Big Data Management & Analytics

EXERCISE 3

16th of November 2015

Sabrina Friedl
LMU Munich

1. Revision of Lecture

PARALLEL COMPUTING, MAPREDUCE

Parallel Computing Architectures

- Required to analyse large amounts of data
- Organisation of distributed file systems
  - Replicas of files on different nodes
  - Master node with directory of file copies
  - Examples: Google File System, Hadoop DFS
- Goals
  - Fault tolerance
  - Parallel execution of tasks

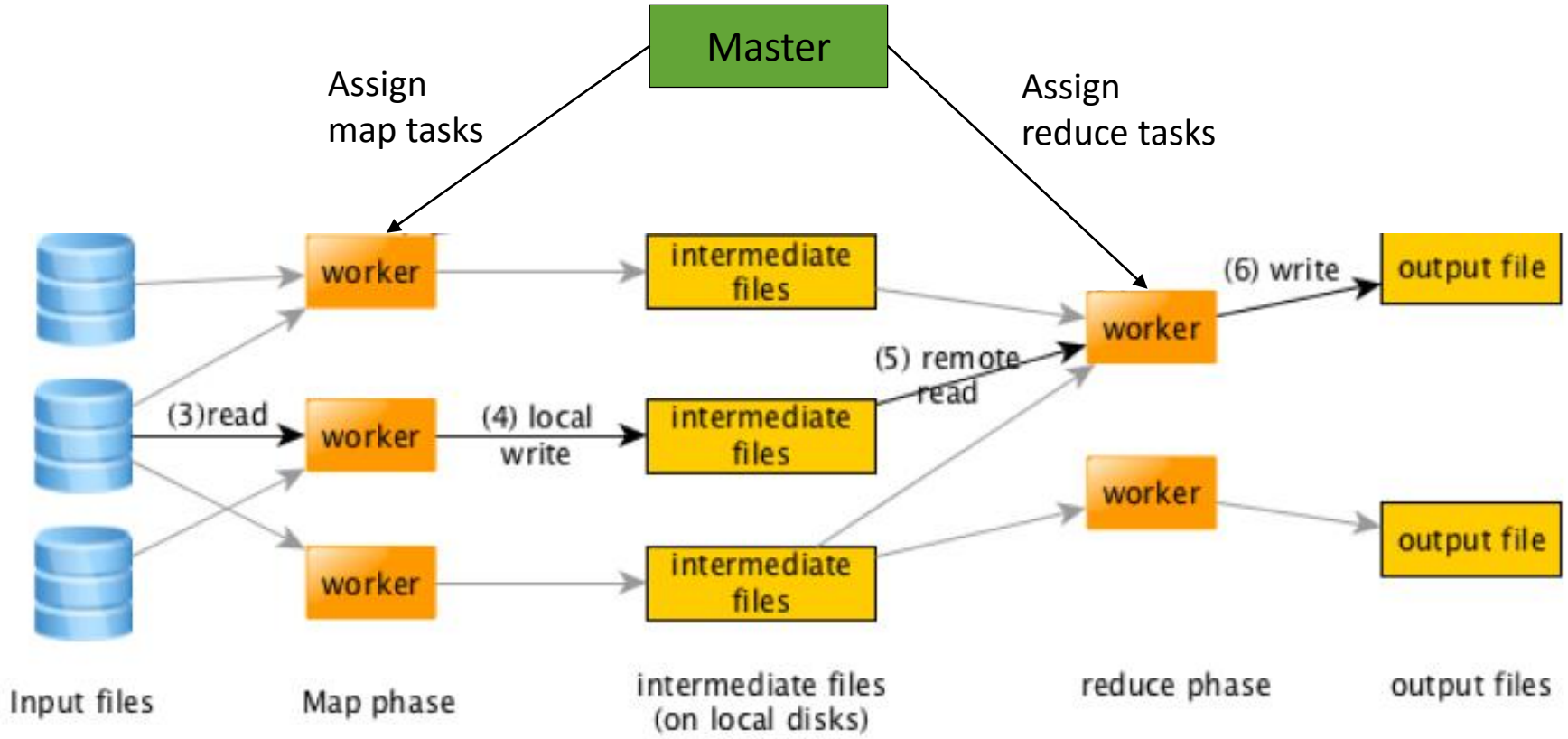
Racks of servers (and switches at the top), at Google's Mayes County, Oklahoma data center [extremetech.com]

MapReduce - Motivation

MapReduce: Programming model for parallel processing Big Data on clusters

- Stores data that can be processed together close to each other/close to worker (Data Locality)
- Handles data flow, parallelization and coordination of tasks automatically
- Copes with failures and stragglers

MapReduce – Processing (High Level)



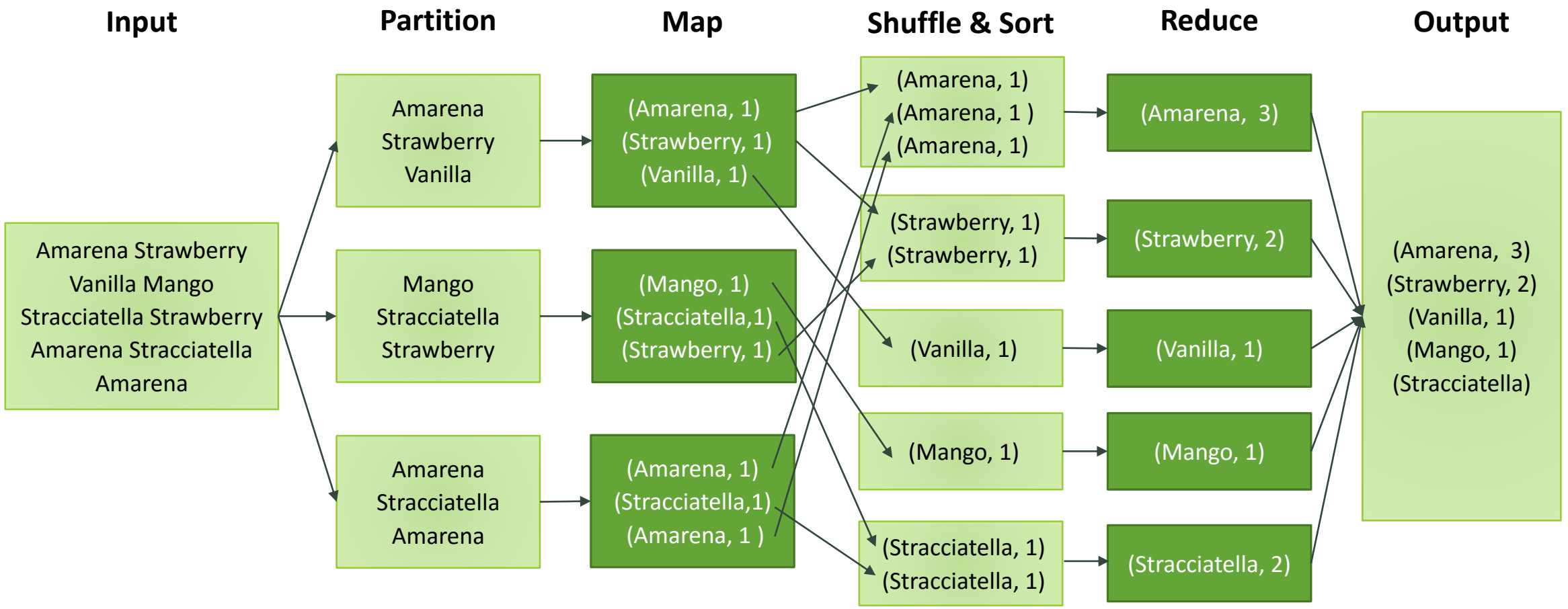
MapReduce – Programming Model

Transform set of input key-value pairs into set of output key-value pairs

- **Step 1**: map(k1, v1) -> list(k2, v2)
- **Step 2**: sort by k2 -> list(k2, list(v2))
- **Step 3**: reduce(k2, list(v2)) -> list(k3, v3)

-> Programmer specifies map() and reduce() function

MapReduce – Word Count



MapReduce – Matrix Multiplication

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} \quad A \cdot B = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{pmatrix}$$

Can be written as $A = (I, J, V),\ B = (J, K, W)$ where $[0] = row,\ [1] = column$ and $[2] = values$

Steps

- 1. Map $(i, j, a_{ij}) \longrightarrow (j, (A, i, a_{ij}))$ $(j, k, b_{jk}) \longrightarrow (j, (B, k, b_{jk}))$
- 2. Join $(j, (A, i, a_{ij})) \bowtie (j, (B, k, b_{jk})) \longrightarrow (j, [(A, i, a_{ij}), (B, k, b_{jk})])$
- 3. Map $(j, [(A, i, a_{ij}), (B, k, b_{jk})]) \longrightarrow ((i, k), (a_{ij}b_{jk}))$
- 4. ReduceByKey $((i, k), [(a_{ij}b_{jk})]) \longrightarrow ((i, k), \sum(a_{ij}b_{jk}))$

2. Spark and PySpark

WORKING WITH PYSPARK

Apache Spark™

Open source framework for cluster computing

- Cluster managers that Spark runs on:

  Hadoop YARN, Apache Mesos, standalone

- Distributed Storage Systems that can be used:

  Hadoop Distributed File System, Cassandra, Hbase, Hive

Homepage: http://spark.apache.org/docs/latest/index.html

PySpark - Usage

Spark Python API

- Spark shell: $./bin/spark-shell (‘\‘ for windows)
- PySpark shell: $./bin/pyspark
- Use in Python program:

```
from pyspark import SparkConf, SparkContext
sc = SparkContext('local')
```

Programming Guide: http://spark.apache.org/docs/latest/programming-guide.html#overview

Quick Start Guide: http://spark.apache.org/docs/latest/quick-start.html

PySpark – Main Concepts

Resilient distributed dataset (RDD)*

- Collection of elements that can be operated on in parallel
- To work with data in Spark, RDDs have to be created
- Examples

```
sc = SparkContext('local')
data = sc.parallelize([1, 2, 3, 4]) #use sc.parallize() to create RDD from a list
lines = sc.textFile("text.txt")
```

Actions and transformations, lazy evaluation principle*

* see next lectures

PySpark – Working with MapReduce

MapReduce in PySpark

- *rdd*.map(*f*) -> returns RDD (transformation)
- *rdd*.reduce(*f*) -> returns RDD (transformation)
- *rdd*.collect() -> returns content of RDD as list (action)

Examples: see code examples provided on course website

3. Exercises

Will be discussed during next exercise on 23th of November

Exercises

Install Spark on your computer and configure your IDE to work with PySpark (shown for Anaconda PyCharm on Windows).

1. Implement the word count example in PySpark. Use any text file you like
2. Implement the matrix multiplication example in PySpark.
   - Use the prepared code in matrixMultiplication_template.py and implement the missing parts
3. Implement K-Means in in PySpark. (see lecture slides)
   - Define or generate some points to do the clustering on and initialize 3 centroids.
   - Write two functions *assign_to_centroid(point)* and *calculate_new_centroids(*cluster_points)* to use in your map() and reduce()-calls.
   - Apply map() and reduce() iteratively and print out the new centroids as list in each step