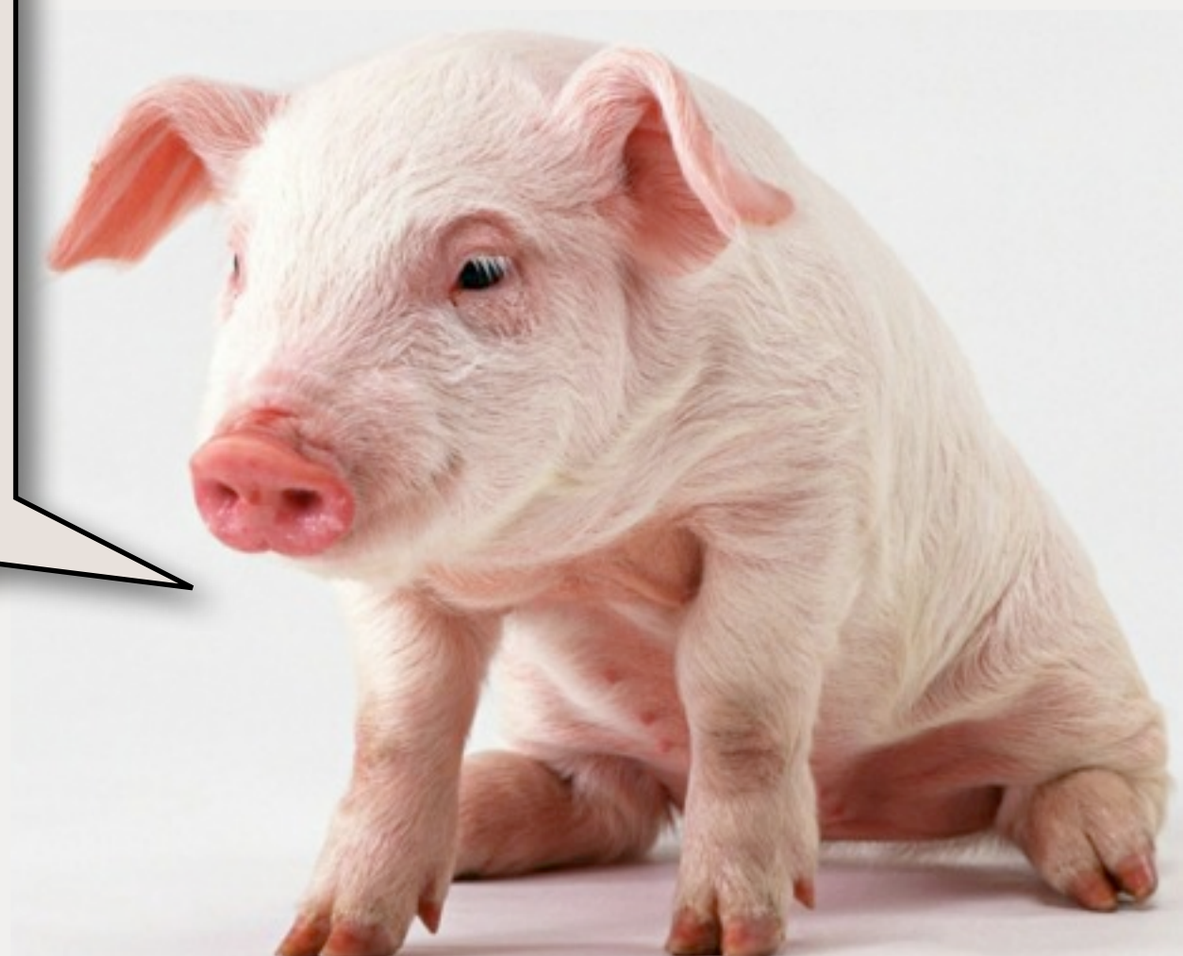
Schweine-Latein

Big Data Verarbeitung mit Apache Pig

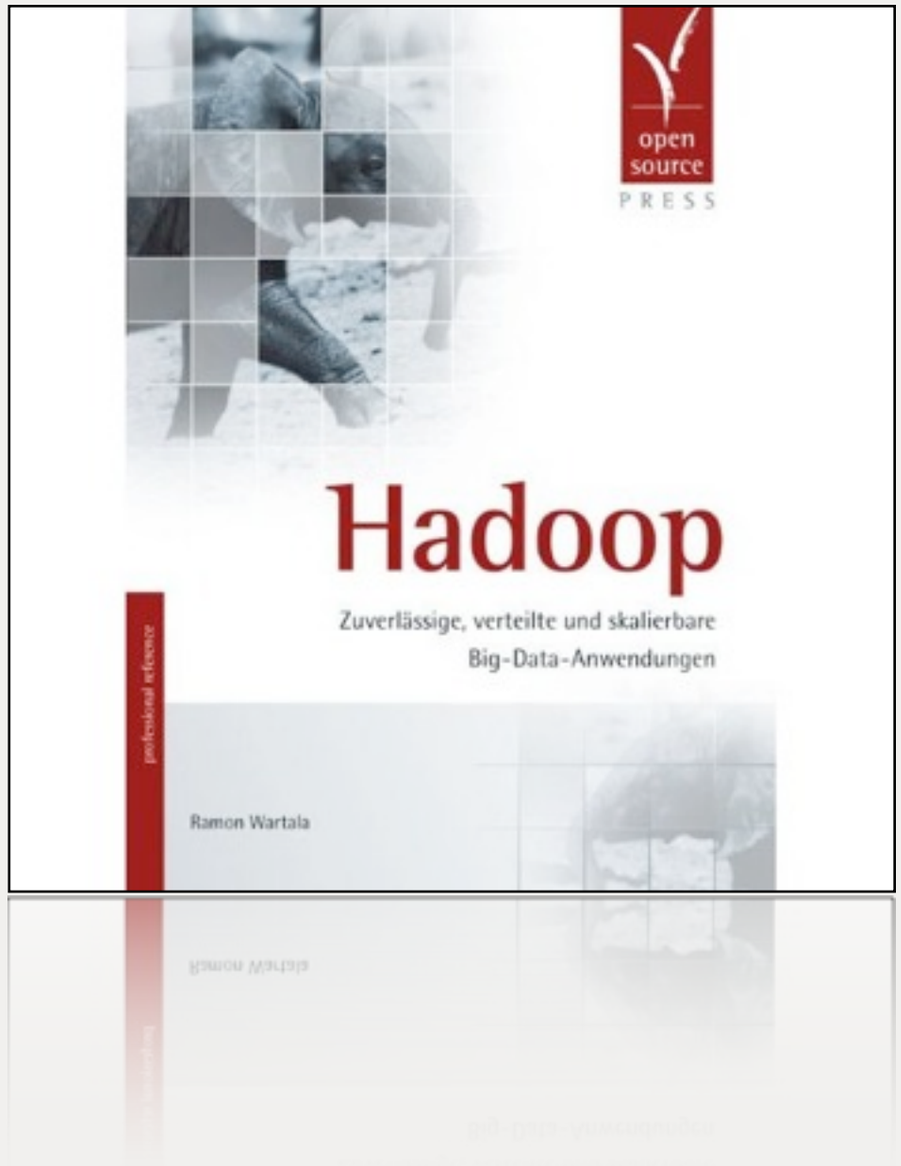
FrOSCon 7 - 25. August 2012

Von Ramon Wartala



Ramon Wartala

- **Diplom Informatiker**
- **Director Technology** bei der Online Marketing Agentur Performance Media GmbH in Hamburg
- **freier Autor** zum Thema Software Entwicklung und Daten Analyse
- Cloudera Certified Administrator for Apache Hadoop (**CCAH**)
- **Hadoop Trainer** und Speaker



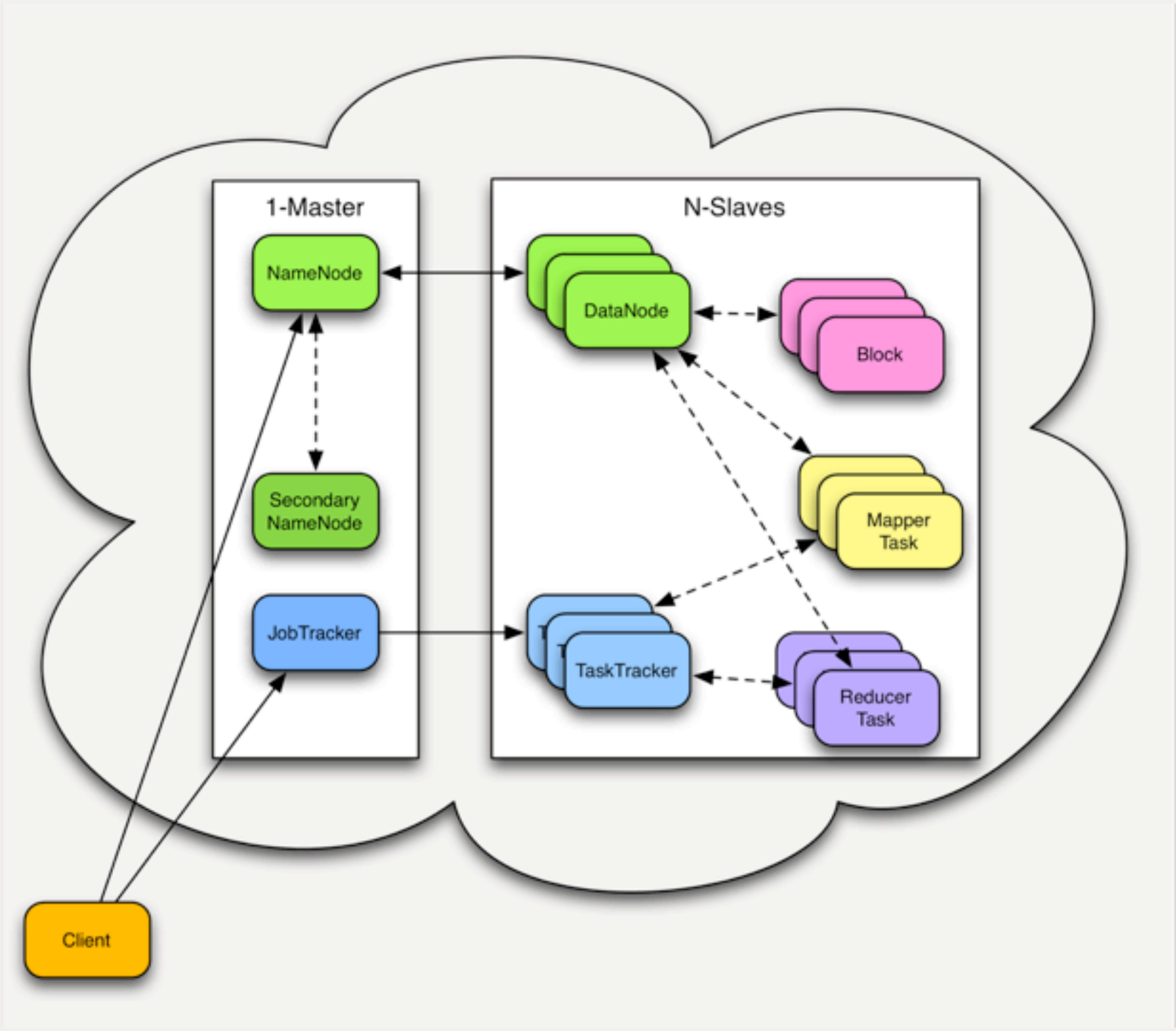
EMail: ramon.wartala@performance-media.de
Twitter: @rawar
Xing: https://www.xing.com/profile/Ramon_Wartala
Buch: https://www.opensourcepress.de/index.php?26&backPID=15&tt_products=343

Ein paar Worte zu Hadoop

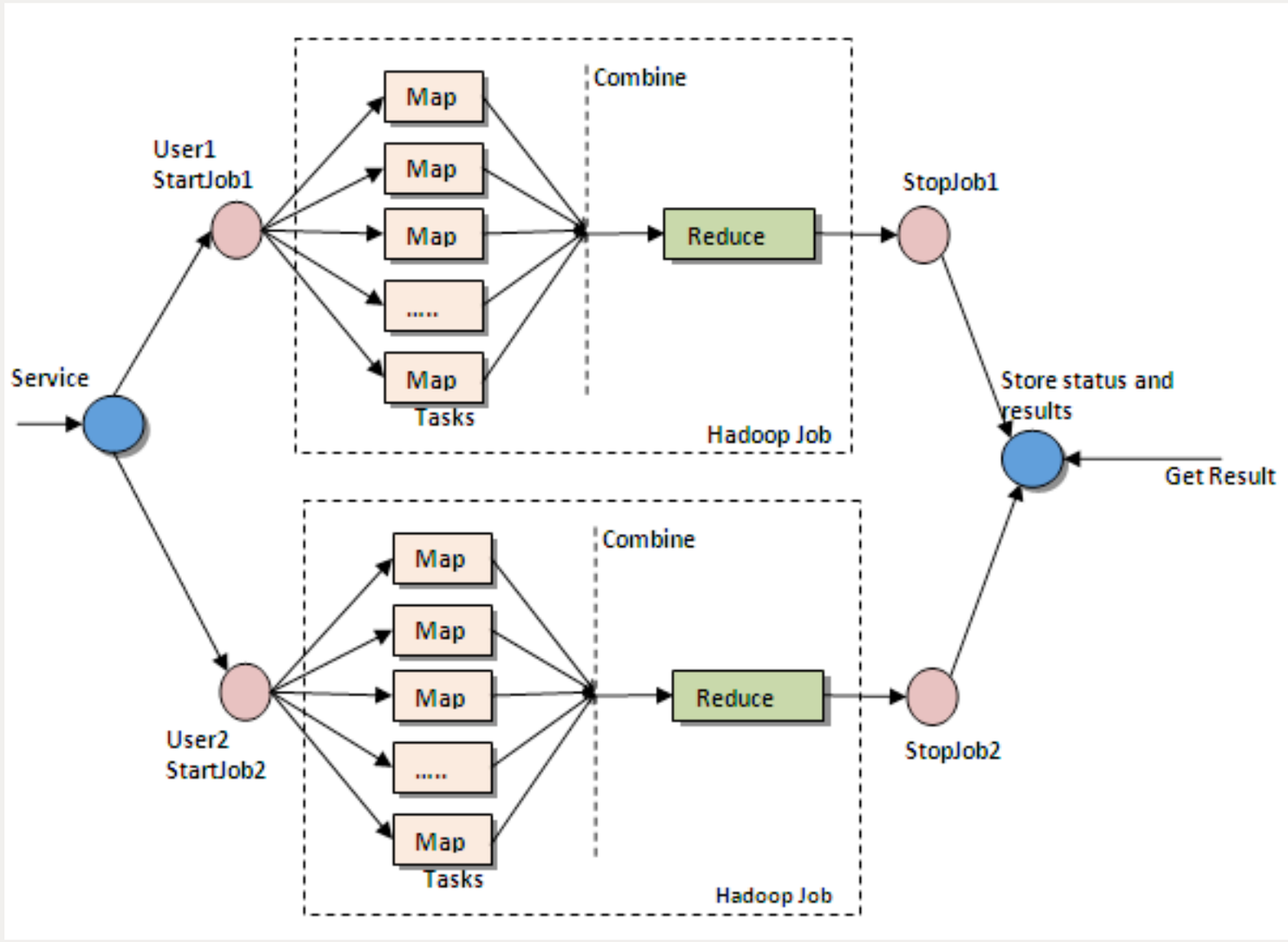
- Antwort auf die Frage: Wie lassen sich große Datenmengen skalierbar verarbeiten?
- Prinzip: Teile und Herrsche ☞ Viele Disks, viele Kerne, RAM, große Dateiblöcke
- Google Filesystem * + Google MapReduce ** (closed source)
- Ausfallsicher durch Datenredundanz (in der Regel 3x)
- Hadoop ist Open Source Implementierung von GFS / MapReduce in Java
- Verteiltes Dateisystem + MapReduce-Framework unter Apache Lizenz

* http://research.google.com/archive/gfs.html
**http://research.google.com/archive/mapreduce.html

Hadoop Architektur

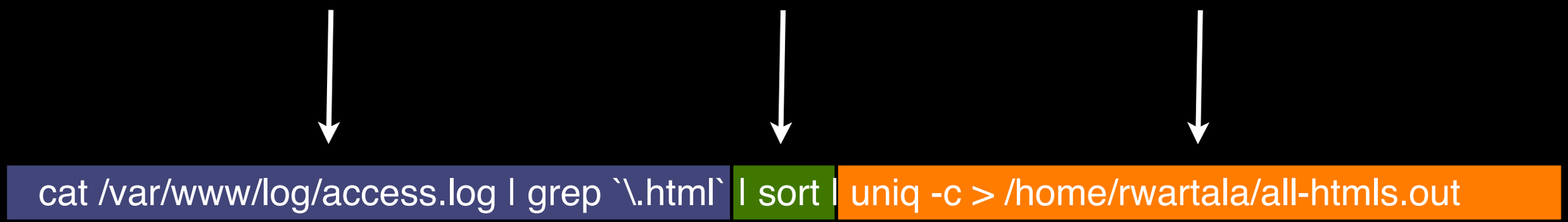


Was bringt MapReduce?



Quelle: http://map-reduce.wikispaces.asu.edu/file/view/figure5_(1).png/159615807/figure5_(1).png

Map Phase Shuffle Phase Reduce Phase



Was bringt MapReduce?

✓ Verteilung von Aufgaben über Rechnergrenzen hinweg

✓ Drei Phasen Modell: Map, Shuffle/Sort, Reduce

✓ Mapper-Tasks arbeitet auf einzelnen Daten-Records

✓ Reducer-Tasks aggregieren Ergebnisse der Mapper-Tasks

- In Hadoop: Mapper und Reducer werden in Java implementiert

- Implementierung **muss** in Map und Reduce „gedacht“ werden

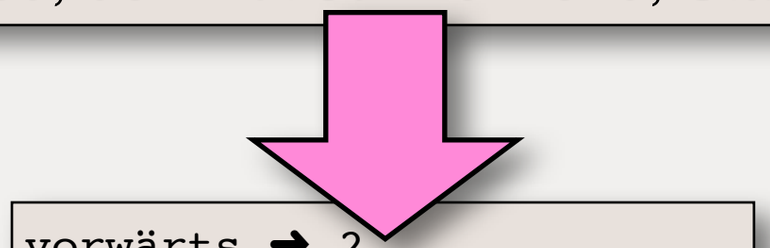
- Komplizierte Analysen müssen „zu Fuß“ entwickelt werden

➡ **Zu viel Java-Code!**

Hadoop hello world! - wordcount

- Programm liest Textdatei ein
- Programm extrahiert für jede Zeile des Textes die darin enthaltenen Wörter
- Für alle Wörter wird deren Vorkommen bestimmt:

„Vorwärts, rief Kapitän Hod, den Hut schwenkend, Stahlriese, vorwärts!“



```
vorwärts ➜ 2
rief ➜ 1
Kapitän ➜ 1
den ➜ 1
Hut ➜ 1
schwenkend ➜ 1
Stahlriese ➜ 1
```

Quelle: http://wiki.apache.org/hadoop/WordCount

```java
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map extends 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, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}

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

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

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

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

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

   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(TextOutputFormat.class);

   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.waitForCompletion(true);
}
```

Wobei hilft Pig?

- Ermöglicht die einfache Analyse großer Datensätze
- Pig Latin bietet Sprachumfang um komplexe Abfragen mit wenigen Zeilen Code zu implementiert
- Pig führt Pig Latin Skripte aus und **erzeugt** daraus eine Folge von **Mappern und Reducern** für Hadoop
- **Implementierungsaufwand** gegenüber nativen MapReduce-Anwendungen **deutlich reduziert**
- Auch von Datenanalysten nutzbar (!= Java Entwickler)

Der Beweis:
63 Zeilen Java-Code
vs.
5 Zeilen Pig Latin

```
1 Zeile = LOAD 'data/das-dampfhaus.txt' AS (zeile:chararray);
2 Worte = FOREACH Zeile GENERATE FLATTEN(TOKENIZE(zeile)) AS Wort;
3 Gruppe = GROUP Worte BY Wort;
4 Zaehler = FOREACH Gruppe GENERATE group, COUNT(Worte);
5 DUMP Zaehler;
```

Pigs Geschichte

- Pig wurde als Projekt bei Yahoo! Research gestartet
- Vorgestellt wurde die Arbeit* auf der SIGMOD 2008**
- Ziel von Pig die Nutzung von MapReduce zu vereinfachen
- Pig Latin als „SQL für MapReduce“
- „Schnelle und schmutziges“ Datenverarbeitung
- Apache Projekt und Teil des Hadoop Ökosystems

* http://infolab.stanford.edu/~usriv/papers/pig-latin.pdf
** http://www.sigmod08.org/

Wer nutzt Pig?

Quelle: https://cwiki.apache.org/confluence/display/PIG/PoweredBy


Was hat das mit
Schweine-Latein zu tun?

Pigs Philosophie*

- ***Pigs eat anything*** (mit oder ohne Datenschema)
- ***Pigs live anywhere*** (unterstützt die parallele Datenverarbeitung, aber nicht ausschliesslich)
- ***Pigs are domestic animals*** (sollen dem Nutzer helfen)
- ***Pigs fly*** (Pig verarbeitet Daten sehr schnell)

* http://pig.apache.org/philosophy.html

eat anything: Schema

- data = LOAD *Dateiname* USING *Loader* AS *Schema*;
- data = LOAD *Dateiname*;
- Pig geht von TSV-Dateien aus
- Schema-Position ($0 ... $N)
- Ohne Datentypen = alle Felder vom Typ bytearray
- PigStorage(), TextLoader(), ...
- Datei-Globs

	A	B	C
1	2000	Januar	318400
2	2000	Februar	321108
3	2000	Maerz	365609
4	2000	April	344540
5	2000	Mai	355112
6	2000	Juni	349851
7	2000	Juli	361692
8	2000	August	366615
9	2000	September	373474
10	2000	Oktober	397639

load_schema.pig (~/Desktop) – VIM

```
1 countainer_umschlag = LOAD '/daten/umgeschlagene_container.csv' USING
  PigStorage(';') AS (year: int, month: chararray; values: long);
```

eat anything: Datenformate

- Schema/Teil-Schema/Kein-Schema
- CSV/TSV/Texte
- HCatLoader/HCatStorer
- AvroStorage
- HBaseStorage
- HiveColumnarLoader aus /usr/lib/pig/contrib/piggybank/java/piggybank.jar ➜ RCFile (http://en.wikipedia.org/wiki/RCFile)
- Eigen Loader implementieren (http://pig.apache.org/docs/r0.9.2/

Eingabe / Ausgabe

- ***LOAD*** liest Daten mit Hilfe von Loadern TextLoader(), PigStorage()

- STORE speichert Daten

- DUMP gibt Daten in die Standardausgabe

- STREAM sendet die Daten an externe Anwendung

pig_load_ex1.pig (~/Desktop) – VIM

```
1 TXT = LOAD 'meine_daten.txt';
2 CSV = LOAD 'meine_daten.csv' USING PigStorage(',');
3 TSV = LOAD 'meine_daten.tsv' USING PigStorage('\t');
4 TSV2 = LOAD 'meine_daten.tsv' USING PigStorage(';') AS \
5   (col1:int, col2:int, col3:chararray, col4:float);


"~/Desktop/pig_load_ex1.pig" [New] 5L, 243C written
```

pig_store_dump_stream_ex1.pig (~/Desktop) – VIM

```
1 STORE CSV INTO 'meine_daten.csv' USING PigStorage(';');
2 A = LOAD 'meine_daten.csv' USING PigStorage(';');
3 DUMP A;
4 B = STREAM A THROUGH 'streaming_app.py';


"~/Desktop/pig_store_dump_stream_ex1.pig" [New] 4L, 155C written
```

live anywhere: Pig ausführen

- Keine Server-/Cluster-Installation (Pig ist Hadoop-Client)
- Pig Version muss Hadoop Version unterstützen (0.20, 1.0 etc.)
- Lokales (Datei-)System: `pig -x local grunzen.pig`
- Hadoop (Datei-)System: `pig -x mapreduce grunzen.pig`
- grunt>
- import org.apache.pig.PigServer;

Pig Datenmodell

- skalare und komplexe Typen
- skalare Typen: int, long, float, double, chararray (Strings), bytearray (Bytes),
- komplexe Typen: map (key/ values), tuple (Listen), bag (Liste aus Listen)
- Schema ist Definition von skalaren und komplexen Datentypen (ähnlich einer Tabelle in RDBS)
- keine Variablen, nur Relationen
- Pig Relationen sind immer bags von tuples
- Relationen heißen in Pig Latin ‚Alias‘
- Pig Befehle sind nicht Case-sensitiv, Relation schon
- Pig kann Hadoop-Dateibefehle ausführen

Komplexe Datentypen

- `tuple` - Reihe von Skalaren unterschiedlichen Types:

  ```
  ('FrOSCon', 7, '25', 8, 2012)
  ```

- `bag` - Ungeordnete Liste von tuples:

  ```
  {('FrOSCon,7,'25.08.2012'),('Schweine-Latein',11,15)}
  ```

- `map` - Schlüssel-/Wertepaare. Schlüssel muss vom Typ chararray sein. Werte können beliebige Skalare sein:

  ['25.08.2012'#'FrOSCon 7']

Pigs Relationale Operationen

- FOREACH
- FILTER
- GROUP
- ORDER BY
- DISTINCT
- JOIN
- LIMIT
- SAMPLE
- CROSS
- UNION
- DESCRIBE
- EXPLAIN
- ILLUSTRATE
- SPLIT
- DUMP
- REGISTER
- ...

Beispiele: Relationale Operationen

FOREACH transformiert die Daten spaltenweise ähnlich SELECT von SQL

FILTER reduziert die Datenmenge über Bedingungen

ORDER BY sortiert Daten

DISTINCT filtert doppelte Datensätze

pig_foreach_ex1.pig (~) – VIM

```
1 year_value = FOREACH sorted_container GENERATE year, value;
2 DUMP year_value;
```

pig_filter_ex1.pig (~) – VIM

```
1 values_of_2011 = FILTER sorted_container BY year == '2011';
2 DUMP values_of_2011;
```

pig_order_by_ex1.pig (~) – VIM

```
1 container_by_year_value = ORDER container BY year.value;
```

pig_distinct_ex1.pig + (~) – VIM

```
1 ship_names = LOAD '/tmp/schiffsnamen.txt';
2 unique_ship_names = DISTINCT ship_names;
3 DUMP unique_ship_names;+
```

Beispiele: Relationale Operationen

JOIN vereinigung zweier Relationen mit gleichen Schlüssel (-Werten)

UNION führt zwei oder mehr Relationen zusammen

CROSS erzeugt kartesisches Produkt zweier Relationen

pig_join_ex1.pig + (~/Desktop) – VIM

```
1 ship_names = LOAD '/tmp/ship_names.csv' USING \
2   PigStorage(';') AS (id:int, name:chararray);
3
4 number_of_masted = LOAD '/tmp/ship_masted.csv' USING \
5   PigStorage(';'i) AS (id:int, number_of_masted: int);
6
7 ship_names_and_masted = JOIN ship_names BY id, number_of_masted BY id;
```

pig_union_ex1.pig (~/Desktop) – VIM

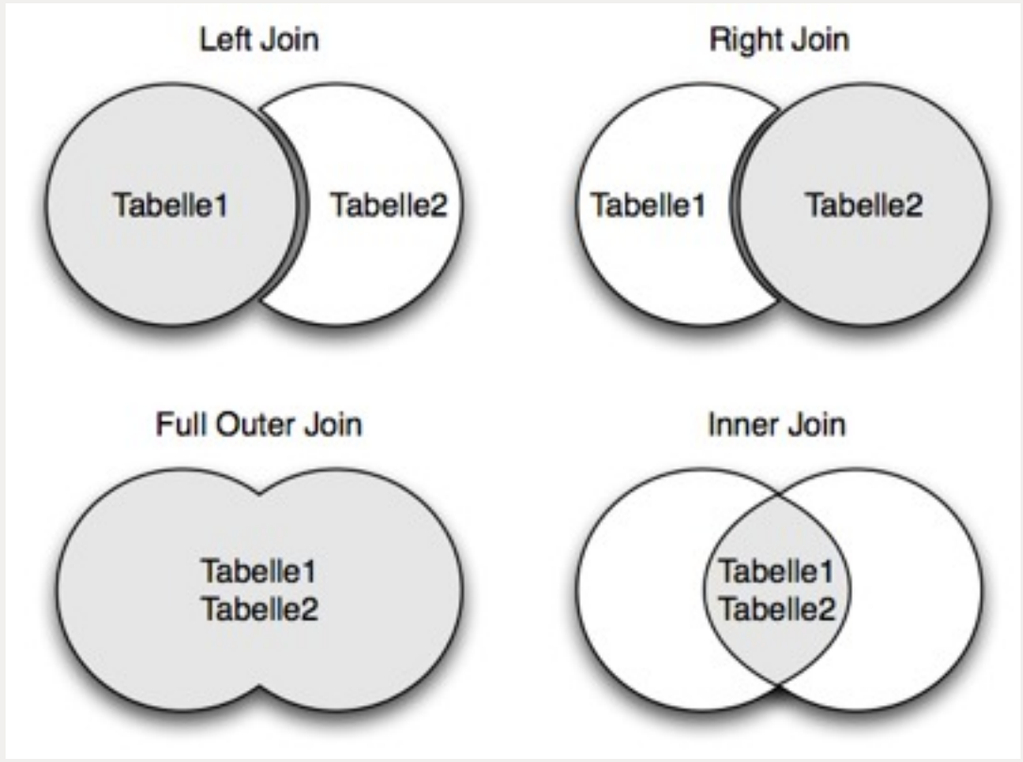
```
1 ship_names = LOAD '/tmp/ship_names.csv' \
2   USING PigStorage(';') AS (id:int, name:chararray);
3 boat_names = LOAD '/tmp/boat_names.csv' \
4   USING PigStorage(';') AS (id:int, name:chararray);
5 both_tables = UNION ship_names, boat_names;
6 DUMP both_tables;
```

pig_union_ex1.pig + (~/Desktop) – VIM

```
1 ship_names = LOAD '/tmp/ship_names.csv' \
2   USING PigStorage(';') AS (id:int, name:chararray);
3 boat_names = LOAD '/tmp/boat_names.csv' \
4   USING PigStorage(';') AS (id:int, name:chararray);
5 crossed_tables = CROSS ship_names, boat_names;
6 DUMP crossed_tables;
```

Verschiedene JOINs

- `JOIN` (Left, Outer etc.)
- `USING 'merge'` ➜ Merge Join nützlich wenn Eingabedaten bereits sortiert
- `USING 'skewed'` ➜ Join bei unterschiedlich vielen Keys
- using `'replicated'` ➜Fragment-Replicated Join ➜ die zu joinenden Daten werden im Speicher der Mapper-Phase gehalten ➜ keine Reducer-Phase



Pigs Ausführung lässt sich analysieren

- Pig erzeugt eigene Mapper und Reducer
- `EXPLAIN alias;` oder `pig -e "explain -script wordcount.pig"`
- `ILLUSTRATE alias;` ☞ zeigt die einzelnen Schritte der Verarbeitung von alias auf
- `DESCRIBE alias;` ☞ zeigt Typ der Relation an
- `DUMP alias;` ☞ gibt Inhalt der Relation aus

```
#--------------------------------------------------
# Map Reduce Plan
#--------------------------------------------------
MapReduce node scope-49
Map Plan
Gruppe: Local Rearrange[tuple]{chararray}(false) - scope-61
|   |
|   Project[chararray][0] - scope-62
|
|---Zaehler: New For Each(false,false)[bag] - scope-50
    |   |
    |   Project[chararray][0] - scope-51
    |   |
    |   POUserFunc(org.apache.pig.builtin.COUNT$Initial)[tuple] - scope-52
    |   |
    |   |---Project[bag][1] - scope-53
    |
    |---Pre Combiner Local Rearrange[tuple]{Unknown} - scope-63
        |
        |---Worte: New For Each(true)[bag] - scope-37
            |   |
            |   POUserFunc(org.apache.pig.builtin.TOKENIZE)[bag] - scope-35
            |   |
            |   |---Cast[chararray] - scope-34
            |       |
            |       |---Project[bytearray][0] - scope-33
            |
            |---Zeile: Load(hdfs://r1nn1.pmd.local:8020/user/root/data/das-dampfhaus.txt:org.apache.pig.builtin.PigStorage) - scope-32--------
Combine Plan
Gruppe: Local Rearrange[tuple]{chararray}(false) - scope-65
|   |
|   Project[chararray][0] - scope-66
|
|---Zaehler: New For Each(false,false)[bag] - scope-54
    |   |
    |   Project[chararray][0] - scope-55
    |   |
    |   POUserFunc(org.apache.pig.builtin.COUNT$Intermediate)[tuple] - scope-56
    |   |
    |   |---Project[bag][1] - scope-57
    |
    |---POCombinerPackage[tuple]{chararray} - scope-59--------
Reduce Plan
Zaehler: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-48
|
|---Zaehler: New For Each(false,false)[bag] - scope-47
    |   |
    |   Project[chararray][0] - scope-42
    |   |
    |   POUserFunc(org.apache.pig.builtin.COUNT$Final)[long] - scope-45
    |   |
    |   |---Project[bag][1] - scope-58
```

Testing und Diagnose

- Unit Tests mit **PigUnit** (http://pig.apache.org/docs/r0.9.2/test.html#pigunit)
- **Penny** - Monitoring und Debugging (https://cwiki.apache.org/confluence/display/PIG/PennyToolLibrary)

Bild von http://www.flickr.com/photos/judo10/4108776118/in/photostream

Einbetten

- Pig kann in alle gängigen „Java-resken“ Sprachen eingebettet werten: JRuby, Jython, Rhino etc.

- Vorteil: Benutzerdefinierte Funktionen lassen sich einfacher implementieren

- Schleifen können einfacher realisiert werden

- Einfachere Parameter-Behandlung

pig_python_ex1.py (~) – VIM

```
1 #!/usr/bin/python
2
3 from org.apache.pig.scripting import *
4 P = Pig.compile(...)
"~/pig_python_ex1.py" 4L, 79C
```

Benutzer Definierte Funktionen

- User Defined Functions (http://wiki.apache.org/pig/UDFManual)
- **PiggyBank.jar** (https://cwiki.apache.org/PIG/piggybank.html)
- **DataFu** von LinkedIn (https://github.com/linkedin/datafu):
  - PageRank
  - Statistische Funktionen wie Quantile, Varianz etc.
  - Geo-Distanz-Bestimmung
- **Elephant-Bird** von Twitter (https://github.com/kevinweil/elephant-bird)
  - Lzo-, Protobuffer- JSON-, HBase-Loader

Nutzung in der Cloud

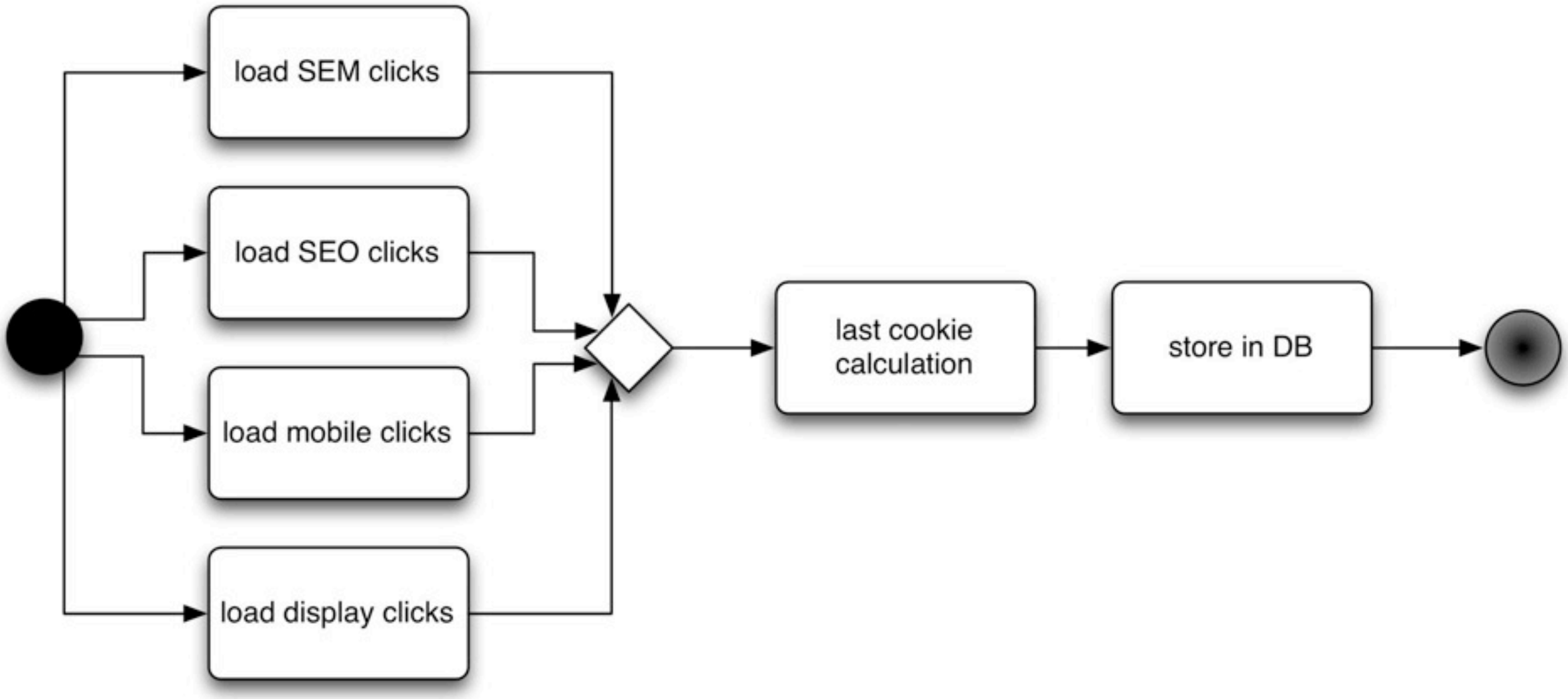
- Apache Pig Nutzung innerhalb von Amazon Elastic Map Reduce möglich
- AMI Version 2.2.0 (Stand 6.8.2012): Hadoop 1.0.3 und Pig 0.9.2.2*

* http://docs.amazonwebservices.com/ElasticMapReduce/latest/DeveloperGuide/EnvironmentConfig_AMIVersion.html#ami-versions-supported

Typische Anwendungsfälle

- ETL
- Daten-Pipeline (Sortierung, Filterung, Aggregation)
- Logdatei-Analysen
- Click-Pfad-Analysen
- Clickfraud-Analysen
- Multikanal-Analysen

ETL-Prozess


load SEM clicks
load SEO clicks
load mobile clicks
load display clicks
last cookie calculation
store in DB

Fragen?

Quellen

- **Mailinglisten**

  http://pig.apache.org/mailing_lists.html

- **Webseiten**

  http://pig.apache.org/

  http://pig.apache.org/docs/r0.9.2/

- **Artikel**

  http://www.ibm.com/developerworks/linux/library/l-apachepigdataquery/

  http://www.slideshare.net/AdamKawa/apache-pig-at-whug

  http://www.cloudera.com/wp-content/uploads/2010/01/IntroToPig.pdf

- **Bücher**: Programming Pig von Alan Gates