812121220063 4...`
  `|Mtr |Datum |N|Mtr |Datum |N|...`

Ablauf des Beispiels



Freiknecht, & Papp. „Big Data in der Praxis.“

MapReduce mit Hadoop: Mapper i

```
package de.jofre.grades;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

// Eingabe-Key, Eingabe-Wert, Ausgabe-Key, Ausgabe-Wert
public class GradesMapper extends
 ↪  Mapper<Text,Text,IntWritable,IntWritable> {

    private IntWritable year_int = null;
    private IntWritable grade_int = null;
```

MapReduce mit Hadoop: Mapper ii

```
    public void map(Text key, Text value, Context context) throws
     ↪  IOException, InterruptedException {
        // Auslesen des Jahres und der Note aus einem String wie
         ↪  "2853972308201319"
        String year_str = value.toString().substring(10,13);
        year_int = new IntWritable(Integer.parseInt(year_str));
        String grade_str = value.toString().substring(14,15);
        grade_int = new IntWritable(Integer.parseInt(grade_str));

        context.write(year_int, grade_int);
    }
}
```

MapReduce mit Hadoop: Reducer i

```
package de.jofre.grades;

import java.io.IOException;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

// Eingabe-Key, Eingabe-Wert, Ausgabe-Key, Ausgabe-Wert
public class GradesReducer extends Reducer<IntWritable,
 ↪  IntWritable, IntWritable, FloatWritable> {

    @Override
    protected void reduce(IntWritable key, Iterable<IntWritable>
     ↪  values, Context context) throws IOException,
     ↪  InterruptedException {
```

MapReduce mit Hadoop: Reducer ii

```
        // Summiere alle Noten eines Jahres auf...
        float sum = 0;
        int count = 0;

        for (IntWritable val : values) {
            sum +=val.get();
            count +=1;
        }

        // Arithmetisches Mittel bilden:
        float result = sum / count;

        // Schreibe den Durchschnitt für das Jahr in key
        context.write(key, new FloatWritable(result));
    }
}
```

MapReduce mit Hadoop: Driver i

```
package de.jofre.grades;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
 ↪  org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

MapReduce mit Hadoop: Driver ii

```
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;


public class GradesDriver extends Configured implements Tool {
    private final static Logger log =
     ↪  Logger.getLogger(GradesDriver.class.getName());

    public static void main(String[] args) {
        int res = 1; // Wenn 1 nicht verändert wird, endet der
         ↪  Job nicht korrekt
        try {
            res = ToolRunner.run(new Configuration(), new
             ↪  GradesDriver(), args);
        } catch (Exception e) {
            log.log(Level.SEVERE, "Fehler beim Ausführen des
             ↪  Jobs!");
```

MapReduce mit Hadoop: Driver iii

```
            e.printStackTrace();
        }
        System.exit(res);
    }

    @Override
    public int run(String[] args) {
        log.log(Level.INFO, "Starte Map Reduce-Job
        ↪  'GradesDriver'... ");

        // Wenn Configured erweitert wird, kann die bestehende
        ↪  Konfiguration
        // per getConf abgerufen werden.
        Configuration conf = this.getConf();
        Job job = null;
        try {
```

MapReduce mit Hadoop: Driver iv

```
        job = Job.getInstance(conf);
    } catch (IOException e1) {
        log.log(Level.SEVERE, "Fehler bei Instanziierung des
         ↪  Jobs!");
        e1.printStackTrace();
    }

    // Hadoop soll ein verfügbares JAR verwenden, das die
     ↪  Klasse
    // GradesDriver enthält.
    job.setJarByClass(GradesDriver.class);

    // Mapper- und Reducer-Klasse werden festgelegt
    job.setMapperClass(GradesMapper.class);
    job.setReducerClass(GradesReducer.class);
```

MapReduce mit Hadoop: Driver v

```
// Ausgabetypen werden festgelegt
job.setOutputKeyClass(IntWritable.class);
job.setOutputValueClass(FloatWritable.class);
job.setMapOutputKeyClass(IntWritable.class);
job.setMapOutputValueClass(IntWritable.class);
job.setInputFormatClass(KeyValueTextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

// Der Pfad, aus dem Hadoop die Eingabedateien liest,
 ↪  wird als erstes Argument
// beim Starten des JAR übergeben.
try {
    FileInputFormat.addInputPath(job, new Path(args[0]));
} catch (IllegalArgumentException e) {
    log.log(Level.SEVERE, "Fehler (Argument) beim Setzen
     ↪  des Eingabepfades!");
```

MapReduce mit Hadoop: Driver vi

```
    e.printStackTrace();
} catch (IOException e) {
    log.log(Level.SEVERE, "Fehler (IO) beim Setzen des
     ↪  Eingabepfades!");
    e.printStackTrace();
}

// Der Ausgabeordner wird als zweites Argument übergeben
FileOutputFormat.setOutputPath(job, new Path(args[1]));
boolean result = false;
try {
  // Führe den Job aus und warte, bis er beendet wurde
  result = job.waitForCompletion(true);
} catch (ClassNotFoundException e) {
    log.log(Level.SEVERE, "Fehler (ClassNotFound) beim
     ↪  Ausführen des Jobs!");
```

MapReduce mit Hadoop: Driver vii

```
            e.printStackTrace();
        } catch (IOException e) {
            log.log(Level.SEVERE, "Fehler (IOException) Ausführen
             ↪  des Jobs!");
            e.printStackTrace();
        } catch (InterruptedException e) {
            log.log(Level.SEVERE, "Fehler (Interrupted) beim
             ↪  Ausführen des Jobs!");
            e.printStackTrace();
        }
        log.log(Level.INFO, "Fertig!");
        return result ? 0 : 1;
    }
}
```

Ausführen

```
hadoop jar 02_MapReduceStudentData.jar /hdfs/mr1/input
 ↪  /hdfs/mr1/output
```

MapReduce-Paradigma – Rigide Ausführungsweise

MapReduce diktiert ein Programmier-/Ausführungsparadigma

- Gerichteter, azyklischer Graph
- Nur Schlüssel-Wert-Paare
- Feste Eingabedatentypen pro Schritt
- Kein(e)...
  - Mehrfaches Iterieren
  - Wenn-dann-Sonst-Verzweigungen

Lösungsansatz: Verketten von *Map-Reduce*-Jobs

Verketten: Ein Beispiel

Schritt 1:
Durchschnitt
der Studierenden
berechnen

Freiknecht	1,6
Daniela Meier	1,3
Julian Müller	2,0
Daniela Meier	1,5

Jonas Freiknecht	1,6
Daniela Meier	1,4
Julian Müller	2,0

Schritt 2:
Geschlechterspezifischen
Durchschnitt berechnen

Jonas Freiknecht	1,6
Daniela Meier	1,4
Julian Müller	2,0

Männlich	1,8
Weiblich	1,4

Freiknecht, & Papp. „Big Data in der Praxis.“

Verketten von MapReduce-Jobs i

```
public class AverageGradeDriver extends Configured implements
 ↪  Tool {
    private final static Logger log =
     ↪  Logger.getLogger(AverageGradeDriver.class.getName());

    public static void main(String[] args) {
        int res = 1; // Wenn 1 nicht verändert wird, endet der
         ↪  Job nicht korrekt
        try {
            res = ToolRunner.run(new Configuration(), new
             ↪  AverageGradeDriver(), args);
        } catch (Exception e) {
            log.log(Level.SEVERE, "Fehler beim Ausführen des
             ↪  Jobs!");
            e.printStackTrace();
```

Verketten von MapReduce-Jobs ii

```
        }
        System.exit(res);
    }

    @Override
    public int run(String[] args) {
        log.log(Level.INFO, "Starte Map Reduce-Job
        ↪  'GradesDriver'... ");

        // Wenn Configured erweitert wird, kann die bestehende
        ↪  Konfiguration
        // per getConf abgerufen werden.
        Configuration conf = this.getConf();
        Job job = null;
        try {
            job = Job.getInstance(conf);
```

Verketten von MapReduce-Jobs iii

```
} catch (IOException e1) {
    log.log(Level.SEVERE, "Fehler bei Instanziierung des
     ↪  Jobs!");
    e1.printStackTrace();
}

// Hadoop soll ein verfügbares JAR verwenden, das die
 ↪  Klasse
// GradesDriver enthält.
job.setJarByClass(AverageGradeDriver.class);

// Mapper- und Reducer-Klasse werden festgelegt
job.setMapperClass(AverageGradeMapper.class);
job.setReducerClass(AverageGradeReducer.class);

// Ausgabetypen werden festgelegt
```

Verketten von MapReduce-Jobs iv

```
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(FloatWritable.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(FloatWritable.class);
job.setInputFormatClass(KeyValueTextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

// Der Pfad, aus dem Hadoop die Eingabedateien liest,
 ↪  wird als erstes Argument
// beim Starten des JAR übergeben.
try {
    FileInputFormat.addInputPath(job, new Path(args[0]));
} catch (Exception e) {
    log.log(Level.SEVERE, "Fehler beim Setzen des
     ↪  Eingabepfades!");
    e.printStackTrace();
```

Verketten von MapReduce-Jobs v

```
}

// Der Ausgabeordner wird als zweites Argument übergeben
FileOutputFormat.setOutputPath(job, new Path(args[1]));
boolean result = false;

try {
// Führe den Job aus und warte, bis er beendet wurde
    result = job.waitForCompletion(true);
} catch (Exception e) {
    log.log(Level.SEVERE, - 28492581 "Fehler beim -
     ↪  transid Ausführen des - 28492581_1D Jobs!");
    e.printStackTrace();
}

log.log(Level.INFO, "Fertig!");
```

Verketten von MapReduce-Jobs vi

```
        return result ? 0 : 1;
    }
}
```

Einlesen der Durchschnittsnoten i

```
// Eingabe-Key, Eingabe-Wert, Ausgabe-Key, Ausgabe-Wert
public class AverageGradeMapper extends
 ↪  Mapper<Text,Text,Text,FloatWritable> {
    public void map(Text key, Text value, Context context) throws
     ↪  IOException, InterruptedException {
        // Formatieren der Gleitkommazahlen (Ersetzen der Kommata
         ↪  durch Punkte)
        String pointFloat = value.toString().replace(',','.');
        FloatWritable floatValue = new
         ↪  FloatWritable(Float.parseFloat(pointFloat));

        // Hier müssen wir einfach nur die vorhandenen Daten in
         ↪  den Mapper einlesen
        context.write(key, floatValue);
```

Einlesen der Durchschnittsnoten ii

```
        }
    }
```

Trenner konfiguriere

- Ausgabe des Mappers z.B.

  `Jones Freiknecht 1.9`

- Wie Name und Note trennen?
- Trennzeichen festlegen:

```
conf.set(
    "mapreduce.input." +
        "keyvaluelinerecordreader.key.value.separator",
    "\t");
```

Reducer: Durchschnittsnoten bilden i

```
// Eingabe-Key, Eingabe-Wert, Ausgabe-Key, Ausgabe-Wert
public class AverageGradeReducer extends Reducer<Text,
 ↪  FloatWritable, Text, FloatWritable> {
    private final static Logger log =
     ↪  Logger.getLogger(AverageGradeReducer.class.getName());


    @Override
    protected void reduce(Text key, Iterable<FloatWritable>
     ↪  values, Context context) throws IOException,
     ↪  InterruptedException {
        // Summiere alle Noten eines Studierenden auf...
        float sum = 0;
        float count = 0;
        for (FloatWritable val : values) {
```

Reducer: Durchschnittsnoten bilden ii

```
            sum += val.get();
            count +=1;
        }

        // Und bilde den Durchschnitt
        float result = sum / count;
        log.log(Level.INFO, "Name: "+key+" Note: "+result);

        // Schreibe den Durchschnitt für den Studierenden in key
        context.write(key, new FloatWritable(result));
    }
}
```

Job Starter i

```
public boolean startJob(JspWriter writer) {
    boolean result = false;

    Job job = null;
    try {
        job = Job.getInstance(conf);
    } catch (IOException e1) {
        log.log(Level.SEVERE, "Fehler beim Setzen der
        ↪  Job-Config.");
        e1.printStackTrace();
    }

    job.setJarByClass(GradesDriver.class);

    // Mapper- und Reducer-Klasse festlegen
```

Job Starter ii

```
job.setMapperClass(GradesMapper.class);
job.setReducerClass(GradesReducer.class);

// Ausgabetypen festlegen
job.setOutputKeyClass(IntWritable.class);
job.setOutputValueClass(FloatWritable.class);
job.setMapOutputKeyClass(IntWritable.class);
job.setMapOutputValueClass(IntWritable.class);
job.setInputFormatClass(KeyValueTextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

// Den Input-Pfad setzen wir diesmal im Code
try {
    FileInputFormat.addInputPath(job,
            new Path(HadoopProperties.get("hdfs_address")
                    + "/hdfs/mr1/input"));
```

Job Starter iii

```
} catch (IOException e) {
    log.log(Level.SEVERE, "Fehler beim Setzen des
     ↪  Eingabepfades!");
    e.printStackTrace();
}

FileOutputFormat.setOutputPath(job,
        new Path(HadoopProperties.get("hdfs_address")
                + "/hdfs/mr2/output"));

// Dank "waitForCompletion" können wir linear nach Job 1 den
 ↪  zweiten
// starten:
try {
    result = job.waitForCompletion(true);
} catch (Exception e) {
```

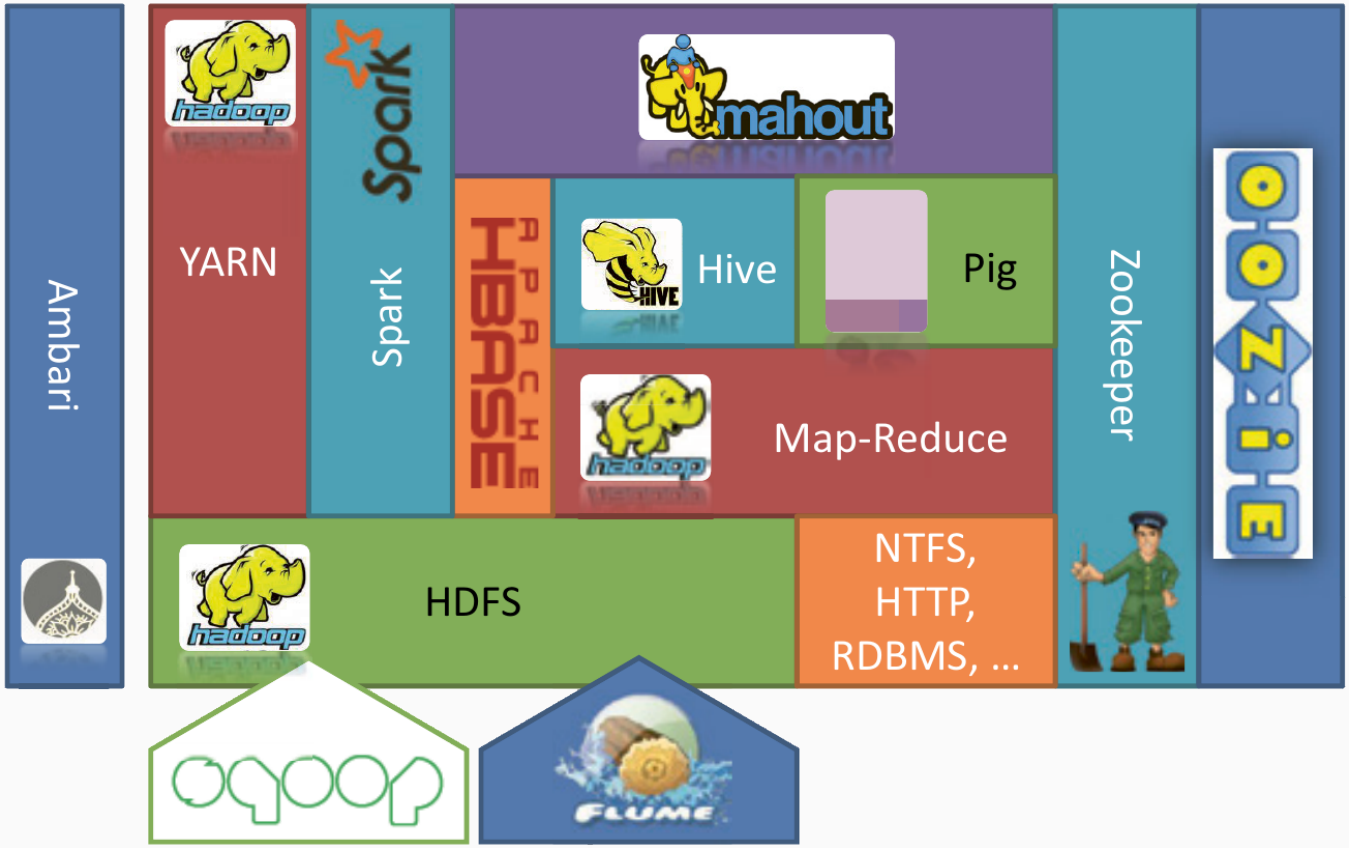
Job Starter iv

```
            log.log(Level.SEVERE, "Fehler beim Ausführen des Jobs!");
            e.printStackTrace();
        }

        // ... job 2 startet hier ...

        return result;
    }
```

Das Hadoop-Ökosystem



Freiknecht, & Papp. „Big Data in der Praxis.“