Übung Datenbanksysteme II
Web-Scale Data Management

Thorsten Papenbrock
MapReduce: Introduction

- MapReduce ...
  - is a **paradigm** derived from functional programming.
  - is implemented as **framework**.
  - operates primarily **data-parallel** (not task-parallel).
  - **scales-out** on multiple nodes of a cluster.
  - uses the Hadoop distributed filesystem.
  - is designed for **Big Data Analytics**:  
    - Log-files
    - Weather-statistics
    - Sensor-data
    - ...
  - “Competitors“:

"Spark"  "Stratosphere"
Who is using Hadoop?

- Yahoo!
  - Biggest cluster: 2000 nodes, used to support research for Ad Systems and Web Search.

- Amazon
  - Process millions of sessions daily for analytics, using both the Java and streaming APIs. Clusters vary from 1 to 100 nodes.

- Facebook
  - Use Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics. 600 machine cluster.

- ...
MapReduce: Introduction

A programming model
Large-scale distributed data processing
Simple but restricted
Parallel programming
Extensible
Inspired in Functional programming but not equivalent
Think in recursive solutions (sometimes)

Map
Given a list create a new output list applying a function to each element
Reduce
Given a list iterates creating an aggregated value

Function Patterns
Applications
Trends
Success Stories
Features

MapReduce

Design Patterns

Relational-based
Iterative Message Passing

Data Organization
Filtering
Summarization

Other patterns

Google
Avalanche Hadoop

Others

Analytics
Query
Others

Summarization and statistics
Sorting and merging
Frequency distribution
SQL-based queries
Generation of graphics

Other patterns

Input/Output
Query
Others

Index and Search
Filtering
Classification
Recommendations
Collaborative filtering
Clustering

Data Cleaning
Top N records
Bloom Filtering
Distinct

Filtering
Sorting and merging
Frequency distribution
SQL-based queries
Generation of graphics

Other patterns

Join, projection, selection, union, etc.
cartesian product
graph processing
PageRank

Iterative Message Passing

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

Others

MapReduce: Introduction

http://dme.rwth-aachen.de/de/research/projects/mapreduce
MapReduce: Introduction

http://mohamednabeel.blogspot.de/2011/03/starting-sub-sandwich-business.html
MapReduce: Phases

- **map-task:**
  - record reader
  - mapper
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - reducer
  - output formatter
MapReduce: Phases

- **map-task:**
  - record reader
  - mapper
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - reducer
  - output formater

- **Input:** <data entry> (row/split/item)
- **Output:** <key, record>

- “key” is usually positional information
- “record” represents a raw data record

- Translates a given input into records
- Parses data into records but not the records itself
MapReduce: Phases

- **map-task:**
  - record reader
  - mapper
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - reducer
  - output formatter

- **Input:** <key, record>
- **Output:** <key*, value>

  “key*” is a problem-specific key
  - e.g. the word for the word-count-task
  “value” is a problem-specific value
  - e.g. “1” for the occurrence of a word

- Executes user defined code that starts solving the given task
- Defines the grouping of the data

- A single mapper can emit multiple <key*, value> output pairs for a single <key, record> input pair
MapReduce: Phases

- **map-task:**
  - record reader
  - mapper
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - reducer
  - output formater

- Input: <key*, values>
- Output: <key*, value>

  - “key*“ is a problem-specific key
    - e.g. the word for the word-count-task
  - “value“ is a problem-specific value
    - e.g. “1“ for the occurrence of a word

- Executes user defined code that merges a set of values
- Pre-aggregates values to reduce network traffic
- Is an optional, localized reducer
MapReduce: Phases

- **map-task:**
  - record reader
  - mapper
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - reducer
  - output formatter

- **Input:** `<key*, value>`
- **Output:** `<key*, value> + reducer`

  “reducer“ is the reducer number that should handle this key/value pair; reducer might be located on other compute nodes.

  Distributes the keyspace randomly to the reducers.

  Calculates the reducer by e.g. `key*.hashCode() % (number of reducers)`
MapReduce: Phases

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner

- reduce-task:
  - shuffle and sort
  - reducer
  - output formatter

- Input: <key*, value> + reducer
- Output: <key*, value> + reducer

Downloads the <key*, value> data to the local machines that run the corresponding reducers
MapReduce: Phases

- **map-task:**
  - record reader
  - mapper
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - reducer
  - output formatter

- **Input:** <key*, values>
- **Output:** <key*, result>

"result" is the solution/answer for the given "key*"

Executes user defined code that merges a set of values
Calculates the final solution/answer to the problem statement for the given key
MapReduce: Phases

- **map-task:**
  - record reader
  - mapper
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - reducer
  - output formater

- **Input:** <key*, result>
- **Output:** <key*, result>

  - Writes the key/result pairs to disk
  - Formats the final result and writes it record-wise to disk
MapReduce: Phases

- **map-task:**
  - record reader
  - **mapper**
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - **reducer**
  - output formatter

- **useful to increase the performance**
- **helpful to build a sorting algorithm**
- **basic building blocks with user defined code**
MapReduce:
Example 1: Distinct

- **map-task:**
  - record reader
  - **mapper**
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - **reducer**
  - output formatter

**Input:**
- A relational table instance
  \( \text{Car}(\text{name}, \text{vendor}, \text{color}, \text{speed}, \text{price}) \)

**Output:**
- A distinct list of all **vendors**

map (key, record) {
  emit (record.vendor, null);
}
reduce (key, values) {
  write (key);
}
MapReduce: Example 2: Index-Generation

- **map-task:**
  - record reader
  - **mapper**
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - **reducer**
  - output formatter

**Input:**
- A relational table instance
  \[\text{Car}(\text{name, vendor, color, speed, price})\]

**Output:**
- An index on \text{Car.vendor}

```java
map (key, record) {
  emit (record.vendor, key);
}
```

```java
reduce (key, values) {
  String refs = concat(values);
  write (key, refs);
}
```
MapReduce:
Example 3: Join

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner

- reduce-task:
  - shuffle and sort
  - reducer
  - output formater

- Input:
  - Two relational table instances
    - Car(name, vendor, color, speed, price)
    - Plane(id, weight, length, speed, seats)

- Output:
  - All pairs of cars and planes with the same speed
MapReduce: Example 3: Join

- **map-task:**
  - record reader
  - **mapper**
  - combiner
  - partitioner

- **reduce-task:**
  - shuffle and sort
  - **reducer**
  - output formatter

```java
Car(name, vendor, color, speed, price) Plane(id, weight, length, speed, seats)
```

```java
map (key, record) {
    emit (record.speed,
        {'table' => table(record),
        'record' => record});
}
```

```java
reduce (key, values) {
    cars = valuesWhere('table', 'car');
    planes = valuesWhere('table', 'plane');
    for (car : cars)
        for (plane : planes)
            write (car.record, plane.record);
}
```