

Middleware - Cloud Computing – Übung

Tobias Distler, Klaus Stengel, Timo Hönig, Johannes Behl

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

www4.informatik.uni-erlangen.de

Wintersemester 2011/12



Programmieren mit Hadoop

Einführung Apache Hadoop

Ein- und Ausgabe, Serialisierung

Auftragsverwaltung

JAR-Dateien erstellen

Verteilte Ausführung

Dateisysteme in Hadoop

Aufgabe 4

Visualisierung

Hadoop-Instanzen in der Cloud



- Freie Implementierung eines MapReduce-Frameworks
- Besteht aus mehreren Teilprojekten
 - **Common**: Gemeinsame Infrastruktur
 - **HDFS**: Verteiltes Dateisystem
 - **MapReduce**: MapReduce-Framework
- Implementierung in Java
- Bindings zu anderen Sprachen verfügbar (JNI)



- `org.apache.hadoop.conf`
Verwaltung der Konfiguration
- `org.apache.hadoop.fs`
Dateisystem-Abstraktion
- `org.apache.hadoop.io`
Serialisierung und Deserialisierung von Daten
- `org.apache.hadoop.mapreduce`
MapReduce-Framework
- `org.apache.hadoop.util`
Verschiedene Hilfsklassen



- Objekte zur Bearbeitung durch MapReduce müssen in Dateien gespeichert werden können
- Zur Speicherung von Objekten muss bekannt sein, **welche** Daten **wie** abgelegt werden müssen
- Schnittstelle `WritableComparable` beschreibt Klassen, die
 - **serialisierbar** und **deserialisierbar** sind
 - untereinander **vergleichbar** sind
- Hadoop liefert Implementierung für elementare Datentypen mit:
 - `int` → `IntWritable`
 - `long` → `LongWritable`
 - ...
 - Platzhalter: `NullWritable`



- **Serialisierung:** Schreibt alle zur Rekonstruktion des Objekts in den Datenstrom `out`

```
public void write(DataOutput out)
```

- **Deserialisierung:** Rekonstruiert Zustand des Objekts aus Datenstrom `in`

```
public void readFields(DataInput in)
```

- **Vergleichs-Operator:** Vergleicht `this` mit `x` (siehe Java-Interface Comparable)

```
public int compareTo(WritableComparable x)
```

- **Hashing:** Ermittelt Hash-Wert über Daten

```
public int hashCode()
```



- Beispiel:

```
public class MyData implements WritableComparable {  
    protected int node;  
    protected long weight;  
  
    public void write(DataOutput out)  
        throws IOException {  
        out.writeInt(node);  
        out.writeLong(weight);  
    }  
    public void readFields(DataInput in)  
        throws IOException {  
        node = in.readInt();  
        weight = in.readLong();  
    }  
    public int compareTo(MyData x) {  
        return (weight < x.weight) ? -1 :  
            ((weight == x.weight) ? 0 : 1);  
    }  
    [...]
```



- Standard-`hashCode()`-Implementierung garantiert **keine** gleichen Hash-Werte bei verschiedenen Objekten mit gleichem **Inhalt**
- Wird jedoch von `HashPartitioner` zum Partitionieren nach Schlüssel für die Reduce-Phase verwendet
- → Eigene Implementierung basierend auf Inhalt notwendig
- Beispiel (Fortsetzung):

```
[...]
public boolean equals(Object x) {
    if (!(o instanceof MyData)) return false;
    MyData z = (MyData)x;
    return (z.node == node) && (z.weight == weight);
}

public int hashCode() {
    return (node * 7) + (int)(weight * 13);
}
```



- Einlesen der Daten über InputFormat
- Ausgabe über OutputFormat
- Hadoop enthält bereits Klassen für verschiedene Ein-/Ausgabeformate:
 - TextInputFormat: Einlesen aus Textdateien
 - SequenceFileInputFormat: Hadoop-internes Binärformat
 - ...OutputFormat entsprechend
- Für Übungsaufgabe vorgegeben: Lesen und schreiben von Binärdaten mit konstanter Satzlänge
 - `mw.cliquefind.io.RecordInputFormat`
 - `mw.cliquefind.io.RecordOutputFormat`
- InputFormat benötigt je abgeleiteten Typ eigene Spezialisierung, da beim Einlesen neue Instanzen erzeugt werden (via Reflection)



- Klasse `org.apache.hadoop.mapreduce.Job`
- Speichert Informationen zum Ablauf eines Jobs
 - Zur Ausführung benötigte Klassen (Mapper, Reducer...)
 - Ein- und Ausgabedateien
 - Jobspezifische Konfiguration
 - Ausführungszustand
- Änderbare Einstellungen:
 - Job-Bezeichnung

```
void setJobName(String name)
```
 - .jar-Datei mit benötigten Klassen

```
void setJarByClass(Class cls)
```



- Änderbare Einstellungen (Fortsetzung):
 - Klasse zur Verarbeitung des Eingabeformats

```
void setInputFormatClass(Class cls)
```
 - Klasse zum Schreiben der Ausgabedaten

```
void setOutputFormatClass(Class cls)
```
 - Mapper-Klasse und deren Key/Value-Ausgabeformate

```
void setMapperClass(Class cls)
void setMapOutputKeyClass(Class cls)
void setMapOutputValueClass(Class cls)
```
 - Reducer-Klasse und deren Key/Value-Ausgabeformate

```
void setReducerClass(Class cls)
void setOutputKeyClass(Class cls)
void setOutputValueClass(Class cls)
```



- Beispiel:

```
class MyJob extends Job {  
    [...]  
    public MyJob(Configuration cfg, String name) {  
        super(cfg, name);  
  
        setJarByClass(getClass());  
  
        setInputFormatClass(EdgeInputFormat.class);  
        setOutputFormatClass(RecordOutputFormat.class);  
  
        setMapperClass(Map.class);  
        setMapOutputKeyClass(IntWritable.class);  
        setMapOutputValueClass(LongWritable.class);  
  
        setReducerClass(Reduce.class);  
        setOutputKeyClass(NullWritable.class);  
        setOutputValueClass(IntWritable.class);  
    }  
}
```



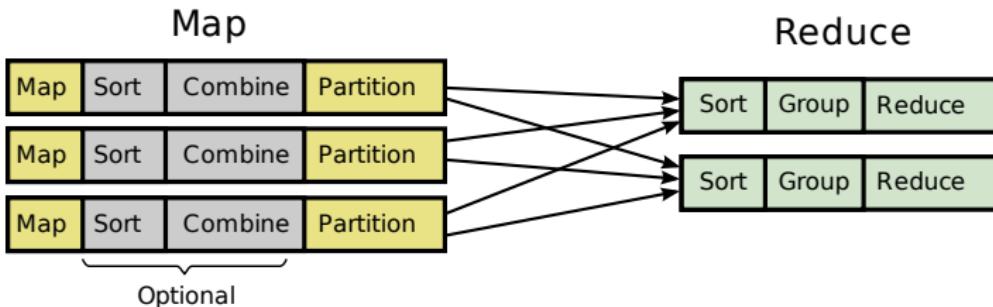
- Beispiel (Fortsetzung):

```
// class MyJob extends Job...
public static class Map
    extends Mapper<NullWritable, Edge,
                IntWritable, LongWritable> {
    @Override
    public void map(NullWritable k, Edge v, Context ctx) {
        [...]
    }
}
public static class Reduce
    extends Reducer<IntWritable, LongWritable,
                  NullWritable, IntWritable> {
    @Override
    public void reduce(IntWritable k,
                      Iterable<LongWritable> v, Context ctx) {
        [...]
    }
}
```



Sortierung, Gruppierung und Partitionierung

- **Sortierung:** Reihenfolge der Schlüssel/Wert-Paare
- **Gruppierung:** Schlüssel, die beim Reduce zusammengefasst werden
- **Partitionierung:** Zuordnung der Schlüssel zu Reduce-Instanzen mittels Hash-Funktion
- Implementierung in Hadoop:



- Festlegen von Combiner, Sorting, Grouping und Partitioner
- Beispiel:

```
// Partitioner
setPartitionerClass(Partition.class);

// Grouping
setGroupingComparatorClass(Grouping.class);

// Sorting
setSortComparatorClass(Sorting.class);

// Combiner
setCombinerClass(Combine.class);
```



- Konfigurationseinstellungen für Job abfragen/ändern

```
Configuration getConfiguration()
```

- Manipulation von Zählern:

```
Counters getCounters()
```

- Festlegen der Ein- und Ausgabedateien für Job:

```
static FileInputFormat.setInputPaths(Job job, Path path);
```

```
static FileOutputFormat.setOutputPath(Job job, Path path);
```

- Erfolgt indirekt über statische Methoden der verwendeten Klasse zur Formatierung der Ein- bzw. Ausgabe!



- Starten des Jobs (asynchron):

```
void submit()
```

- Nach Start können Job-Parameter nicht mehr verändert werden!
- Statusabfrage:

```
boolean isComplete()
```

```
boolean isSuccessful()
```

- Warten auf Ende des Jobs:

```
boolean waitForCompletion(boolean verbose)
```



- Java-Klasse `org.apache.hadoop.util.Tool`
 - Interface mit Einstiegspunkt für eigene Hadoop-Anwendungen

```
int run(String[] args)
```

- Java-Klasse `org.apache.hadoop.util.ToolRunner`
 - Initialisierung des Frameworks
 - Verarbeitung der Kommandozeile
 - Starten des angegebenen Tools

```
static int run(Configuration cfg, Tool t, String[] args)
```



- Beispiel:

```
public class MyApp extends Configured implements Tool {  
    public int run(String[] args) throws Exception {  
        System.out.println("Hello " + args[0] + "!");  
        System.out.println(getConf().get("mycfg.setting"));  
        return 0;  
    }  
  
    public static void main(String[] args) throws Exception {  
        int res = ToolRunner.run(new Configuration(),  
                               new MyApp(), args);  
        System.exit(res);  
    }  
}
```

- Aufruf:

```
$ hadoop jar myapp.jar -D mycfg.setting=hi World  
Hello World!  
hi
```



- Hadoop-Programme müssen als .jar-Dateien vorliegen
- Aufruf:

```
$ hadoop jar myprogram.jar <parameter>
```

- .jar-Dateien werden im ZIP-Format gespeichert und enthalten
 - Java-Klassendateien (.class)
 - Meta-Informationen (META-INF/MANIFEST.MF)
- Manifest-Datei ermöglicht
 - Definieren von Abhängigkeiten und Classpath
 - Versionierung
 - Festlegen der zu verwendenden `main()`-Methode



- Beispiel für Manifest:

```
Manifest-Version: 1.0
Main-Class: mw.MyApp
Class-Path: auxlib.jar
```

- Direktes starten der .jar-Datei (Main-Class muss definiert sein):

```
$ java -jar myapp.jar
```

- Erstellen eines JARs auf der Kommandozeile (siehe `man jar`):

```
$ jar cfe myapp.jar mw.MyApp mw/
```



- ant ist ein Programm zum automatischen Bauen von Programmen (vgl. make)
- Zu erstellenden Objekte mittels XML-Datei (`build.xml`) definiert
- Enthält Modul zum bauen von .jar-Dateien:

```
<?xml version="1.0" encoding="UTF-8" ?>
<project default="makejar" name="jar example">
    <target name="makejar">
        <jar destfile="myapp.jar">
            <manifest>
                <attribute name="Main-Class" value="mw.MyApp"/>
            </manifest>
            <fileset dir="bin/">
                <include name="mw/**"/>
            </fileset>
        </jar>
    </target>
</project>
```

- Erzeugen der .jar-Datei durch Aufruf von ant



- Hadoop unterstützt zwei verschiedene Ausführungsmodi
 - Lokal
 - Verteilt
- Standardmäßig lokale, sofortige Ausführung
- Verteilte Ausführung erfordert
 - Dienste für Job-Verteilung und Ausführung
 - Hadoop Filesystem (HDFS)
 - Konfiguration (/etc/hadoop/conf)



- **Jobtracker**
 - Verwaltet und verteilt MapReduce-Jobs
 - Nur eine zentrale Instanz
- **Tasktracker**
 - Zuständig für die Ausführung von Jobtracker zugewiesener Jobs
 - Läuft auf jedem Knoten, der Jobs bearbeiten soll
- **Namenode**
 - Verwaltet Dateien und deren Speicherorte im verteilten Dateisystem HDFS
 - Eine zentrale, primäre Instanz
 - Sekundärer Namenode ermöglicht schnellere Wiederherstellung
- **Datanode**
 - Zuständig für die Speicherung von Datenblöcken des HDFS
 - Läuft auf jedem Knoten, der Speicherplatz zur Verfügung stellen soll



- Kommunikation der Dienste untereinander erfolgt per HTTP
- Web-Server liefert HTML-Seiten zur Statusabfrage
 - Status Jobtracker:
`http://hostname:50030/`
 - Status Namenode/HDFS:
`http://hostname:50070/`
- Zugriff kann derzeit nicht beschränkt werden
- Betrieb daher **nur** in privatem Netz oder hinter entsprechender Firewall



- Setzen von Konfigurationsparametern
 - Auf der Kommandozeile:

```
$ hadoop <command> -D dfs.data.dir=/mnt/hadoop
```

- XML-Dateien:

```
<configuration>
  <property>
    <name>dfs.data.dir</name>
    <value>/mnt/hadoop</value>
  </property>
</configuration>
```

- Verfügbare Konfigurationsdateien
 - `core-site.xml`: Generelle Einstellungen
 - `hdfs-site.xml`: HDFS-Spezifische Einstellungen
 - `mapred-site.xml`: MapReduce Einstellungen



- Typische Konfiguration:
 - Ein Knoten für Jobtracker und Namenode
 - Restliche Knoten: Gleichzeitig Tasktracker und Datanode
 - Erlaubt Nutzung lokaler Datenspeicherung
- Namenode (`core-site.xml`)

```
<name>fs.default.name</name> <value>hdfs://131.188.42.111/</value>
```

- Jobtracker (`mapred-site.xml`)

```
<name>mapreduce.jobtracker.address</name> <value>131.188.42.111</value>
```

- Liste von Tasktracker/Datanode-Adressen in `conf/slaves` (kein XML!)

```
131.188.42.112  
131.188.42.113  
[...]
```



- `mapred.tasktracker.map.tasks.maximum`,
`mapred.tasktracker.reduce.tasks.maximum`:
Maximale Anzahl parallel ablaufender Map/Reduce-Tasks **je Knoten**
- `mapred.map.tasks`, `mapred.reduce.tasks`:
Hinweis an Partitionierer über die gewünschte Zahl von Teilen für Mapper/Reducer
- `mapred.min.split.size`, `mapred.max.split.size`:
Grenzen für Partitionierer bezüglich Größe der pro Mapper/Reducer verarbeiteten Teile



- Java-API: `org.apache.hadoop.conf.Configuration`
- Einstellungen können zur Laufzeit abgefragt und verändert werden:

```
public void changeBlocksize(Configuration mycfg) {  
    long sz = mycfg.getLong("mapred.min.split.size");  
    mycfg.setLong("mapred.min.split.size", sz * 10);  
}
```

- Beliebige eigene Definitionen möglich:

```
Configuration mycfg = new Configuration(getConf());  
int ni = mycfg.getInt("mytool.num_iterations", 100);  
  
$ hadoop jar mytool.jar -D mytool.num_iterations=500
```



- Hadoop abstrahiert über verschiedene „Dateisysteme“
- URI-Schema zur Unterscheidung
- Unterstützt werden unter anderem:
 - Lokales Dateisystem (`file://...`)
 - HDFS (`hdfs://...`)
 - Amazon S3 / Walrus (`s3://...`, `s3n://...`)
 - HTTP (`http://...`)
- Zugriff über einheitliche Java-API
- Ein- und Ausgaben von MapReduce-Jobs können beliebige Dateisysteme direkt verwenden



- Repräsentation von Pfaden mittels Path
- Operationen auf Objekte im Dateisystem mittels FileSystem
- Komplexere Operationen über statische Methoden in FileUtil
- Beispiel: Löschen einer Datei

```
public void delete(Configuration cfg, String name) {  
    try {  
        Path p = new Path(name);  
        FileSystem fs = p.getFileSystem(cfg);  
  
        fs.delete(p, true); //recursive delete  
  
    } catch(Exception e) {  
        System.err.println("Delete failed for " + name);  
    }  
}
```



- Hadoop bringt Werkzeug zum Zugriff auf Dateisysteme mit

- Aufruf mit:

```
$ hadoop fs <command> ...
```

- Mögliche Unterbefehle:

- **-ls <path>**: Auflisten von Dateien
- **-cp <src> <dst>**: Kopieren von Dateien
- **-getmerge <src> <dest>**: Zusammenfügen und ins lokale Dateisystem kopieren
- **-cat <src> <...>**: Zusammenfügen und auf Standardausgabe ausgeben
- **-rmr <path>**: Rekursives löschen

- Siehe **Hadoop FS Shell Guide**



Programmieren mit Hadoop

Einführung Apache Hadoop

Ein- und Ausgabe, Serialisierung

Auftragsverwaltung

JAR-Dateien erstellen

Verteilte Ausführung

Dateisysteme in Hadoop

Aufgabe 4

Visualisierung

Hadoop-Instanzen in der Cloud



- Zur Visualisierung ist das Program Cytoscape im CIP installiert
- Vorgegebenes Start-Skript lädt Daten mittels bereitgestelltem Import-Plugin
- Lediglich Kanten-Dateien unterstützt (Binär, ID-Paare verbundener Knoten)
- Beispiel:

```
$ cd /proj/i4mw/pub/aufgabe4/tools
$ ./visualize.sh ~/output.bin
```



- Gestartete Hadoop-Dienste sind öffentlich erreichbar
 - Einschränkung der Zugriffe notwendig
 - Integrierte Rechteverwaltung unsicher
 - Bietet keine echte Authentifizierung
- Eucalyptus-Abbild und Skripte zum Starten der Dienste vorgegeben
 - Konfigurationsdateien werden automatisch verteilt
 - Beschränkt Zugriffe auf Rechner, welcher Hadoop-Cluster gestartet hat
- Unterschiedliche SSH-Schlüssel trotz gemeinsamen Abbilds
 - Hinterlegen des öffentlichen Schlüssels in Eucalyptus
→ `euca-add-keypair`
 - Privater Schlüssel im Konfigurationsverzeichnis für Skript



SSH-Schlüssel einrichten

- Umgebungsvariablen aus eucarc einbinden

```
$ source ~/.euca/eucarc
```

- SSH-Keypair erzeugen und speichern (**einmalig**)

```
$ touch hadoop.sshkey
$ chmod 600 hadoop.sshkey
$ euca-add-keypair hadoop | tee -a hadoop.sshkey
KEYPAIR hadoop  xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx
-----BEGIN RSA PRIVATE KEY-----
[...]
-----END RSA PRIVATE KEY-----
```

- Privater SSH-Schlüssel in Datei hadoop.sshkey
- Öffentlicher Schlüssel in Eucalyptus als hadoop hinterlegt
- Übergabe an VM bei Instanziierung mittels -k

```
$ euca-run-instances -k hadoop emi-42424242
```



Verwendung der verteilten Hadoop-Instanzen

- Beispielkonfiguration kopieren und anpassen

```
$ cp -a /proj/i4mw/pub/aufgabe4/tools/sample-config my-cfg  
$ cp hadoop.sshkey my-cfg/ssh-private-key #Private Key  
$ vi my-cfg/core-site.xml #Eigene Anpassungen
```

- Konfiguration hochladen und Starten mittels Skript

```
$ /proj/i4mw/pub/aufgabe4/tools/start-hadoop.sh my-cfg \  
131.188.42.1 131.188.42.2 [...]
```

- Kopieren notwendiger .jar- und Datendateien in VM, z.B.:

```
$ scp -i hadoop.sshkey myapp.jar root@131.188.42.1:
```

- Einloggen und Job starten

```
$ ssh -i hadoop.sshkey root@131.188.42.1  
$ hadoop jar myapp.jar [...]
```



- Apache Hadoop Dokumentation

<http://hadoop.apache.org/common/docs/r0.20.2/index.html>

- Apache Hadoop API Javadoc

<http://hadoop.apache.org/common/docs/r0.20.2/api/index.html>

