

13 June 2012

MapReduce

Introduction and Hadoop Overview

Lab Course: Databases & Cloud Computing
SS 2012

Martin Przyjaciel-Zablocki
Alexander Schätzle
Georg Lausen

University of Freiburg
Databases & Information Systems



Agenda

1. Why MapReduce?

- a. Data Management Approaches
- b. Distributed Processing of Data

2. Apache Hadoop

- a. HDFS
- b. MapReduce
- c. Pig
- d. Hive
- e. HBase

3. Programming MapReduce

- a. MapReduce with Java
- b. Moving into the Cloud

MapReduce: Why?



▶ Large datasets

- Amount of data increases constantly
- “Five exabytes of data are generated every to days”
(corresponds to the whole amount of data generated up to 2003) by Eric Schmidt

▶ Facebook:

- >800 million active users in Facebook, interacting with >1 billion objects
- 2.5 petabytes of userdata per day!

The Facebook logo, consisting of the word "facebook" in white lowercase letters on a blue rectangular background, is located to the right of the Facebook-related text.

▶ How to explore, analyze such large datasets?

The last.fm logo, featuring the text "last.fm" in white lowercase letters on a red rectangular background, is positioned at the bottom center of the slide.The Google logo, with its characteristic multi-colored letters, is located in the bottom right corner of the slide.

MapReduce: Why?

- ▶ **Processing 100 TB dataset**
- ▶ On 1 node
 - Scanning @ 50 MB/s = **23 days**
- ▶ On 1000 node cluster
 - Scanning @ 50 MB/s = **33 min**
- ▶ **Current development**
 - Companies often can't cope with logged user behavior and **throw away data** after some time → lost opportunities
 - Growing **cloud-computing** capacities
 - Price/performance advantage of **low-end servers** increases to about a factor of twelve

Data Management Approaches

▶ High-performance single machines

- “Scale-up” with limits (hardware, software, costs)
- Workloads today are beyond the capacity of any single machine
- I/O Bottleneck

▶ Parallel Databases

- Fast and reliable
- “Scale-out” restricted to some hundreds machines
- Maintaining & administrations of parallel databases is hard

▶ Specialized cluster of powerful machines

- “specialized” = powerful hardware satisfying individual software needs
- fast and reliable but also very expensive
- For data-intensive applications: scaling “out” is superior to scaling “up” → performance gap insufficient to justify the price

Data Management Approaches (2)

► Clusters of commodity servers (with MapReduce)

“Commodity servers” = not individually adjusted

- e.g. 8 cores, 16G of RAM
- Cost & energy efficiency

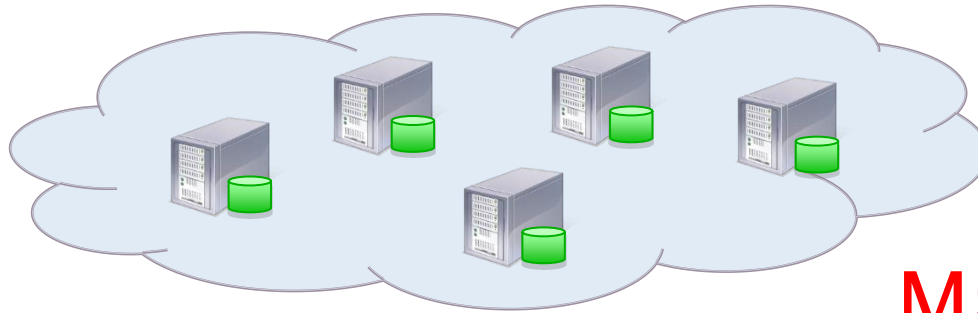
MapReduce

- Designed around clusters of commodity servers
- Widely used in a broad range of applications
- By many organizations
- Scaling “out”, e.g. Yahoo! uses > 40.000 machines
- Easy to maintain & administrate



Distributed Processing of Data

- ▶ **Problem:** How to compute the PageRank for a crawled set of websites on a cluster of machines?



MapReduce!

- ▶ **Main Challenges:**
 - How to break up a large problem into smaller tasks, that can be executed in parallel?
 - How to assign tasks to machines?
 - How to partition and distribute data?
 - How to share intermediate results?
 - How to coordinate synchronization, scheduling, fault-tolerance?

Big ideas behind MapReduce

- ▶ **Scale “out”, not “up”**
 - Large number of commodity servers
- ▶ **Assume failures are common**
 - In a cluster of 10000 servers, expect 10 failures a day.
- ▶ **Move processing to the data**
 - Take advantage of data locality and avoid to transfer large datasets through the network
- ▶ **Process data sequentially and avoid random access**
 - Random disk access causes seek times
- ▶ **Hide system-level details from the application developer**
 - Developers can focus on their problems instead of dealing with distributed programming issues
- ▶ **Seamless scalability**
 - Scaling “out” improves the performance of an algorithm without any modifications

MapReduce

▶ MapReduce

- Popularized by Google & widely used
- Algorithms that can be expressed as (or mapped to) a sequence of **Map()** and **Reduce()** functions are automatically parallelized by the framework

▶ Distributed File System

- Data is split into equally sized blocks and stored distributed
- Clusters of commodity hardware
→ Fault tolerance by replication
- Very large files / write-once, read-many pattern

▶ Advantages

- Partitioning + distribution of data
- Parallelization and assigning of task
- Scalability, fault-tolerance, scheduling,...

That all is done automatically!

What is MapReduce used for?

▶ At Google

- Index construction for Google Search (replaced in 2010 by Caffeine)
- Article clustering for Google News
- Statistical machine translation



▶ At Yahoo!

- “Web map” powering Yahoo! Search
- Spam detection for Yahoo! Mail



▶ At Facebook

- Data mining, Web log processing
- SearchBox (with Cassandra)
- Facebook Messages (with HBase)
- Ad optimization
- Spam detection



What is MapReduce used for?

► In research

- Astronomical image analysis (Washington)
- Bioinformatics (Maryland)
- Analyzing Wikipedia conflicts (PARC)
- Natural language processing (CMU)
- Particle physics (Nebraska)
- Ocean climate simulation (Washington)
- Processing of Semantic Data (Freiburg)
- <Your application here>

Agenda

1. Why MapReduce?

- a. Data Management Approaches
- b. Distributed Processing of Data

2. Apache Hadoop

- a. HDFS
- b. MapReduce
- c. Pig
- d. Hive
- e. HBase

3. Programming MapReduce

- a. MapReduce with Java
- b. Moving into the Cloud

2. Apache Hadoop

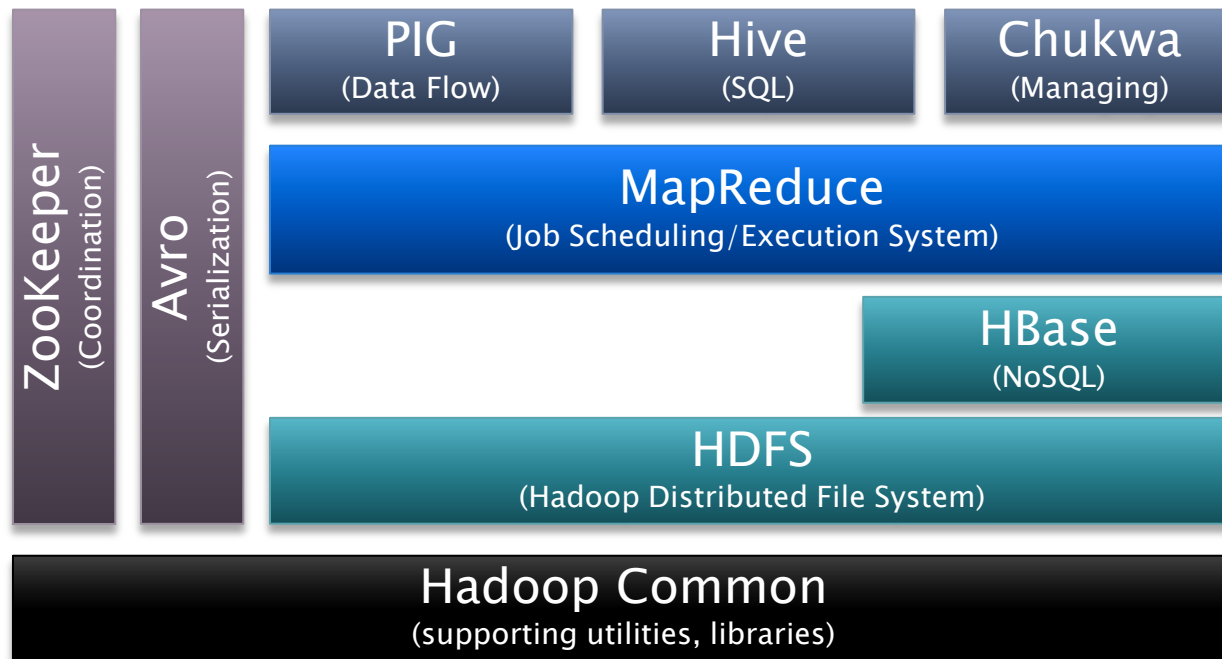
»» “Open–source software for reliable, scalable, distributed computing”

Apache Hadoop: Why?

▶ Apache Hadoop

- Well-known Open-Source implementation of
- Google's MapReduce & Google File System (GFS) paper
- Enriched by many subprojects
- Used by Yahoo, Facebook, Amazon, IBM, Last.fm, EBay ...
- Cloudera's Distribution with VMWare images, tutorials and further patches

Hadoop Ecosystem



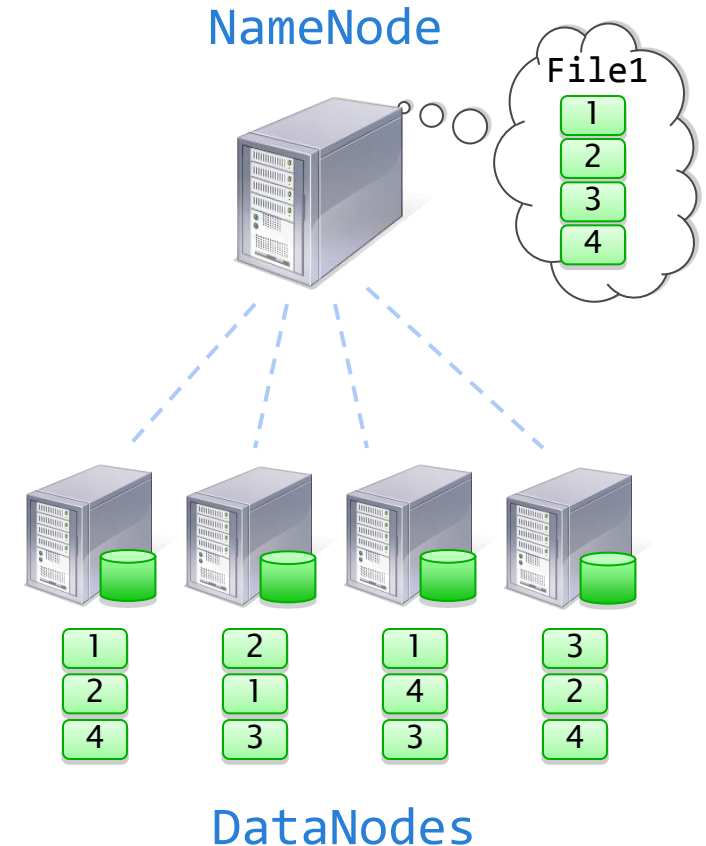
Yahoo's Hadoop Cluster



Image from <http://wiki.apache.org/hadoop-data/attachments/HadoopPresentations/attachments/aw-apachecon-eu-2009.pdf>

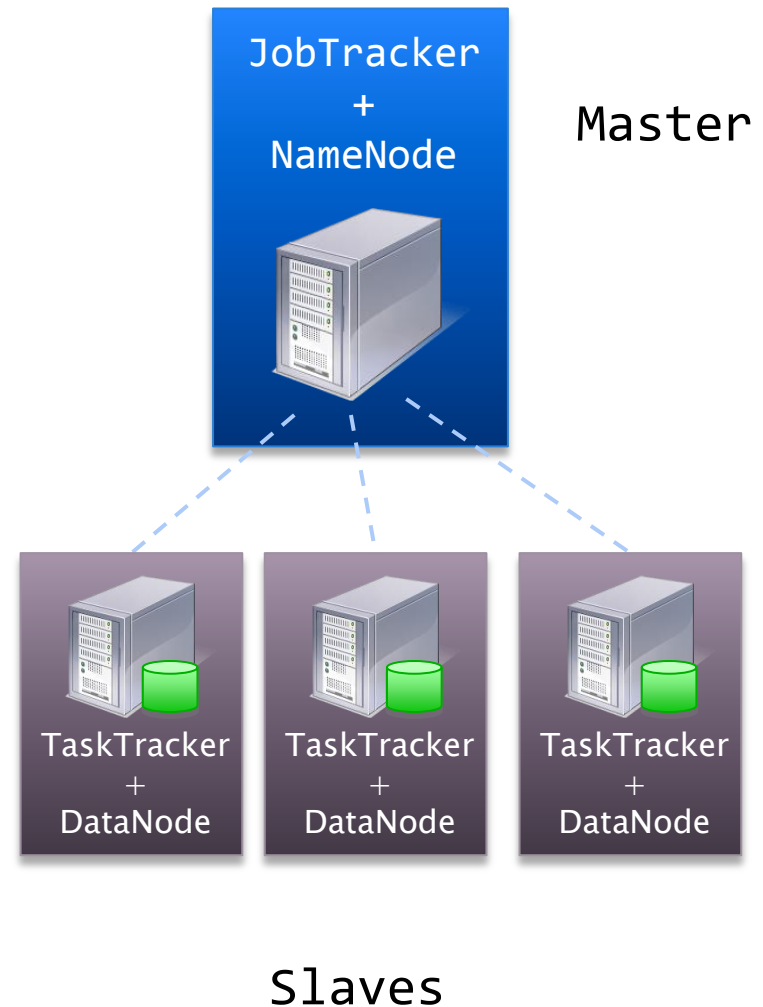
Hadoop Distributed File System

- ▶ Files split into 64MB *blocks*
- ▶ Blocks replicated across several **DataNodes** (usually 3)
- ▶ Single **NameNode** stores metadata
 - file names, block locations, etc
- ▶ Optimized for large files, sequential reads
- ▶ Files are append-only



Hadoop Architecture

- ▶ Master & Slaves architecture
- ▶ **JobTracker** schedules and manages jobs
- ▶ **TaskTracker** executes individual map() and reduce() task on each cluster node
- ▶ JobTracker and Namenode as well as TaskTrackers and DataNodes are placed on the same machines



MapReduce Workflow

(1) Map Phase

- Raw data read and converted to key/value pairs
- Map() function applied to any pair

(2) Shuffle Phase

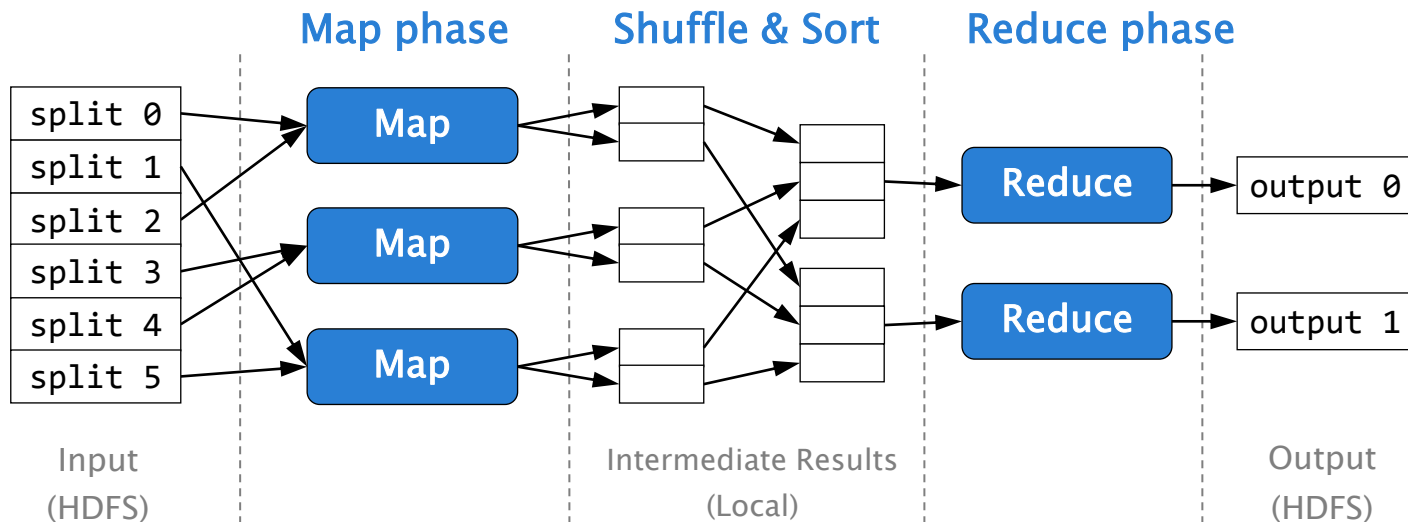
- All key/value pairs are sorted and grouped by their keys

(3) Reduce Phase

- All values with a the same key are processed by within the same reduce() function

MapReduce Workflow (2)

► Steps of a MapReduce execution



► Signatures

- **Map():** `(in_key, in_value) → list (out_key, intermediate_value)`
- **Reduce():** `(out_key, list (intermediate_value)) → list (out_value)`

MapReduce Programming Model

- ▶ Every MapReduce program must specify a **Mapper** and typically a **Reducer**
- ▶ The Mapper has a **map()** function that transforms input **(key, value)** pairs into any number of intermediate **(out_key, intermediate_value)** pairs
- ▶ The **Reducer** has a **reduce()** function that transforms intermediate **(out_key, list(intermediate_value))** aggregates into any number of output **(value')** pairs

MapReduce Execution Details

- ▶ Single **master** controls job execution on multiple **slaves**
 - Master/Slave architecture
- ▶ Mappers preferentially placed on **same** node or same rack as their input block
 - Utilize Data-locality → move computation to data
 - Minimizes network usage
- ▶ Mappers save outputs to local disk before serving them to reducers
 - Allows recovery if a reducer crashes
 - Allows having more reducers than nodes

Word Count Example

- ▶ Problem: Given a document, we want to count the occurrences of any word
- ▶ Input:
 - Document with words (e.g. Literature)
- ▶ Output:
 - List of words and their occurrences, e.g.
“Infrastructure” 12
“the” 259
...

Word Count Execution

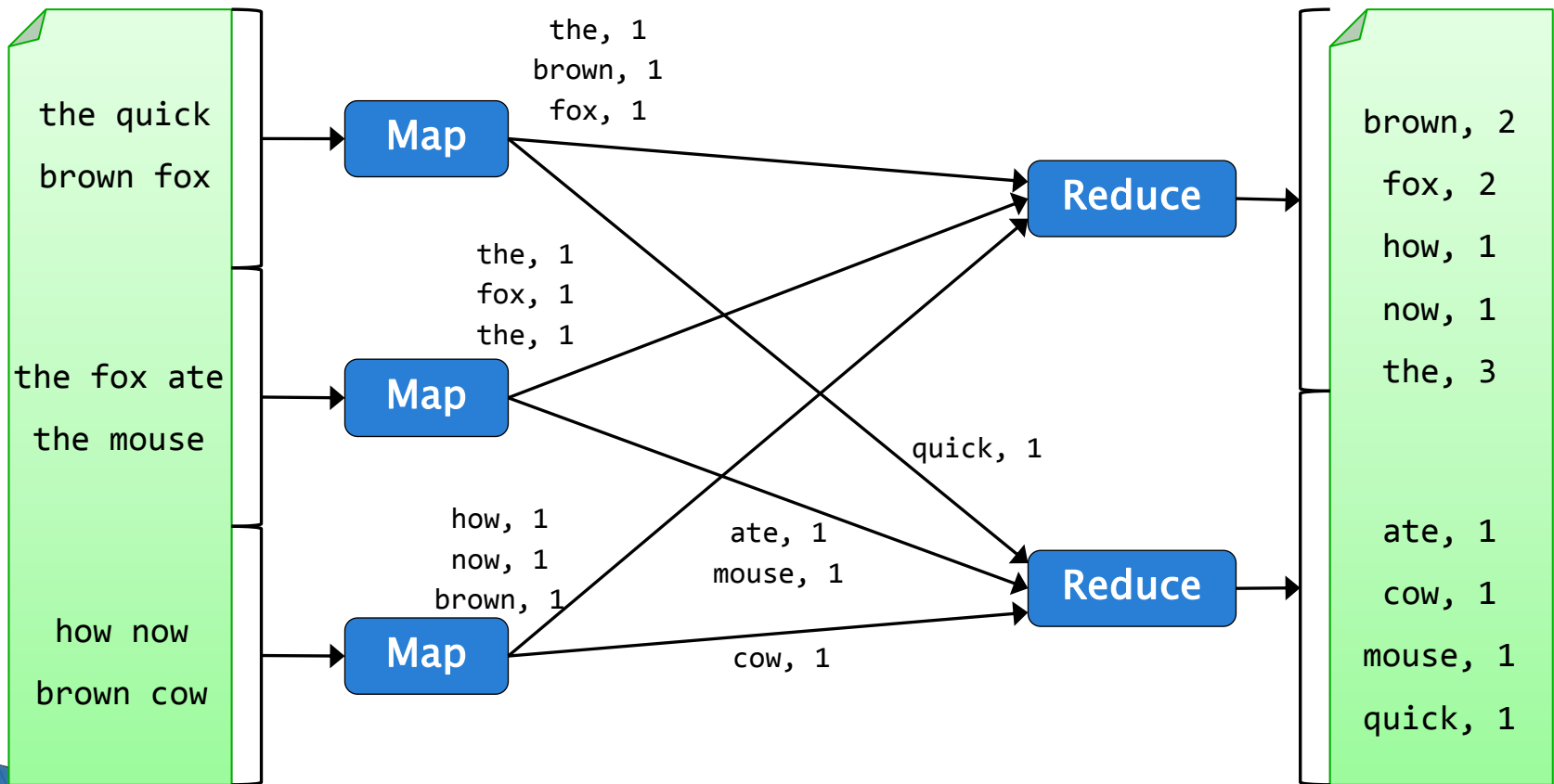
Input

Map

Shuffle & Sort

Reduce

Output



An Optimization: The Combiner

- ▶ A combiner is a local aggregation function for repeated keys produced by same Mapper
- ▶ Works for associative functions like sum, count, max
- ▶ Decreases size of intermediate data
- ▶ Example: map-side aggregation for Word Count

Word Count with Combiner

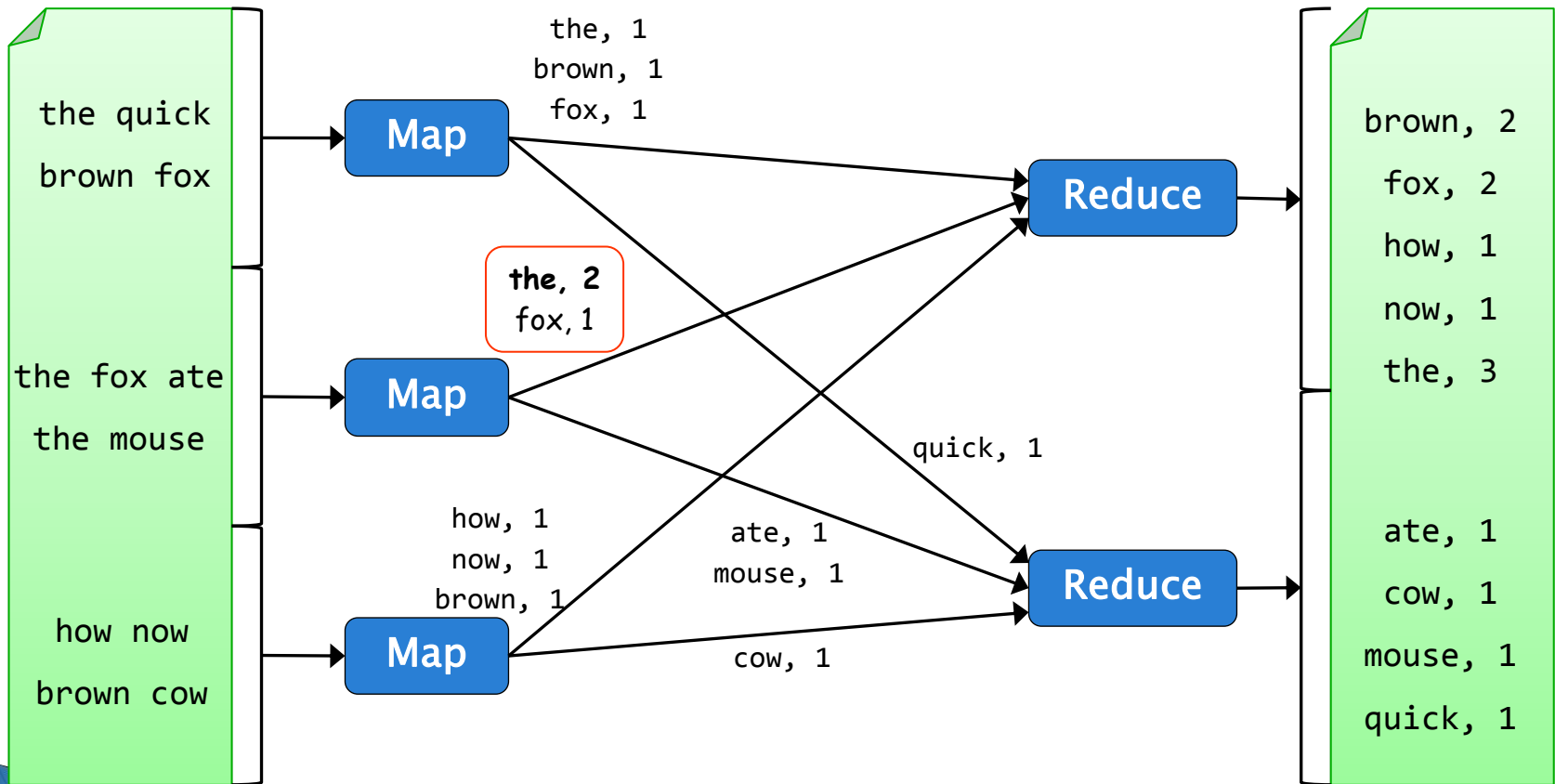
Input

Map

Shuffle & Sort

Reduce

Output



Fault Tolerance in MapReduce

1. If a task crashes:

- Retry on another node
 - OK for a map because it has no dependencies
 - OK for reduce because map outputs are on disk
 - If the same task fails repeatedly, fail the job or ignore that input block (user-controlled)
- Note: For these fault tolerance features to work, your map and reduce tasks must be **side-effect-free**

Fault Tolerance in MapReduce

2. If a node crashes:

- Re-launch its current tasks on other nodes
- Re-run any maps the node previously ran
 - Necessary because their output files were lost along with the crashed node

Fault Tolerance in MapReduce

3. If a task is going slowly (straggler):

- Launch second copy of task on another node (“speculative execution”)
 - Take the output of whichever copy finishes first, and kill the other
- ▶ Surprisingly important in large clusters
- Stragglers occur frequently due to failing hardware, software bugs, misconfiguration, etc
 - Single straggler may noticeably slow down a job

Further handy tools

▶ Partitioners

- Assign keys to reduces
- Default: `key.hashCode() % num_reducers`

▶ Grouping Comparators

- Sort keys within reduces

▶ Combiners

- Local aggregation

▶ Compression

- Supported compression types: `zlib`, `LZO`,...

▶ Counters (global)

- Define new countable events

Further handy tools (2)

▶ Zero Reduces

- If no sorting or shuffling required
- Set number of reduces to 0

▶ Distributed File Cache

- For storing read-only copies of data on local computers

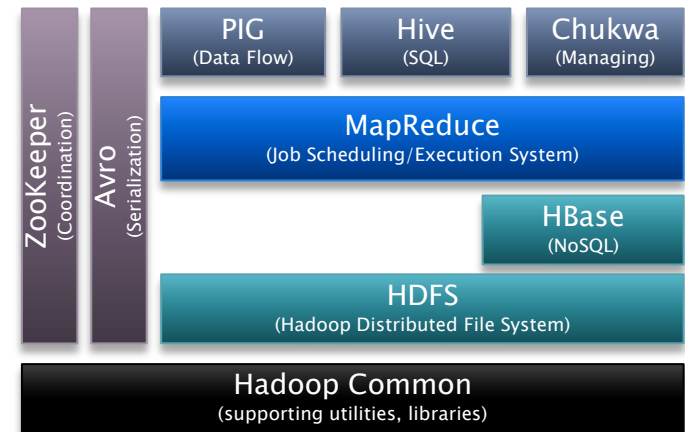
Agenda

1. Why MapReduce?

- a. Data Management Approaches
- b. Distributed Processing of Data

2. Apache Hadoop

- a. HDFS
- b. MapReduce
- c. Pig**
- d. Hive
- e. HBase



3. Programming MapReduce

- a. MapReduce with Java
- b. Moving into the Cloud

Pig (Latin): Why?

- ▶ Many parallel algorithms can be expressed by a series of MapReduce jobs
- ▶ But **MapReduce is fairly low-level**:
 - must think about keys, values, partitioning, etc
- ▶ Can we capture common “job building blocks”?

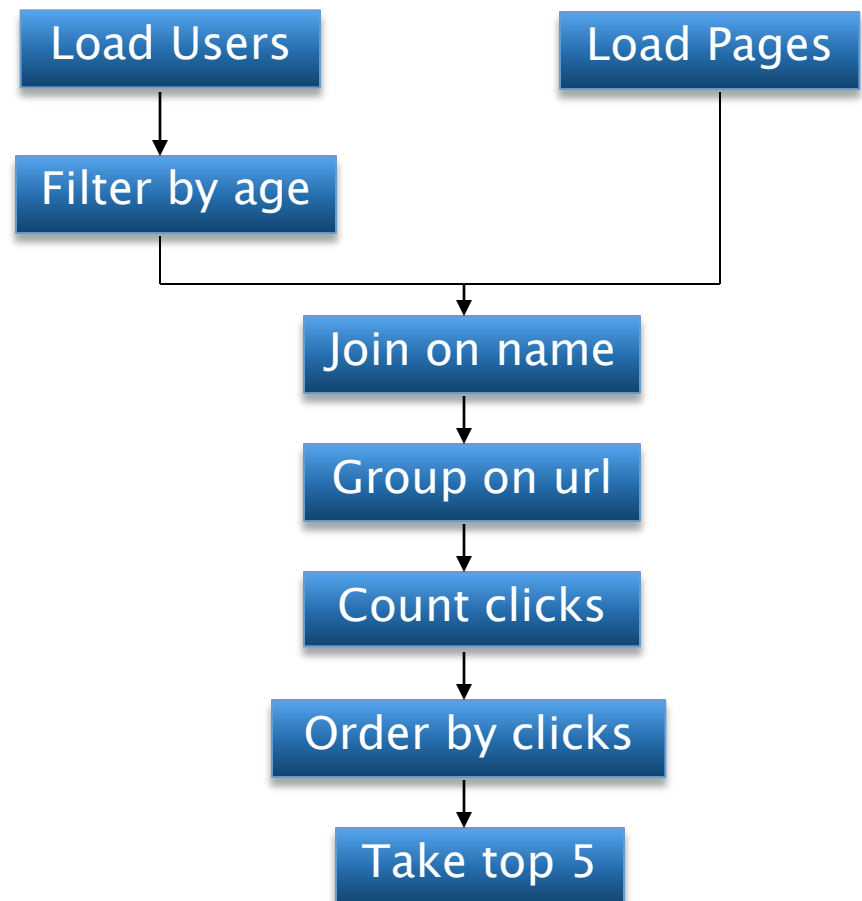
Pig (Latin)

- ▶ Started at [Yahoo! Research](#)
- ▶ Runs about 30% of Yahoo!'s jobs
- ▶ Features:
 - Expresses sequences of MapReduce jobs
 - Data model: nested “bags” of items
 - [Provides relational \(SQL\) operators](#) (JOIN, GROUP BY, etc.)
 - Easy to plug in Java functions
 - Pig Pen development environment for Eclipse



An Example Problem

- ▶ Suppose you have **user data** in one file, **page view data** in another
- ▶ and you need to **find the top 5** most visited pages by users aged 18 – 25



In MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapperContext;
import org.apache.hadoop.mapred.MapperRunner;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.ReducerContext;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.JobControl;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text> {

        public void map(LongWritable k, Text val,
            OutputCollector<Text, Text> cc,
            Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Respond an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            cc.collect(outKey, outVal);
        }
    }

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

        public void map(LongWritable k, Text val,
            OutputCollector<Text, Text> cc,
            Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(0, firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 35) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Respond an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            cc.collect(outKey, outVal);
        }
    }

    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
            Iterator<Text> iter,
            OutputCollector<Text, Text> cc,
            Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
                    first.add(value.substring(1));
                else second.add(value.substring(1));
            }

            reporter.setStatus("OK");
        }

        // Do the cross product and collect the values
        for (String a1 : first) {
            for (String a2 : second) {
                String outVal = key + "," + a1 + "," + a2;
                cc.collect(outVal, new Text(outVal));
                reporter.setStatus("OK");
            }
        }
    }

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

        public void map(
            Text k,
            Text val,
            OutputCollector<Text, LongWritable> cc,
            Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore.
            // Just pass a 1 for the combiner/reducer to sum stored
            Text outKey = new Text(key);
            cc.collect(outKey, new LongWritable(1));
        }
    }

    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
            Writable> {

        public void reduce(
            Text k,
            Iterator<LongWritable> iter,
            OutputCollector<WritableComparable, Writable> cc,
            Reporter reporter) throws IOException {
            // Add up all the values we see
            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }
            cc.collect(key, new LongWritable(sum));
        }
    }

    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
            Text> {

        public void map(
            WritableComparable key,
            Writable val,
            OutputCollector<LongWritable, Text> cc,
            Reporter reporter) throws IOException {
            cc.collect((LongWritable)val, (Text)key);
        }
    }

    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;

        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> cc,
            Reporter reporter) throws IOException {
            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                cc.collect(key, iter.next());
                count++;
            }
        }
    }

    public static void main(String[] args) throws IOException {
        JobConf jg = new JobConf(MRExample.class);
        jg.setJobName("Load Pages");
        jg.setInputFormat(TextInputFormat.class);

        jg.setOutputKeyClass(Text.class);
        jg.setOutputValueClass(Text.class);
        jg.setMapperClass(LoadPages.class);
        jg.setInputFormat(TextInputFormat.class);
        jg.setOutputPath(jg, new
            Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(jg,
            new Path("/user/gates/tmp/indexed_pages"));
        jg.setNumReduceTasks(0);
        Job loadPages = new Job(jg);

        JobConf jg2 = new JobConf(MRExample.class);
        jg2.setJobName("Load and Filter Users");
        jg2.setInputFormat(KeyValueTextInputFormat.class);
        jg2.setOutputKeyClass(Text.class);
        jg2.setOutputValueClass(Text.class);
        jg2.setMapperClass(LoadAndFilterUsers.class);
        jg2.setInputFormat(TextInputFormat.class);
        jg2.setOutputPath(jg2, new
            Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(jg2,
            new Path("/user/gates/tmp/filtered_users"));
        jg2.setNumReduceTasks(0);
        Job loadUsers = new Job(jg2);

        JobConf jg3 = new JobConf(MRExample.class);
        jg3.setJobName("Join Users and Pages");
        jg3.setInputFormat(KeyValueTextInputFormat.class);
        jg3.setOutputKeyClass(Text.class);
        jg3.setOutputValueClass(Text.class);
        jg3.setMapperClass(IdentityMapper.class);
        jg3.setReducerClass(Join.class);
        jg3.setInputFormat(TextInputFormat.class);
        jg3.setOutputPath(jg3, new
            Path("/user/gates/tmp/joined"));
        FileInputFormat.setInputPath(jg3, new
            Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(jg3, new
            Path("/user/gates/tmp/joined_users"));
        jg3.setNumReduceTasks(0);
        Job joinJob = new Job(jg3);
        jg3.addDependingJob(loadPages);
        jg3.addDependingJob(loadUsers);
        Job joinGroup = new Job(jg3);

        JobConf jg4 = new JobConf(MRExample.class);
        jg4.setJobName("Group URLs");
        jg4.setInputFormat(KeyValueTextInputFormat.class);
        jg4.setOutputKeyClass(Text.class);
        jg4.setOutputValueClass(LongWritable.class);
        jg4.setMapperClass(LoadJoined.class);
        jg4.setReducerClass(ReduceUrls.class);
        jg4.setInputFormat(TextInputFormat.class);
        jg4.setOutputPath(jg4, new
            Path("/user/gates/tmp/joined"));
        FileInputFormat.setInputPath(jg4, new
            Path("/user/gates/tmp/joined"));
        jg4.setNumReduceTasks(0);
        Job groupJob = new Job(jg4);
        groupJob.addDependingJob(joinJob);

        JobConf jg5 = new JobConf(MRExample.class);
        jg5.setJobName("Top 100 sites");
        jg5.setInputFormat(SequenceFileInputFormat.class);
        jg5.setOutputKeyClass(LongWritable.class);
        jg5.setOutputValueClass(Text.class);
        jg5.setMapperClass(LoadClicks.class);
        jg5.setReducerClass(LimitClicks.class);
        jg5.setInputFormat(SequenceFileInputFormat.class);
        jg5.setOutputPath(jg5, new
            Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(jg5, new
            Path("/user/gates/top100sitesforusers100"));
        jg5.setNumReduceTasks(0);
        Job limit = new Job(jg5);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
            18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

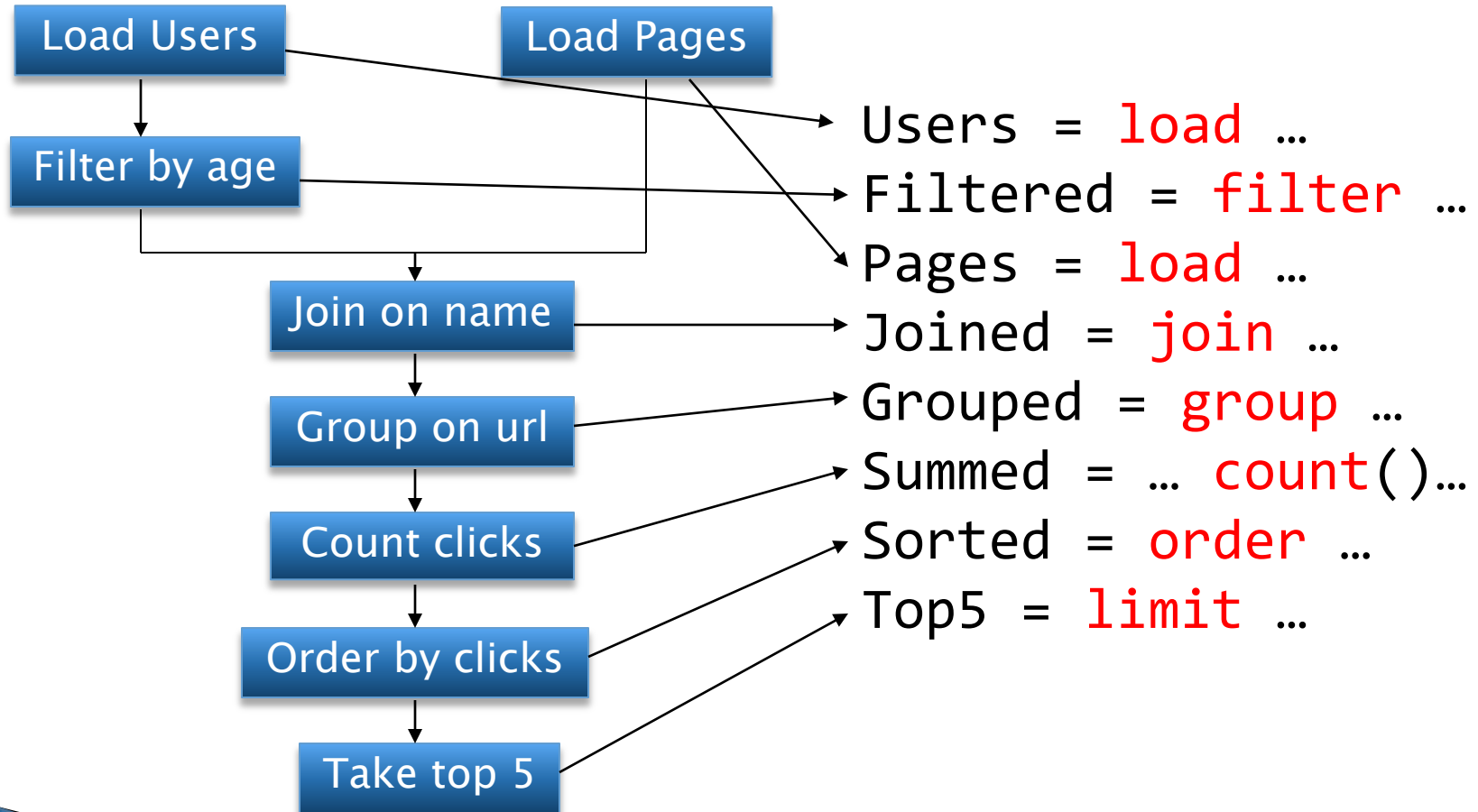
In Pig Latin

```
Users      = load 'users' as (name, age);
Filtered   = filter Users by
              age >= 18 and age <= 25;
Pages      = load 'pages' as (user, url);
Joined     = join Filtered by name, Pages by user;
Grouped    = group Joined by url;
Summed     = foreach Grouped generate group,
              count(Joined) as clicks;
Sorted     = order Summed by clicks desc;
Top5       = limit Sorted 5;

store Top5 into 'top5sites';
```

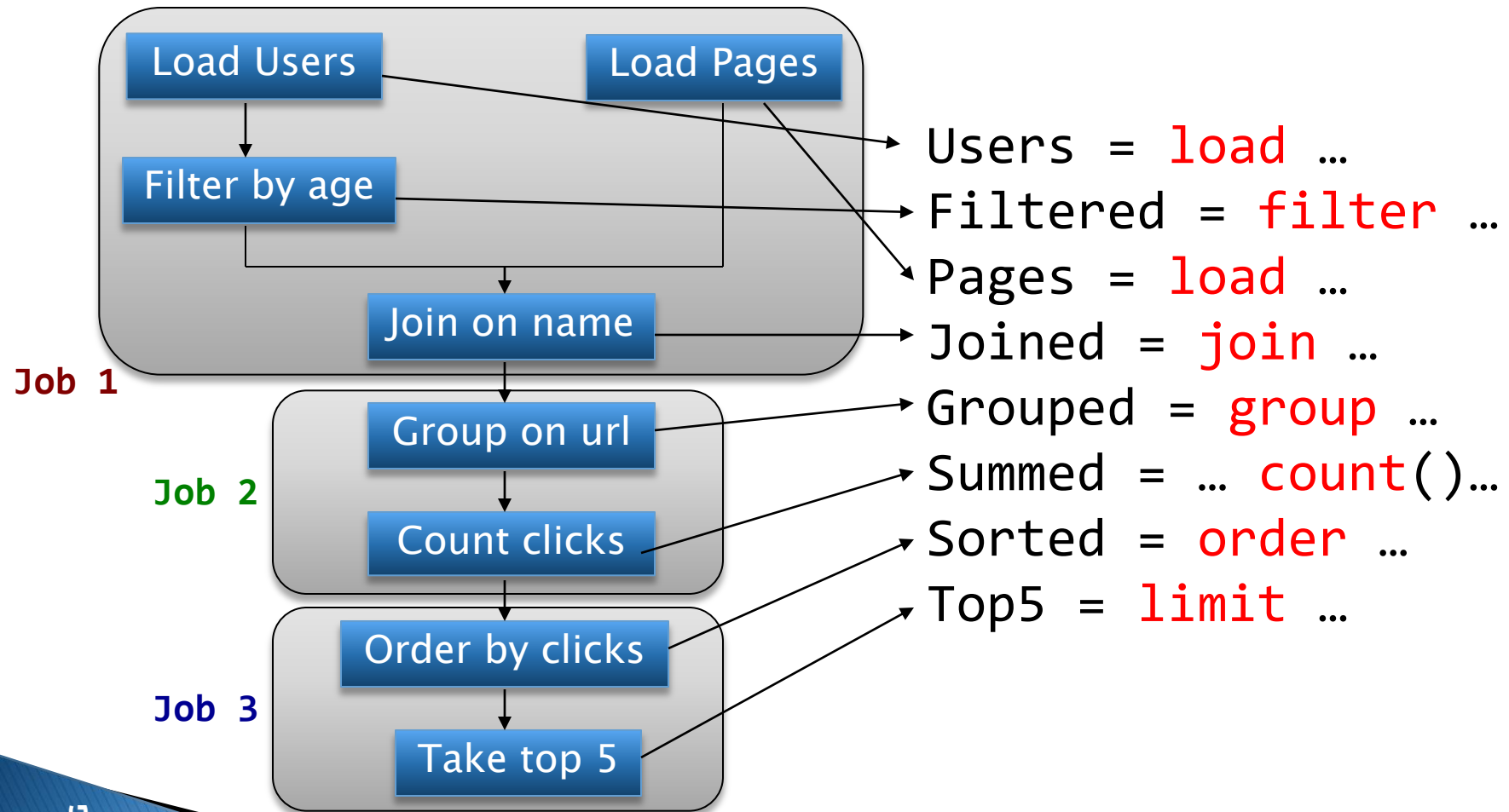
Ease of Translation

Notice how naturally the components of the job translate into Pig Latin.



Ease of Translation

Notice how naturally the components of the job translate into Pig Latin.



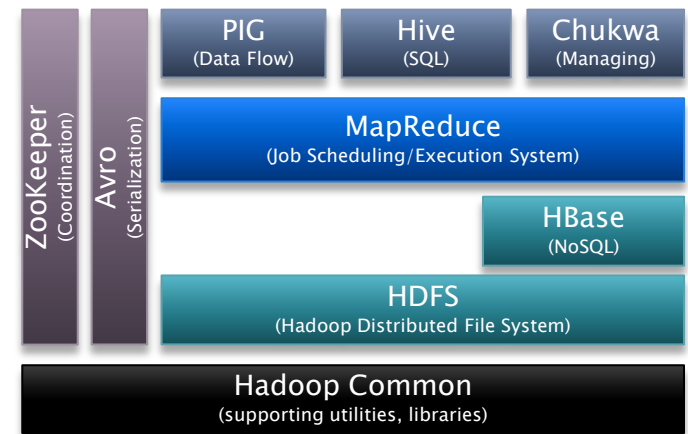
Agenda

1. Why MapReduce?

- a. Comparison of Data Management Approaches
- b. Distributed Processing of Data

2. Apache Hadoop

- a. HDFS
- b. MapReduce
- c. Pig
- d. Hive
- e. HBase



3. Programming MapReduce

- a. MapReduce with Java
- b. Moving into the Cloud

Hive

- ▶ Developed at **Facebook**
- ▶ Used for majority of Facebook jobs
- ▶ **Data Warehouse infrastructure** that provides data summarization and ad hoc querying on top of Hadoop
 - MapReduce for execution
 - HDFS for storage
- ▶ “**Relational database**” built on Hadoop
 - Maintains list of table schemas
 - **SQL-like query language** (HQL)
 - Supports table partitioning, clustering, complex data types, some optimizations



Sample Hive Query

- ▶ Find top 5 pages visited by users aged 18–25:

```
SELECT p.url, COUNT(1) as clicks
FROM users u JOIN page_views p ON (u.name = p.user)
WHERE u.age >= 18 AND u.age <= 25
GROUP BY p.url
ORDER BY clicks
LIMIT 5;
```



HBase

- ▶ Clone of Big Table (Google)
- ▶ Data is stored “Column-oriented”
- ▶ Distributed over many servers
- ▶ Layered over HDFS
- ▶ Strong consistency (CAP Theorem)
- ▶ Scalable up to billions of rows x millions of columns



And many others ...

▶ Sqoop

- Tool designed to help users of large data import existing relational databases into their Hadoop cluster
- Integrates with Hive

▶ Zookeeper

- High-performance coordination service for distributed applications

▶ Avro

- Data serialization system

▶ Chukwa

- Data collection system
- Displaying, monitoring and analyzing log files

Takeaways

- ▶ By providing a data-parallel programming model, **MapReduce** can control job execution in useful ways:
 - Automatic division of job into tasks
 - Automatic partition and distribution of data
 - Automatic placement of computation near data
 - Recovery from failures & stragglers
- ▶ **Hadoop**, an open source implementation of MapReduce, enriched by many useful subprojects
- ▶ User focuses on application, not on complexity of distributed computing

Agenda

1. Why MapReduce?

- a. Comparison of Data Management Approaches
- b. Distributed Processing of Data

2. Apache Hadoop

- a. HDFS
- b. MapReduce
- c. Pig
- d. Hive
- e. HBase

3. Programming MapReduce

- a. MapReduce with Java
- b. Moving into the Cloud

3. Programming MapReduce

»» First steps with Hadoop

Getting started with Hadoop

- ▶ Hadoop Distributions
 - [Apache Hadoop](#)
 - [Cloudera's Hadoop Distribution](#) (**recommended**)
 - ...
- ▶ Installing Hadoop on Linux
 - Follow CDH3 Installation Guide
- ▶ Hadoop within a Virtual Machine
 - Cloudera's Hadoop Demo VMWare Image
 - Ready to use Hadoop Environment
- ▶ Hadoop in the Cloud
 - Amazon's Elastic MapReduce

New MapReduce API

- ▶ Since Hadoop 0.20

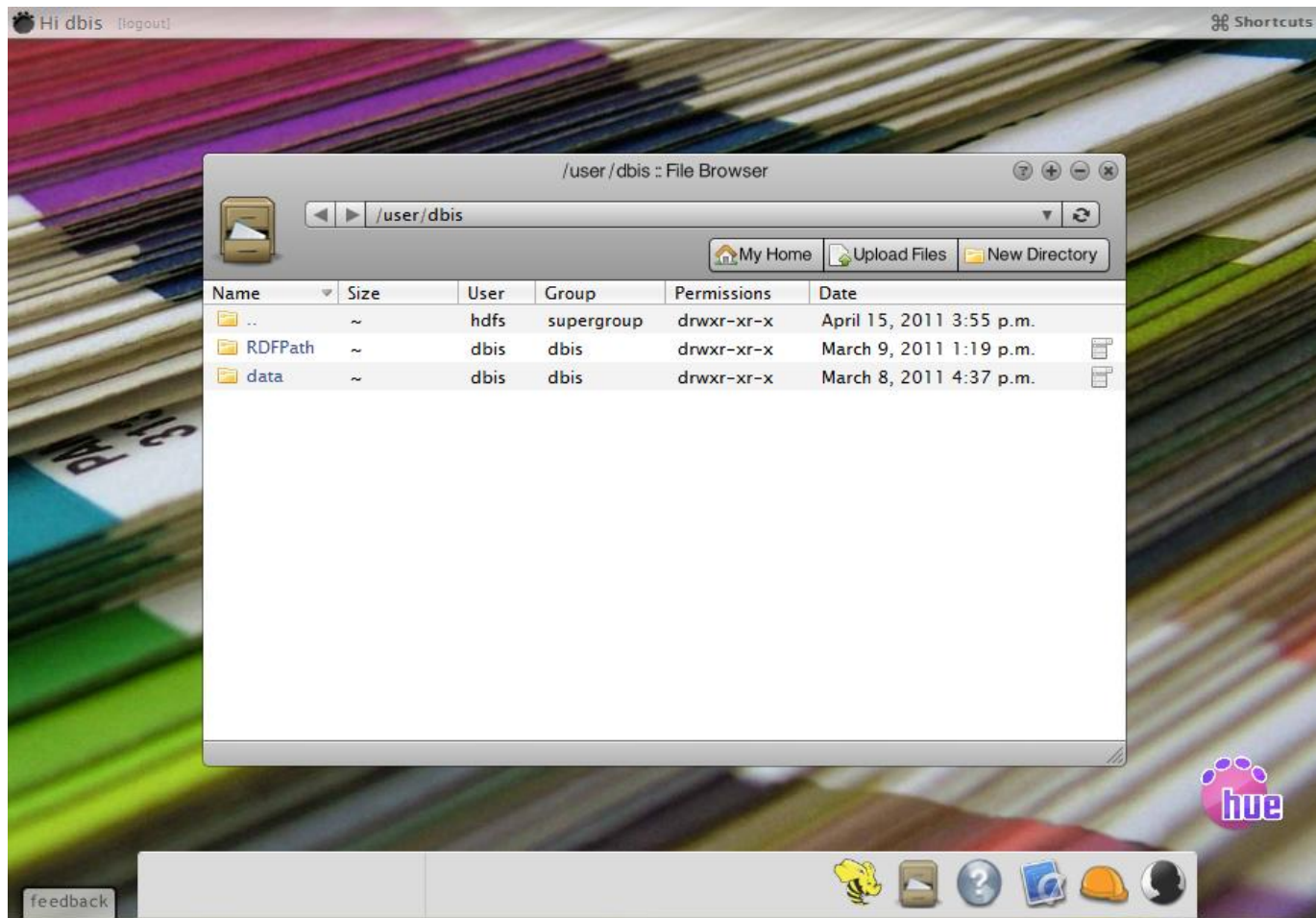
- ~~OLD:~~ ~~org.apache.hadoop.mapred.*~~

- NEW: org.apache.hadoop.mapreduce.*

- ▶ Note

- Many available examples are written using the old API
 - One should not mix both
 - Strongly recommended: new API!

Web Interfaces: HUE



Web Interfaces: JobTracker

sydney Hadoop Map/Reduce Administration

[Quick Links](#)

State: RUNNING

Started: Wed Apr 06 15:28:21 CEST 2011

Version: 0.20.2-CDH3B4, r3aa7c91592ea1c53f3a913a581dbfcdfebe98bfe

Compiled: Mon Feb 21 22:40:16 UTC 2011 by root

Identifier: 201104061528

Cluster Summary (Heap Size is 7.12 MB/888.94 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Excluded Nodes
0	0	0	9	0	0	0	0	36	18	6.00	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)

Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Retired Jobs

none

Local Logs

<http://masterIP:50030/jobtracker.jsp>

Web Interfaces: NameNode

NameNode 'sydney.informatik.privat:8020'

Started:	Wed Apr 06 15:24:30 CEST 2011
Version:	0.20.2-CDH3B4, r3aa7c91592ea1c53f3a913a581dbfcdfebe98bfe
Compiled:	Mon Feb 21 22:40:16 UTC 2011 by root
Upgrades:	There are no upgrades in progress.

[Browse the filesystem](#)

[Namenode Logs](#)

Cluster Summary

2745 files and directories, 10238 blocks = 12983 total. Heap Size is 19.88 MB / 888.94 MB (2%)

Configured Capacity	:	24.08 TB
DFS Used	:	1.65 TB
Non DFS Used	:	1.28 TB
DFS Remaining	:	21.15 TB
DFS Used%	:	6.86 %
DFS Remaining%	:	87.85 %
Live Nodes	:	9
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

NameNode Storage:

Storage Directory	Type	State
/hadoop-dir1/hdfs/name	IMAGE_AND_EDITS	Active
/vol2/hadoop-dir2/hdfs/name	IMAGE_AND_EDITS	Active

[Cloudera's Distribution for Hadoop, 2011.](#)

<http://masterIP:50070/dfshealth.jsp>

Agenda

1. Why MapReduce?

- a. Comparison of Data Management Approaches
- b. Distributed Processing of Data

2. Apache Hadoop

- a. HDFS
- b. MapReduce
- c. Pig
- d. Hive
- e. HBase

3. Programming MapReduce


- a. MapReduce with Java
- b. Moving into the Cloud

Amazon Elastic MapReduce

- ▶ Provides a web-based interface and command-line tools for running Hadoop jobs on Amazon EC2
- ▶ Data stored in Amazon S3
- ▶ Monitors job and shuts down machines after use
- ▶ Small extra charge on top of EC2 pricing

Elastic MapReduce Workflow

Create a New Job Flow

Cancel 

DEFINE JOB FLOW

SPECIFY PARAMETERS

CONFIGURE EC2 INSTANCES

REVIEW

Creating a job flow to process your data using Amazon Elastic MapReduce is simple and quick. Let's begin by giving your job flow a name and selecting its type. If you don't already have an application you'd like to run on Amazon Elastic MapReduce, samples are available to help you get started.

Job Flow Name*:

The name can be anything you like and doesn't need to be unique. It's a good idea to name the job flow something descriptive.

Type*: ☒ Streaming

A Streaming job flow allows you to write single-step mapper and reducer functions in a language other than java.

☐ Custom Jar (advanced)

A custom jar on the other hand gives you more complete control over the function of Hadoop but must be a compiled java program. Amazon Elastic MapReduce supports custom jars developed for Hadoop 0.18.3.

☐ Pig Program

Pig is a SQL-like language built on top of Hadoop. This option allows you to define a job flow that runs a Pig script, or set up a job flow that can be used interactively via SSH to run Pig commands.

☐ Sample Applications

Select a sample application and click Continue. Subsequent forms will be filled with the necessary data to create a sample Job Flow.

Word Count (Streaming)



Word count is a Python application that counts occurrences of each word in provided documents. [Learn more and view license](#)


Continue



* Required field

Elastic MapReduce Workflow

Create a New Job Flow

Cancel 

✓
DEFINE JOB FLOW

SPECIFY PARAMETERS

CONFIGURE EC2 INSTANCES

REVIEW

Specify Mapper and Reducer functions to run within the Job Flow. The mapper and reducers may be either (i) class names referring to a mapper or reducer class in Hadoop or (ii) locations in Amazon S3. ([Click Here](#) for a list of available tools to help you upload and download files from Amazon S3.) The format for specifying a location in Amazon S3 is bucket_name/path_name. The location should point to an executable program, for example a python program. Extra arguments are passed to the Hadoop streaming program and can specify things such as additional files to be loaded into the distributed cache.

Input Location*:

The URL of the Amazon S3 Bucket that contains the input files.

Output Location*:

The URL of the Amazon S3 Bucket to store output files. Should be unique.

Mapper*:

The mapper Amazon s3 location or streaming command to execute.

Reducer*:

The reducer Amazon s3 location or streaming command to execute.

Extra Args:

[< Back](#)

Continue 

* Required field

Elastic MapReduce Workflow

Create a New Job FlowCancel

✓

✓

DEFINE JOB FLOW SPECIFY PARAMETERS **CONFIGURE EC2 INSTANCES** REVIEW

Enter the number and type of EC2 instances you'd like to run your job flow on.

Number of Instances*:

The number of EC2 instances to run in your Hadoop cluster.
If you wish to run more than 20 instances, please complete the [limit request form](#).

Type of Instance*:

The type of EC2 instances to run in your Hadoop cluster ([learn more about instance types](#)).

⌵ Show advanced options

< BackContinue* Required field

Elastic MapReduce Workflow



[Contact Us](#) | [Create an AWS Account](#)

[About AWS](#) | [Products](#) | [Solutions](#) | [Resources](#) | [Support](#) | [Your Account](#)

[Home](#) > [Resources](#) > [AWS Management Console](#) [BETA](#) > [Amazon Elastic MapReduce](#)

Welcome, Rad Lab | [Settings](#) | [Sign Out](#)

[Amazon EC2](#) | **[Amazon Elastic MapReduce](#)** | [Amazon CloudFront](#)

Your Elastic MapReduce Job Flows

Region: US-East [Create New Job Flow](#) [Terminate](#) [Show/Hide](#) [Refresh](#) [Help](#)

Viewing: All 1 to 1 of 1 Job Flows

Name	State	Creation Date	Elapsed Time	Normalized Instance Hours
My Job Flow	STARTING	2009-08-19 14:50 PDT	0 hours 0 minutes	0

1 Job Flow selected

Id: j-46JL0YQ7ZPH1

Name: My Job Flow

State: STARTING

Last State Change Reason: Starting instances

Availability Zone: us-east-1b

Creation Date: 2009-08-19 14:50 PDT

Start Date: -

End Date: -

Instance Count: 4

© 2008 - 2009, Amazon Web Services LLC or its affiliates. All right reserved. | [Feedback](#) | [Support](#) | [Privacy Policy](#) | [Terms of Use](#)

Takeaways #2

- ▶ MapReduce programming model hides the complexity of work distribution and fault tolerance
- ▶ Principal design philosophies:
 - **Make it scalable**, so you can add hardware easily
 - **Make it cheap**, lowering hardware, programming and admin costs
- ▶ MapReduce is not suitable for all problems, but when it works, it may save you quite a bit of time
- ▶ **Cloud computing** or **Cloudera** makes it straightforward to start using Hadoop at scale

Research@DBIS

- ▶ Hadoop Cluster with 10 machines & 30 TB Storage
- ▶ Distributed Processing of Semantic Data
 - Storing strategies for RDF Graphs in HDFS, HBase, Cassandra
 - Mapping SPARQL queries to PIG or directly MapReduce
 - Executing Path queries with PIG or directly MapReduce for investigating e.g. social networks

Resources

- ▶ Hadoop
 - <http://hadoop.apache.org/core/>
- ▶ Pig
 - <http://hadoop.apache.org/pig>
- ▶ Hive
 - <http://hadoop.apache.org/hive>
- ▶ Cloudera's Distribution
 - <http://www.cloudera.com/>
- ▶ Video tutorials
 - <http://www.cloudera.com/hadoop-training>
- ▶ Amazon Web Services
 - <http://aws.amazon.com/>

Resources (2)

- ▶ Hadoop: The Definitive Guide, Second Edition
 - Tom White
 - O'Reilly Media, 2010
- ▶ Data-Intensive Text Processing with MapReduce
 - Jimmy Lin, Chris Dyer, Graeme Hirst
 - Morgan and Claypool Publishers, 2010
- ▶ Cluster Computing and MapReduce Lecture Series
 - Google, 2007: Available on Youtube
- ▶ Verarbeiten großer Datenmengen mit Hadoop
 - Oliver Fischer
 - Heise Developer, 2010: Online

New MapReduce API

»» Additional slides

New API: Top Level Changes

- ▶ Methods can throw `InterruptedException` as well as `IOException`
- ▶ Configuration instead of `JobConf` Objekt
- ▶ Library classes are moved to `mapreduce.lib`
verschoben
 - `{input, map, output, partition, reduce}.*`

Mapper

▶ Map funktion

- `map (K1 key, V1 value, OutputCollector<K2,V2> output, Reporter reporter)` old API
- `map(K1 key, V1 value, Context context)` new API

▶ Close

- `Close()`
- `cleanup(Context context)`

▶ Output

- `Output.collect(K,V)`
- `Context.write(K,V)`

MapRunnable

- ▶ Using mapreduce.Mapper
 - `void run (RecordReader<K1,V1> input,
 OutputCollector<K2,V2> output,
 Reporter reporter)`
 - `void run (Context context)`

Reducer & Combiner

▶ Reduce funktion

- `void reduce (K2, Iterator<V2> values,
OutputCollector<K3,V3> output)`
- `void reduce (K2, Iterable<V2> values,
Context context)`

▶ Iteration

- `while (values.hasNext() {
V2 value = values.next(); ... }`
- `for (V2 value: values) { ... }`

Submitting Jobs

- ▶ `JobConf` + `JobClient` are replaced with `Job`
- ▶ Job Constructor
 - `job = new JobConf(conf, MyMapper.class)`
`job.setJobName(„job name“)`
 - `job = new Job(conf, „job name“)`
 - `job.setJarByClass(MyMapper.class)`

Submitting Jobs (2)

▶ Further Properties

- `Job` has `getConfiguration`
- `FileInputFormat` in `mapreduce.lib.input`
- `FileOutputFormat` in `mapreduce.lib.output`

▶ Ausführung

- `JobClient.runJob(job)`
- `System.exit(job.waitForCompletion(true)?0:1)`