Distributed Processing with Computer Clusters I

Advanced Databases and Information Systems

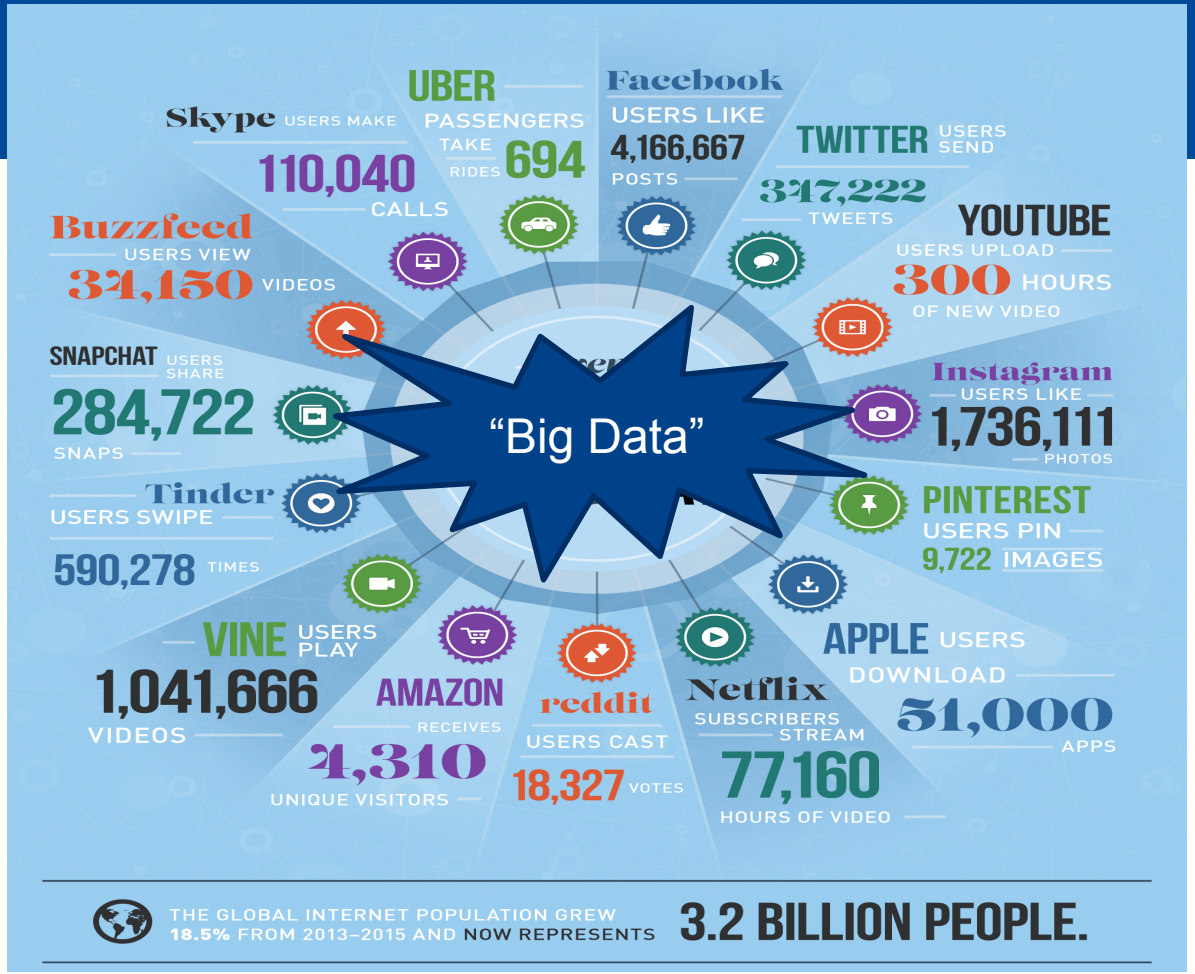
SS18

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Prof. Dr. Georg Lausen

Patrick Philipp



https://www.domo.com/blog/2015/08/data-never-sleeps-3-0/

Examples of Big Data

UNI FREIBURG

- **Google** processes 3.5 billion requests per day, storing 10 Exabytes of data.
- **Amazon** hosts the most servers of any company (est. 1,400,000 servers), having Google and Microsoft close behind. Amazon Web Services (AWS) are used by 60,000 companies and field more than 650,000 requests every second. It is estimated that 1/3 of all Internet users use a website hosted on AWS daily, and that 1% of all Internet traffic goes through Amazon.
- **Facebook** collects 500 terabytes of data daily, including 2.5 billion pieces of content, 2.7 billion likes and 300 million photos.
- 90% of all the data in the world was produced in the last 2 years.
- It is estimated that 40 Zettabytes (40,000 Exabytes) of data will be created by 2020.

See http://cloudtweaks.com/wp-content/uploads/2015/03/bigdata-facts.png

Publicly Available Big Data Sets

UNI FREIBURG

- Common crawl
  https://commoncrawl.org/
- Wikipedia edit history
- AWS Public Datasets
  https://aws.amazon.com/de/datasets/
- Google BigQuery Public Datasets
  https://cloud.google.com/bigquery/public-data/
- ...

What is Big Data?

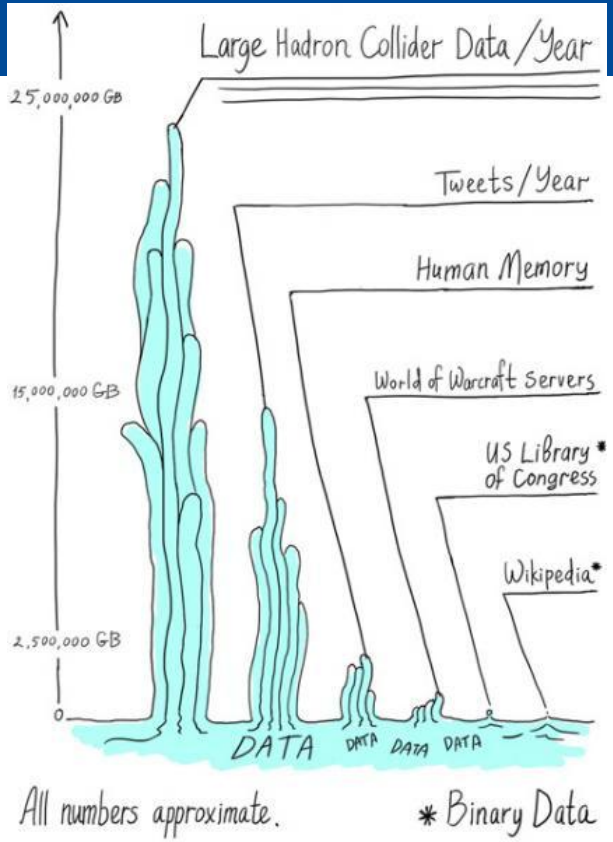
- ‘Big data’ is similar to ‘Small data’, but bigger
  - and often less structured
  - often characterized by the 3 (or 4) V‘s:
    - Volume
    - Variety
    - Velocity
    - (Veracity)

What is Big Data? (2)

UNI FREIBURG

- … but it requires different approaches:
  - techniques, tools, architectures
- … with the aim to solve new problems
  - or old problems in a better way.

What is Big Data and What not?



from: Symmetry Magazine

Big Data related tools

http://www.bigdata-startups.com/open-source-tools/

Distributed processing

UNI FREIBURG

- Distributed processing of Big Data requires non-standard programming models
  - beyond single machines or traditional parallel programming models (like Message Passing Interface, MPI)
  - the aim is to simplify complex programming tasks

- One of the most popular frameworks is **MapReduce**
  - suitable for commodity hardware to reduce costs

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

Big Ideas behind MapReduce

UNI FREIBURG

- **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

DFS & MapReduce

UNI FREIBURG

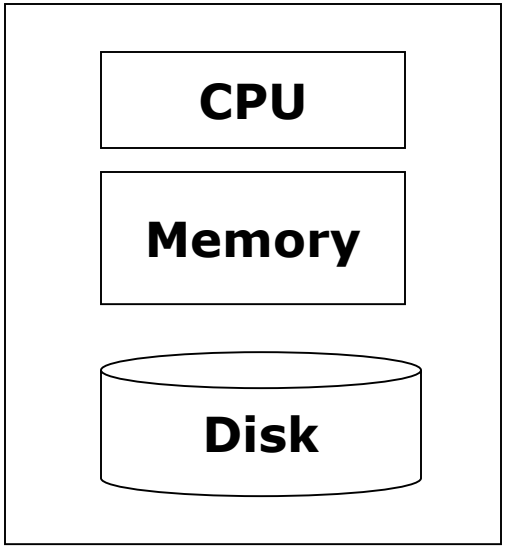
- **Distributed File System (Google: GFS, Hadoop: HDFS)**
  - Data is split into equally sized blocks and stored distributed
  - Fault tolerance by replication
  - Very large files / write-once, read-many pattern

- **MapReduce**
  - Algorithms expressed as a sequence of Map() and Reduce() functions
  - Automatically parallelized by the framework

- **Advantages**
  - Partitioning + distribution of data
  - Parallelization and assigning of task
  - Scalability, fault-tolerance, scheduling,…

That all is done automatically

Single-node architecture



Machine Learning, Statistics

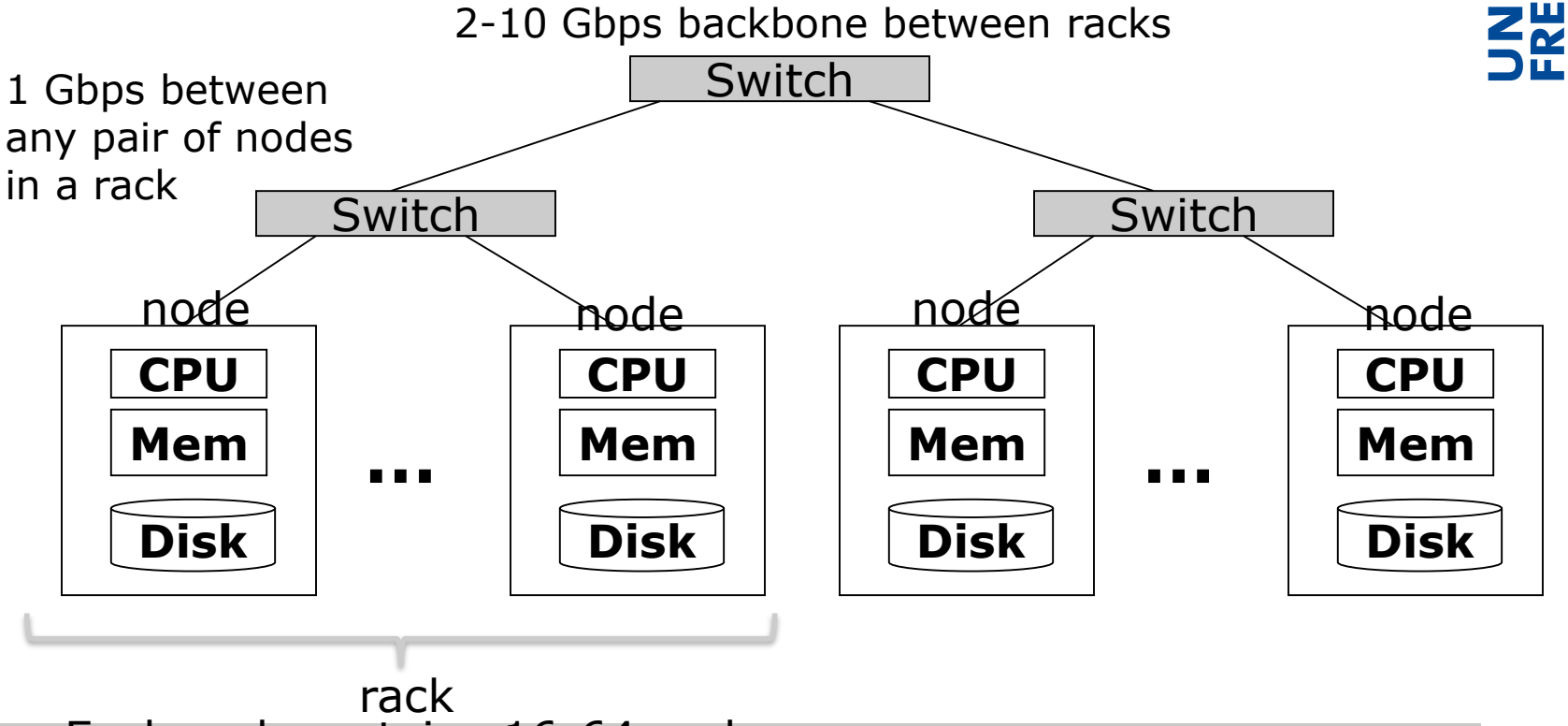
“Classical” Data Mining

Commodity Clusters

UNI FREIBURG

- Data mining on a single server not enough
- Standard architecture emerging:
  - *Cluster of commodity Linux nodes*
  - *Gigabit ethernet interconnect*
- How to organize computations on this architecture?
  - Mask issues such as hardware failure

Cluster Architecture



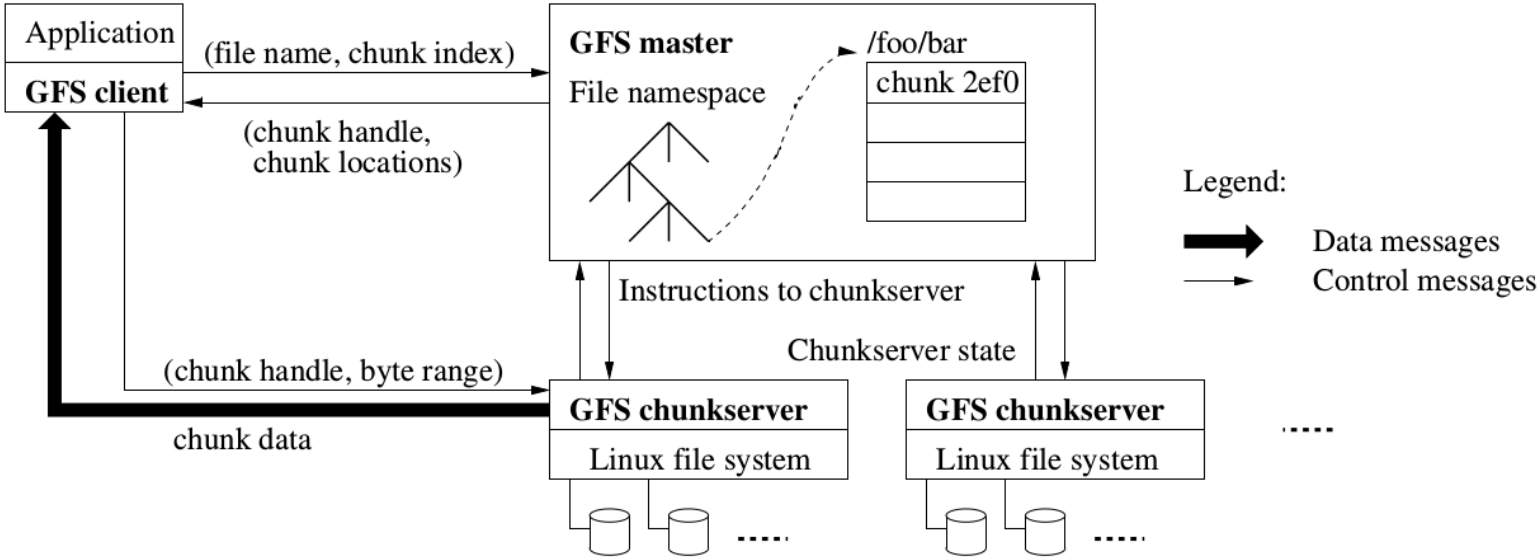
Each rack contains 16-64 nodes

Google File System (GFS)

UNI FREIBURG

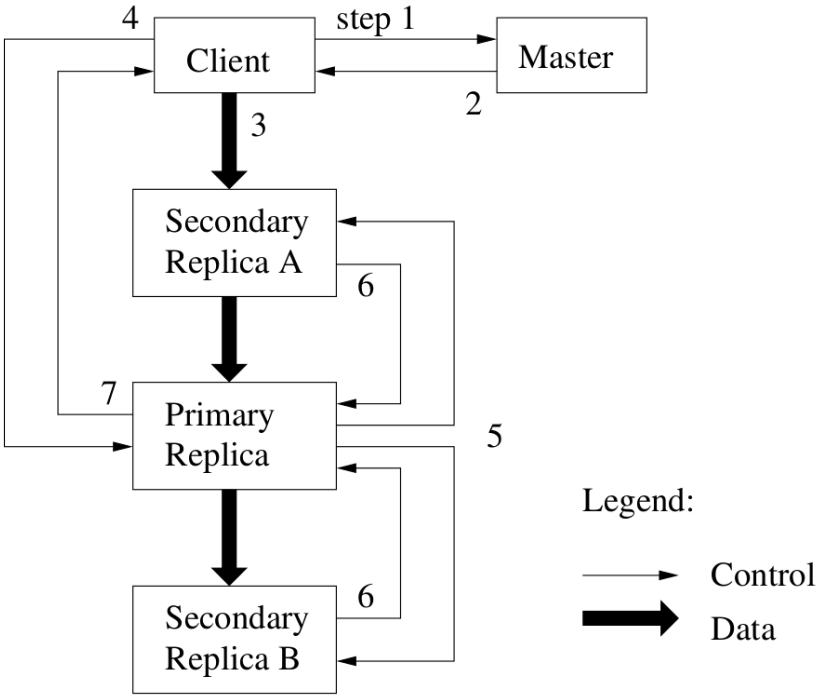
- very first proposal of a DFS
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." SOSP‘2003.
- Available online: http://blough.ece.gatech.edu/6102/presentations/gfs-sosp2003.pdf
- designed for Web search / Matrix-vector multiplication

GFS Architecture



from: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." SOSP'2003.

Writing Workflow in GFS



from: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." SOSP'2003.

DFS Usage Pattern

UNI FREIBURG

- Typical usage pattern for a DFS (and MapReduce)
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Basic Components of a DFS

UNI FREIBURG

1. Chunk Servers
   - File is split into contiguous chunks
   - Typically each chunk is nowadays between 64 and 256MB
   - Each chunk replicated (usually 2x or 3x)
   - Try to keep replicas in different racks
2. Master node
   - a.k.a. Name Nodes in HDFS
   - Stores metadata (about files)
   - Might be replicated

Basic Components of a DFS

UNI FREIBURG

3. Client library for file access
   - Talks to master to find chunk servers
   - Connects directly to chunk servers to access data

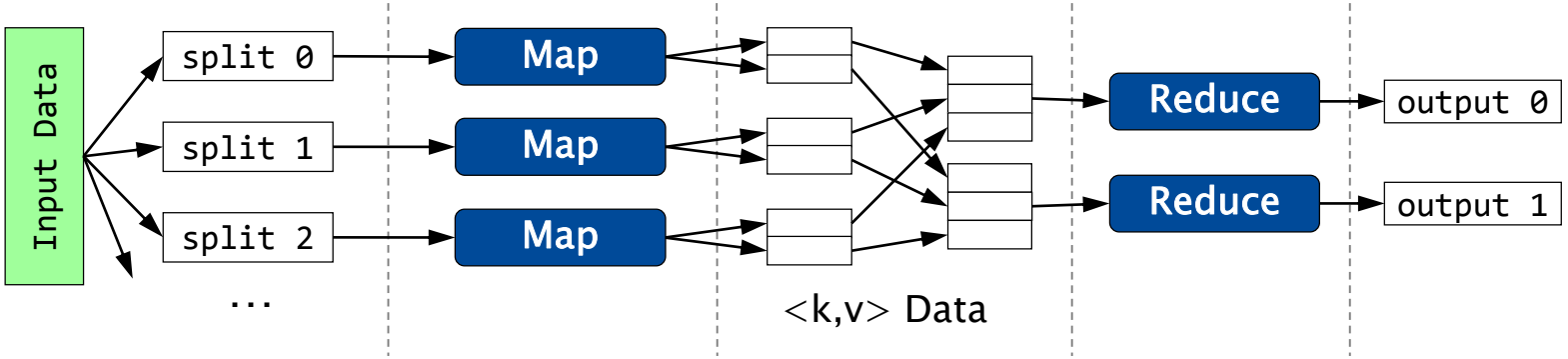
MapReduce

- The key idea of the MapReduce approach:

  A target problem needs to be parallelizable

  1. First, the problem gets split into a set of smaller problems (Map step)
  2. Next, smaller problems are solved in a parallel way
  3. Finally, a set of solutions to the smaller problems get synthesized into a solution of the original problem (Reduce step)

MapReduce overview



MapReduce example:
Counting words in documents

Google Maps charts new territory into businesses

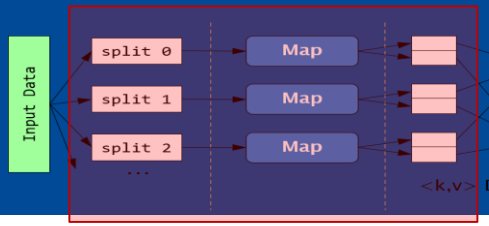
Google selling new tools for businesses to build their own maps

Google promises consumer experience for businesses with Maps Engine Pro

Google is trying to get its Maps service used by more businesses

Google	4
Maps	4
Businesses	4
Engine	1
Charts	1
Territory	1
Tools	1
…	

Map task



Google Maps charts new territory into businesses

Google selling new tools for businesses to build their own maps

Map

Google	2
Charts	1
Maps	2
Territory	1
…	

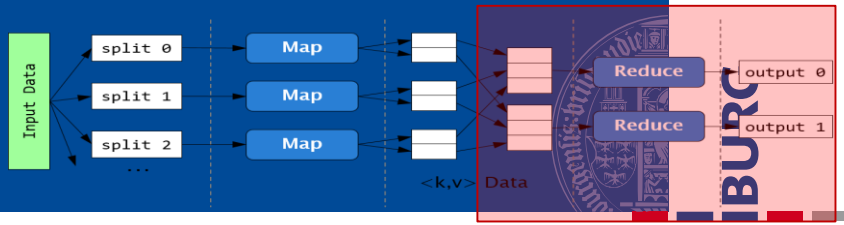
Google promises consumer experience for businesses with Maps Engine Pro

Google is trying to get its Maps service used by more businesses

Map

Google	2
Businesses	2
Maps	2
Service	1
…	

Reduce task



Google	2
Google	2
Maps	2
Maps	2
…	

Reduce

Google	4
Maps	4
…	
…	

Aggregation

Businesses	2
Businesses	2
Charts	1
Territory	1
…	

Reduce

Businesses	4
Charts	1
Territory	1
…	
…	

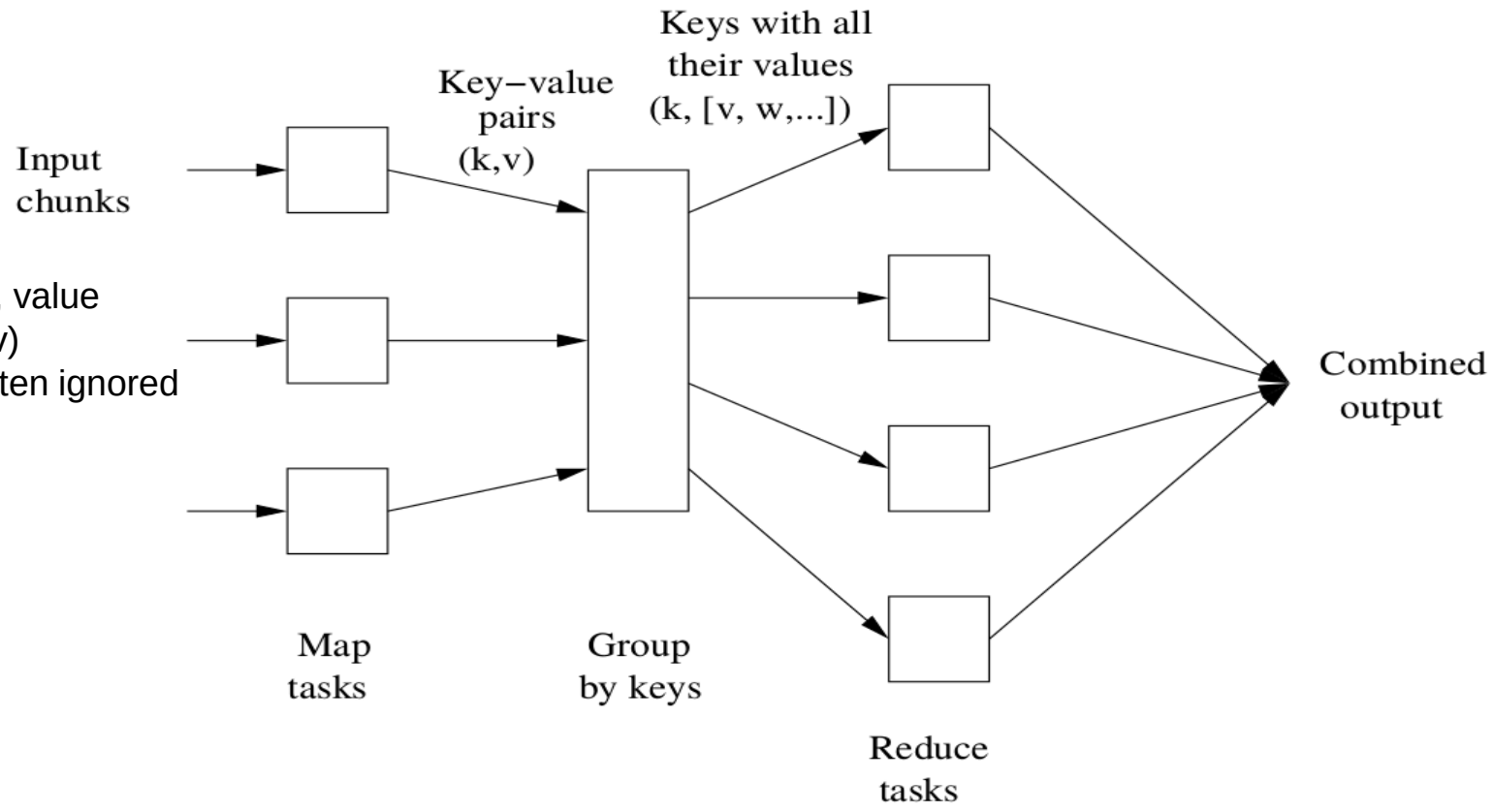
MapReduce: General idea

UNI FREIBURG

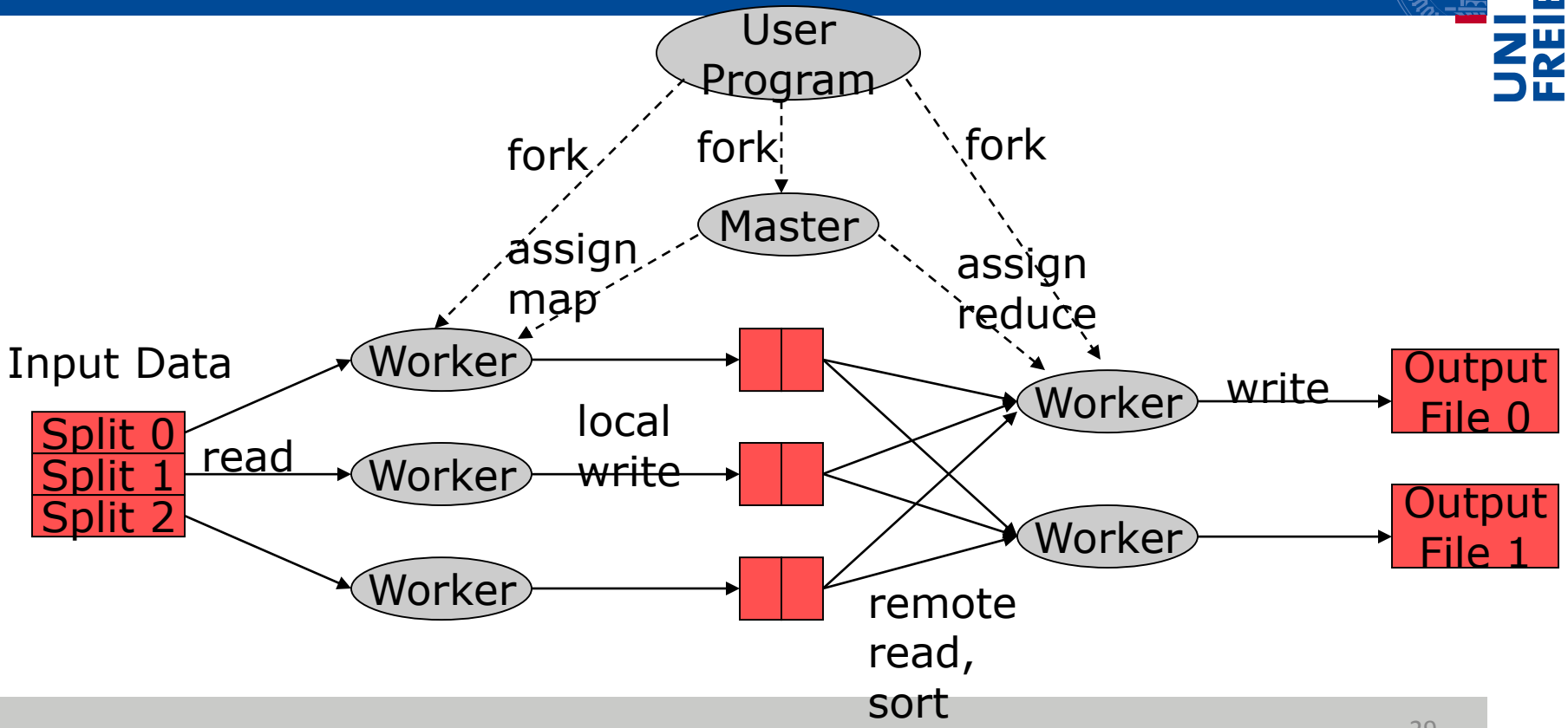
- Two functions: **map()** and **reduce()**
- Input/Output: always key-value-pairs

=> allow composition of MapReduce processes

Distributed Execution Overview



Distributed Execution in more Detail



Data Flow

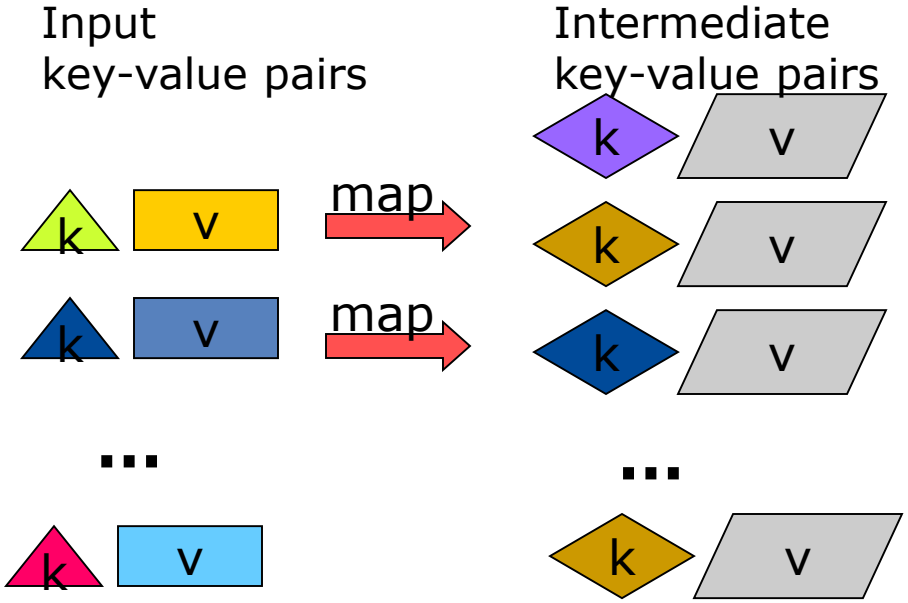
UNI FREIBURG

- Input/final output are stored on a distributed file system
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on local file system of map- and reduce-workers
- Output is often input to another MapReduce task

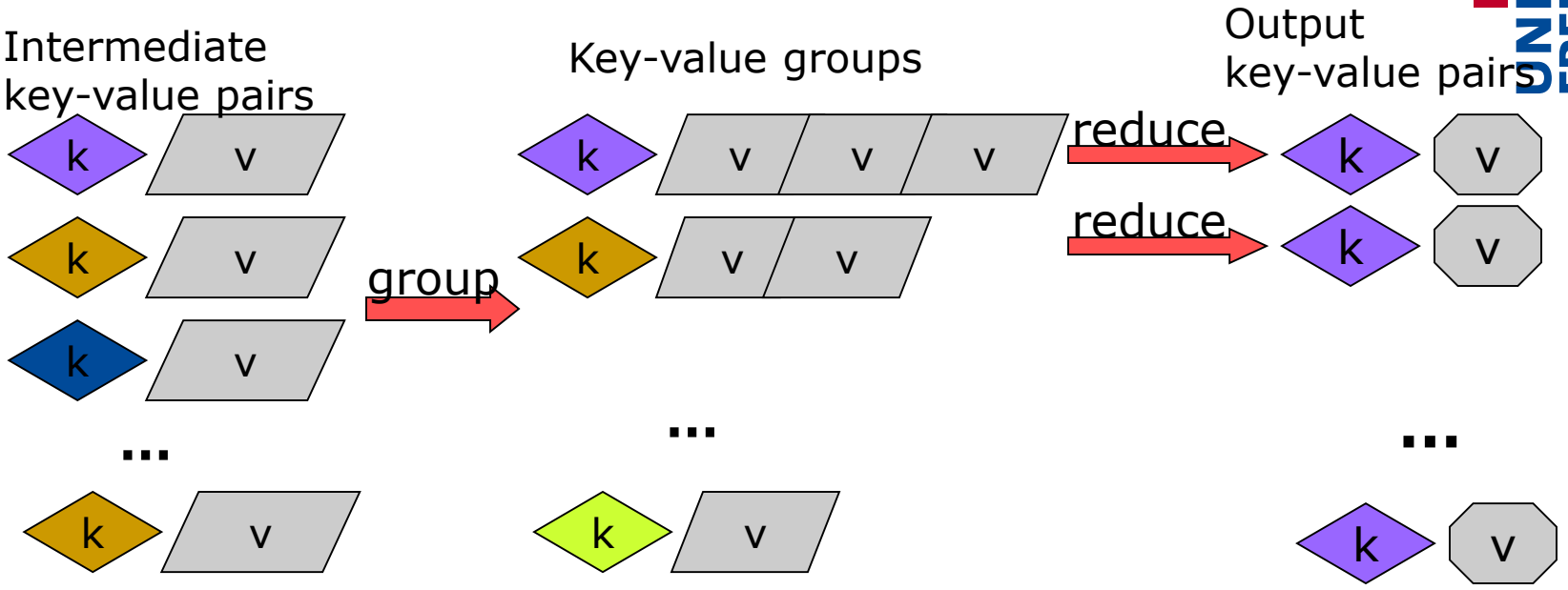
MapReduce: General procedure

- Input: a set of key/value pairs
- User supplies two functions:
  map(k,v) → list(k1,v1)
  reduce(k1, list(v1)) → v2
- (k1,v1) is an intermediate key/value pair
- Output is the set of (k1,v2) pairs

MapReduce: The Map Step



MapReduce: The Reduce Step



Example: Word Count

UNI FREIBURG

- We have a large file of words, one word to a line
- Count the number of times each distinct word appears in the file
- Sample application:
  - analyze web server logs to find popular URLs
  - analyze Web page contents
- Great: This is naturally parallelizable

Word Count using MapReduce

```
map(key, value):
// key: document name; value: text of document
    for each word w in value:
        emit(w, 1)

reduce(key, values):
// key: a word; value: an iterator over counts
        result = 0
        for each count v in values:
            result += v
        emit(result)
```

Note: Hadoop versions

UNI FREIBURG

- Two versions of the APIs
  - The **old API** is in the org.apache.hadoop.**mapred** package
  - The **new API** is in theorg.apache.hadoop.**mapreduce** package
- Two versions of the runtime
  - MRv1 (MapReduce)
  - **MRv2 (MapReduce 2, YARN)**
- Never mix the old and new API

Word Count using MapReduce

UNI FREIBURG

```
import org.apache.hadoop.*;

public class WordCountDriver {

  public static void main(String[] args) {
    JobClient client = new JobClient();
    JobConf conf = new JobConf(WordCountDriver.class);

    // specify output types
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    // specify input and output dirs
    FileInputFormat.addInputPath(conf, new Path("input"));
    FileOutputFormat.setOutputPath(conf, new Path("output"));
```

Word Count using MapReduce

UNI FREIBURG

```
    // specify a mapper
    conf.setMapperClass(WordCountMapper.class);

    // specify a reducer
    conf.setReducerClass(WordCountReducer.class);
    conf.setCombinerClass(WordCountReducer.class);

    client.setConf(conf);
    try {
      JobClient.runJob(conf);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Word Count using MapReduce

UNI FREIBURG

```
import org.apache.hadoop.*;
public class WordCountMapper extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable>
{
        private final IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException
        {
                String line = value.toString();
                StringTokenizer itr = new StringTokenizer(line.toLowerCase());
                while(itr.hasMoreTokens()) {
                  word.set(itr.nextToken());
                  output.collect(word, one);
                }
        }
}
```

Word Count using MapReduce

UNI FREIBURG

```
import org.apache.hadoop.mapred.*;

public class WordCountReducer extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator values,
      OutputCollector output, Reporter reporter) throws IOException {

    int sum = 0;
    while (values.hasNext()) {
      IntWritable value = (IntWritable) values.next();
      sum += value.get(); // process value
    }

    output.collect(key, new IntWritable(sum));
  }
}
```

Coordination

UNI FREIBURG

- **Master** as “central unit” for coordination
  - Task status: (idle, in-progress, completed)
  - Idle tasks get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its R intermediate files, *one for each reducer*
  - Master pushes this information to reducers
- Master checks periodically if workers have failures

Failures

UNI FREIBURG

- Different kinds of failure possible:
- *Map worker* failure
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- *Reduce worker* failure
  - Only in-progress tasks are reset to idle
- *Master* failure
  - MapReduce task is aborted and client is notified

How many Map and Reduce jobs?

UNI FREIBURG

- M map tasks, R reduce tasks
- Rule of thumb:
  - Make M and R much larger than the number of nodes in cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds recovery from worker failure
- Usually R is smaller than M, because output is spread across R files

Combiners

- Often a map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in Word Count
- Can save network time by pre-aggregating at mapper
  - combine(k1, list(v1)) → v2
  - Usually same as reduce function
  - Can make a huge difference! **Optimization**
- Works only if reduce function is commutative and associative
- still grouping and aggregation in reduce() necessary

Partition Function

UNI FREIBURG

- Inputs to map tasks are created by contiguous splits of input file
- For reduce, we need to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function e.g., hash(key) mod R
- Sometimes useful to override
  - E.g., *hash(hostname(URL)) mod R* ensures URLs from a host end up in the same output file

Implementations of MapReduce

UNI FREIBURG

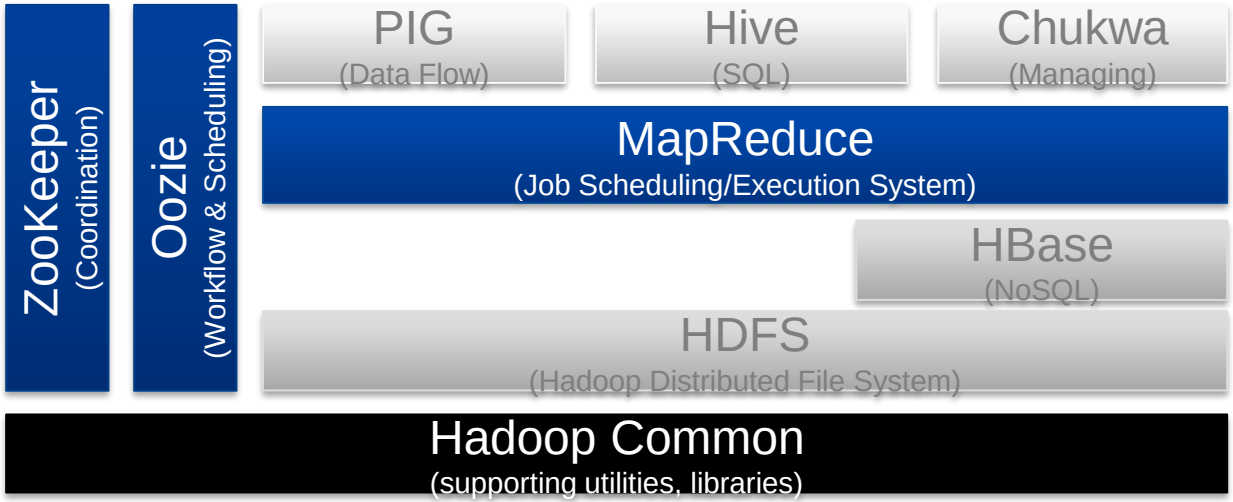
- Google
  - Not available outside Google
- Apache Hadoop
  - Well-known Open-Source implementation of Google's MapReduce & Google File System (GFS)
  - Enriched by many subprojects
  - Cloudera's Distribution: VMWare images, tutorials, patches
  - Used by Yahoo, Facebook, Amazon, IBM, Last.fm, EBay …
- Hadoop and other MapReduce implementations are available on EC2

Implementations (2)

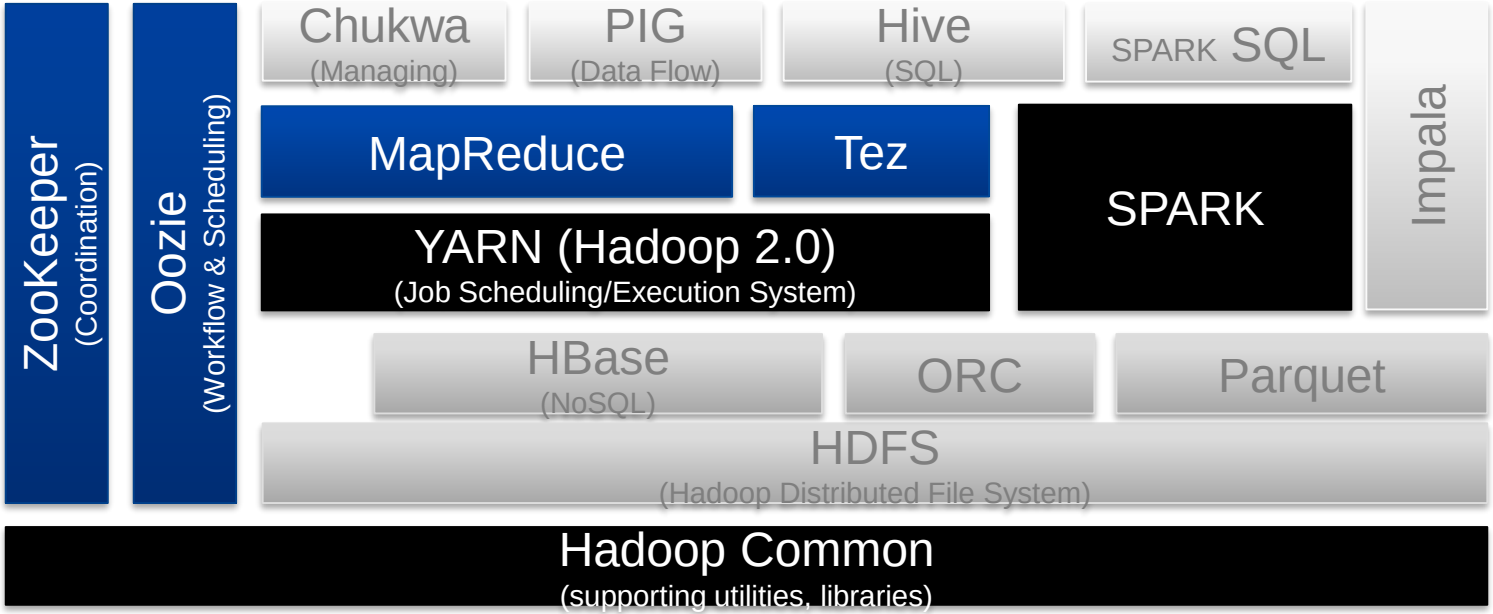
UNI FREIBURG

- Aster Data
  - Cluster-optimized SQL Database that also implements MapReduce
- MongoDB
- CouchDB
  - having an architecture that "completely embraces the Web”
  - ACID Semantics
  - Built for Offline
  - Distributed Architecture with Replication
  - Document Storage
  - Eventual Consistency
  - Map/Reduce Views and Indexes
  - HTTP API

Hadoop Ecosystem 2011



Hadoop Ecosystem 2015



and many more …

Further Hadoop Components

UNI FREIBURG

a. Pig
b. Hive
c. HBase

Pig (Latin): Why?

UNI FREIBURG

- 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”?

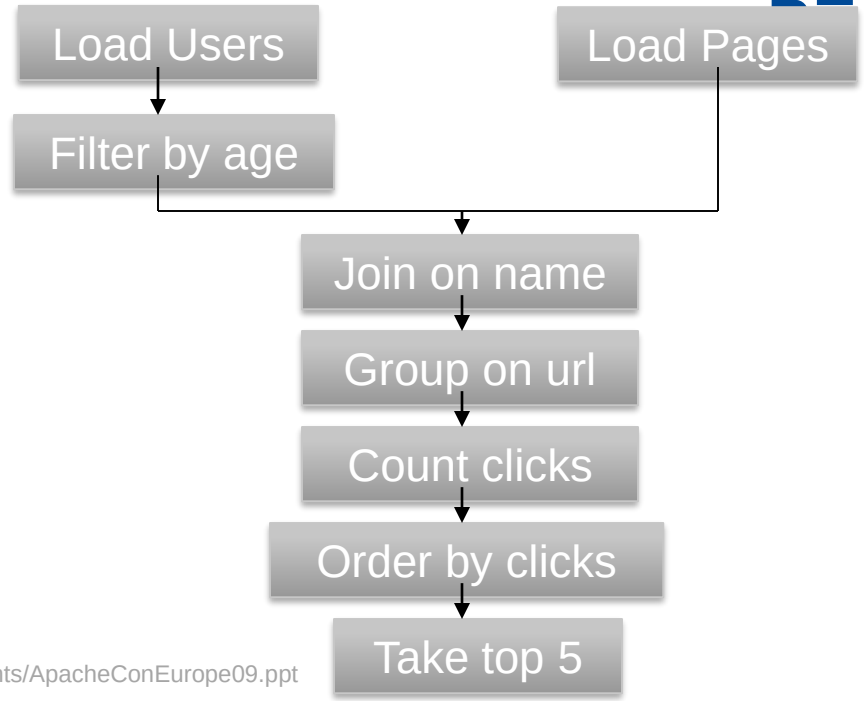
Pig (Latin)

- Apache Pig: a high-level platform for programs that run on Apache Hadoop
- Pig Latin: Language for this platform
- Started at Yahoo! Research in 2006, since 2007 Apache project
- Runs about 30% of Yahoo!'s jobs*
- Features:
  - Expresses sequences of MapReduce jobs
  - Extract-transform-load (ETL) data pipelines
  - 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

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

Example Task

- 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



Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

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.a  pache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
imp ort org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobC      ontrol;
import org.apache.hadoop.mapred.lib.IdentityMapper;

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

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

        public void map(LongWritable k, Text val,
                        OutputCollector<Text, Text> oc,
                        Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(    firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so w       e know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.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> oc,
                           Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.to   String();
                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 s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputColle    ctor<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', first       Comma);
            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 instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text ke   y,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            wh  ile (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        i mplements Mapper<WritableComparable, Writable, LongWritable,
Text> {

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

            // Only output the first 100 records
            while (count    < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.se  tJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/  user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
                new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.s  etJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.add     InputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
                new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(     MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMap     per.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.se     tOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRE     xample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFi     leOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputF       ormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        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();
    }
}
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

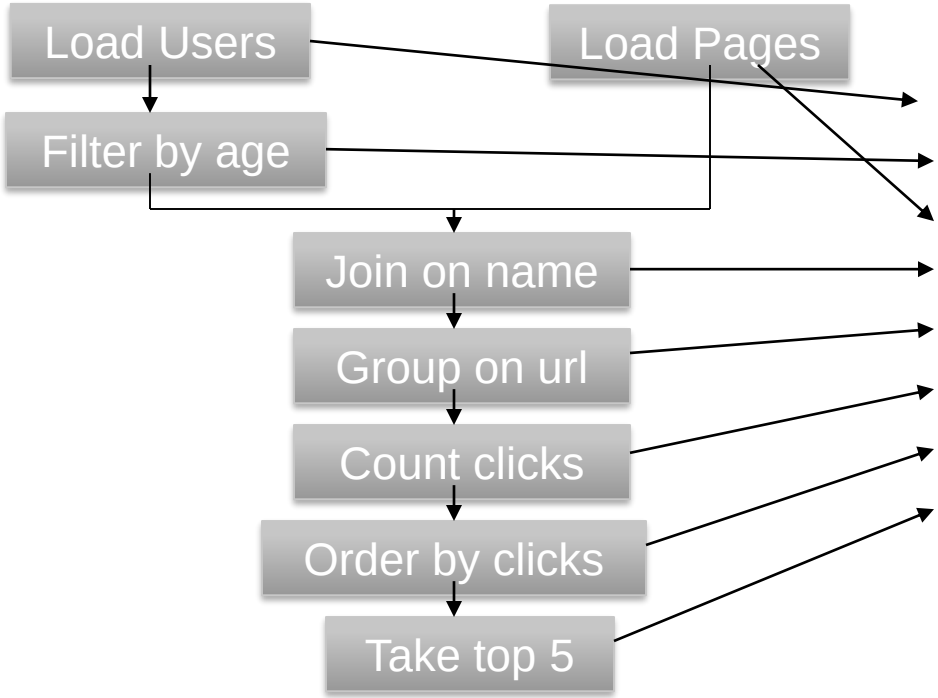
In Pig Latin

```
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’;
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

Ease of Translation




```
Users = load …
Filtered = filter …
Pages = load …
Joined = join …
Grouped = group …
Summed = … count()…
Sorted = order …
Top5 = limit …
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

Hive

- Developed at Facebook
- Used for majority of Facebook jobs
- Written in Java, current version 2.1
- 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

UNI FREIBURG

- 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 *BigTable* (Google)
- Data is stored “column-oriented”
- Distributed over many servers
- Layered over HDFS
- Strong consistency (CAP Theorem)
- Scalable up to billions of rows and millions of columns

See “Bigtable: A Distributed Storage System for Structured Data”, 2006.

And many other Hadoop systems and tools…

UNI FREIBURG

- **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

Resources

UNI FREIBURG

- Cloudera's Hadoop Distribution → recommended
  - http://www.cloudera.com/
  - Virtual Machine Image (VMWare)
- Video tutorials
  - http://www.cloudera.com/training/library/hadoop-essentials.html
  - http://www.cloudera.com/training/library/tutorials.html
  - http://www.cloudera.com/training/library/hadoop-ecosystem.html
- Apache Hadoop
  - http://hadoop.apache.org/core/
  - http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- Project website:
  - MapReduce introduction slides
  - Eclipse projects with some MapReduce examples

Resources (2)

- MapReduce: Simplified data processing on large clusters
  - Dean, J. and Ghemawat, S., ACM 2008
- The Google file system
  - Ghemawat, S. and Gobioff, H. and Leung, S.T., 2003
- **Hadoop: The Definitive Guide, 4th Edition → recommended**
  - Tom White
  - O'Reilly Media, 2015
- 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

Acknowledgement

- Slides are partially from
  - Prof. Jeffrey D. Ullman & Dr. Jure Leskovec.
  - Martin Przyjaciel-Zablocki & Alexander Schätzle.