B0M33BDT
Big Data technologies
Spark - Basics

PROFINIT

Josef Vonášek, Tomáš Duda

13. října 2021

Agenda

PROFINIT

1. Spark Overview
2. How Spark works
3. Spark Dataframes
4. Spark Actions
5. Spark Architecture
6. How to start Spark and setup configuration

1

The What, Why, and When of Apache Spark

PROFINIT

- **What:**
  - Unified engine for big data and machine learning
  - Distributed data processing engine -> up to petabytes of data up to thousands of physical or virtual machines
  - Open Source with over 1000 contributors from 250+ organizations
- **Why:**
  - High speed data querying, analysis, and transformation with large data sets.
  - Great for iterative algorithms (using a sequence of estimations based on the previous estimate).
  - Supports multiple languages (Java, Scala, R, Python)
  - Free of charge
- **When:**
  - When you're using functional programming (output of functions only depends on their arguments, not global states)
  - Performing ETL or SQL batch jobs with large data sets
  - Processing streaming
  - Machine Learning tasks

Spark – Facts

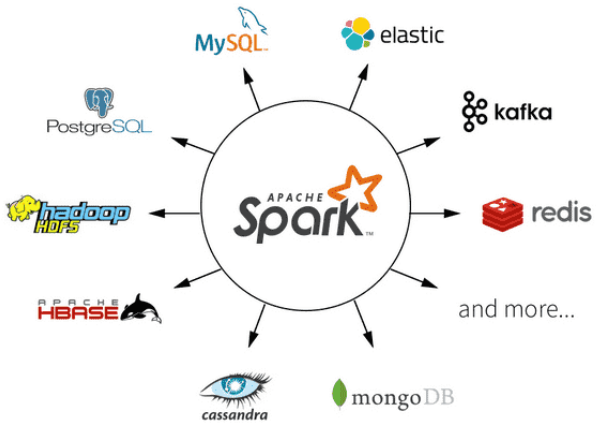
PROFINIT

- In-memory Map-Reduce engine
- Written in Scala
- Fault-tolerant
- Connected with all major big data technologies
- Runs „Everywhere“

Google Cloud

aws

Microsoft Azure



Apache Spark evolution

PROFINIT

- **Spark 1.x – 2014 :**
  - Spark CORE - Fault-tolerant in memory computation engine
  - Spark RDD (Resilient Distributed Dataset) API
  - API for Streaming and Mlib
  - Spark SQL
- **Spark 2.x - 2016:**
  - Speedups the computation 5 to 20 times.
  - API for structured Streaming
  - API for graph data processing
  - SQL 2003 support
  - Datasets API over RDD
- **Spark 3.x - 2020:**
  - adaptive query execution, dynamic partition pruning and other optimizations
  - Significant improvements in pandas APIs, including Python type hints and additional pandas UDFs
  - Up to 40x speedup for calling R user-defined functions
  - SQL ANSI supports

When does Spark work best?

PROFINIT

- On distributed data systems or NoSQL Databases
- Collaboration – Data engineers, data scientist, BI analyst, ..
- Batch and streaming tasks

Common uses:

1. Calculation of client scores (risc score, fraud detection)
2. ETL or SQL batch jobs
3. Using streaming data to trigger a response
4. Machine Learning tasks
5. Graph algorithms

When not to use Spark ?

PROFINIT

- Small
- Low computing capacity (memory)
- Poorly parallelizable
- real-time

e.g.:

1. Modeling on small data
2. Ingesting data in a publish-subscribe model
3. Median calculation
4. JOIN of very big tables

How to learn spark

PROFINIT

- http://spark.apache.org
- Basics in Python | Scala | Java | R
- Help from friends, StackOverflow etc.
- OR **Just try it**

How to work with

PROFINIT

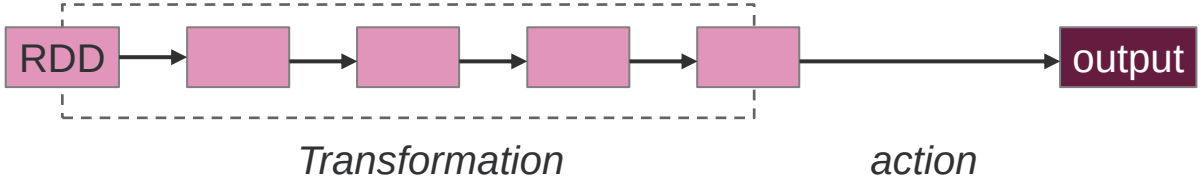
- Interactively
  - Command line (shell for both Python and Scala)
  - Zeppelin/Jupyter notebook
  - Pycharm, IntelliJ, …
- Batch / application
  - compiled .jar file
  - *.py file

2

How Spark works

Logical point of view

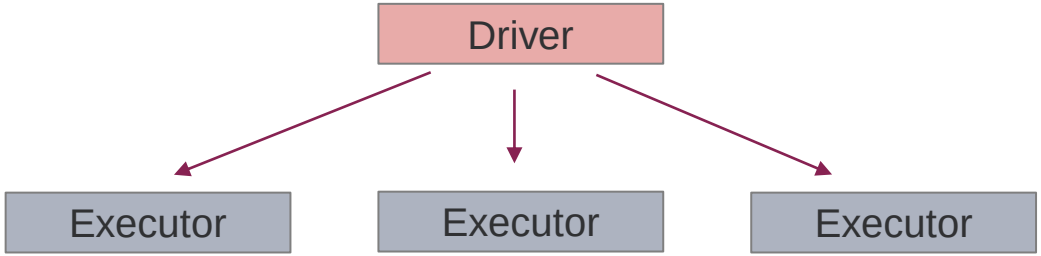
PROFINIT



- **RDD:**
  - resilient distributed dataset - the abstractions of Spark. It is used to handle distributed collection of data elements (e.g.: rows in text file, data matrix, set of binary data) across all the nodes in a cluster.
  - is immutable
- **Transformation:**
  - are planned and optimized, but not evaluated
  - planned as DAG – Direct acyclic graph
- **Action:**
  - action is a trigger that started the whole process

Technical point of view

PROFINIT



- **Driver:**
  - Control all processes
  - Convert user code to transformations and actions -> tasks
  - Distribute tasks across executors
- **Executor:**
  - „worker“ – run tasks and return result to a driver
- **Both run as JVM**

Example – word count:

PROFINIT

- Task: count number of word in document
- Source: text file splitted to lines
- Approach:
  - Load file from disk
  - Transformation of lines: line ⇨ split to words ⇨ split to items (word, 1)
  - Group items with the same word and sum up ones
- Result of transformation: RDD with items (word, frequency)

Example:

PROFINIT

```
lines = sc.textFile("bible.txt")

words = lines.flatMap(lambda line: line.split(" "))

items = words.map(lambda word: (word, 1))

counts = items.reduceByKey(lambda a, b: a + b)



counts.take(5)
```

3

Spark Dataframes

Spark SQL and DataFrames (DataSets)

PROFINIT

- New from spark 2.x ⇨ Enhances the classical RDD approach
- Data structure **DataFrame** = „RDD with columns“
  - similar to database relation table
  - with metadata (field names, types)
  - works with columns –> SQL syntax can be used

RDD

1;Andrea;35;64.3;Praha
2;Martin;43;87.1;Ostrava
3;Simona;18;57.8;Brno

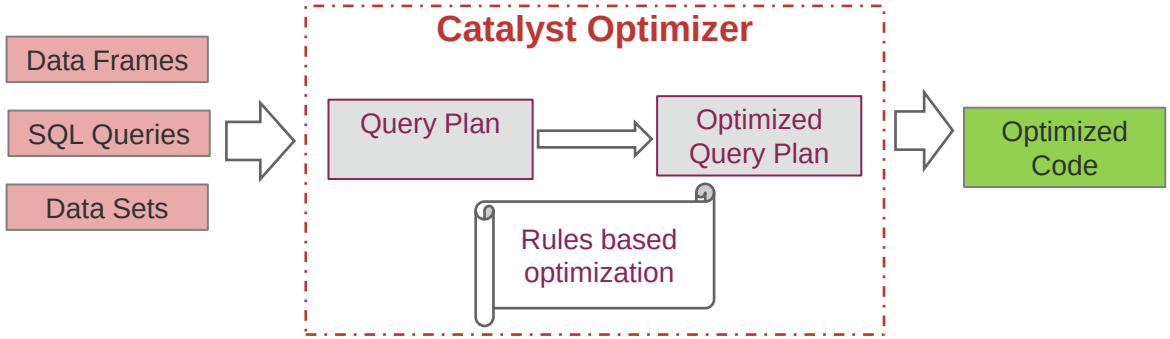
DataFrame

id	name	age	weight	city
1	Andrea	35	64.3	Praha
2	Martin	42	87.1	Ostrava
3	Simona	18	57.8	Brno

WHY use DataFrames

PROFINIT

- Advantage over Spark RDD:
  - Dataframe API - shorter and easier code
  - Columns and Types
  - SQL languague can be used
  - Simplified work with databases
  - **Catalyst Optimizer can be applied**



⇨ Is faster

How to get a DataFrame?

PROFINIT

- transformation from existing RDD
  - if convertable
  - `sqlContext.createDataFrame(RDD, schema)`
- direct input of file
  - schema may be defined (Parquet, ORC) or inferred (CSV)
  - `sqlContext.read.format(format).load(path)`
- Hive query
  - `sqlContext.sql(sql_query)`

How to work with a DataFrame?

PROFINIT

1. registration of temporary table + SQL querying
   - `DF.registerTempTable("table")`
   - `sqlContext.sql("select * from table")`
2. SPARK API
   - `DF.operations`; select, filter, join, groupBy, sort...
3. Convert to RDD -> RDD operation (map, flatMap, shuffle, …) and then convert back -> Dataframe

SPARK API Cheat sheet

PROFINIT

Spark Session APIs
SparkSession.builder.appName(name)
SparkSession.builder.config([key, value, conf])
SparkSession.builder.enableHiveSupport()
SparkSession.builder.getOrCreate()
SparkSession.builder.master(master)
SparkSession.catalog
SparkSession.conf
SparkSession.createDataFrame(data[, schema, ...])
SparkSession.getActiveSession()
SparkSession.newSession()
SparkSession.range(start[, end, step, ...])
SparkSession.read
SparkSession.readStream
SparkSession.sparkContext
SparkSession.sql(sqlQuery)
SparkSession.stop()
SparkSession.streams
SparkSession.table(tableName)
SparkSession.udf
SparkSession.version

DataFrame APIs
DataFrame.agg(*exprs)
DataFrame.alias(alias)
DataFrame.approxQuantile(col, probabilities, ...)
DataFrame.cache()
DataFrame.checkpoint([eager])
DataFrame.coalesce(numPartitions)
DataFrame.colRegex(colName)
DataFrame.collect()
DataFrame.columns
DataFrame.corr(col1, col2[, method])
DataFrame.count()
DataFrame.cov(col1, col2)
DataFrame.createGlobalTempView(name)
DataFrame.createOrReplaceGlobalTempView(nam
DataFrame.createOrReplaceTempView(name)
DataFrame.createTempView(name)
DataFrame.crossJoin(other)
DataFrame.cube(*cols)
DataFrame.describe(*cols)
DataFrame.distinct()
DataFrame.drop(*cols)
DataFrame.dropDuplicates([subset])
DataFrame.drop_duplicates([subset])
...

Input and Output
DataFrameReader.csv(path[, schema, sep, ...])
DataFrameReader.format(source)
DataFrameReader.jdbc(url, table[, column, ...])
DataFrameReader.json(path[, schema, ...])
DataFrameReader.load([path, format, schema]
DataFrameReader.option(key, value)
DataFrameReader.options(**options)
DataFrameReader.orc(path[, mergeSchema, ...
DataFrameReader.parquet(*paths, **options)
DataFrameReader.schema(schema)
DataFrameReader.table(tableName)
DataFrameWriter.bucketBy(numBuckets, col)
DataFrameWriter.csv(path[, mode, ...])
DataFrameWriter.format(source)
DataFrameWriter.insertInto(tableName[, ...])
DataFrameWriter.jdbc(url, table[, mode, ...])
DataFrameWriter.json(path[, mode, ...])

Column APIs
Column.alias(*alias, **kwargs)
Column.asc()
Column.asc_nulls_first()
Column.asc_nulls_last()
Column.astype(dataType)
Column.between(lowerBound, upperBound)
Column.bitwiseAND(other)
Column.bitwiseOR(other)
Column.bitwiseXOR(other)
Column.cast(dataType)
Column.contains(other)
Column.desc()
Column.desc_nulls_first()
Column.desc_nulls_last()
Column.dropFields(*fieldNames)
Column.endswith(other)
Column.eqNullSafe(other)
Column.getField(name)
Column.getItem(key)
Column.isNotNull()
Column.isNull()

Functions
abs(col)
acos(col)
acosh(col)
add_months(start, months)
aggregate(col, initialValue, merge[, finish])
approxCountDistinct(col[, rsd])
approx_count_distinct(col[, rsd])
array(*cols)
array_contains(col, value)
array_distinct(col)
array_except(col1, col2)
array_intersect(col1, col2)
array_join(col, delimiter[, null_replacement])
array_max(col)
array_min(col)
array_position(col, value)
array_remove(col, element)
array_repeat(col, count)
array_sort(col)
array_union(col1, col2)
arrays_overlap(a1, a2)
arrays_zip(*cols)
asc(col)
asc_nulls_first(col)
asc_nulls_last(col)
ascii(col)
asin(col)
asinh(col)
assert_true(col[, errMsg])
atan(col)
atanh(col)
atan2(col1, col2)
avg(col)
base64(col)
bin(col)
bitwiseNOT(col)
broadcast(df)
bround(col[, scale])
bucket(numBuckets, col)
cbrt(col)
.....

Window
Window.currentRow
Window.orderBy(*cols)
Window.partitionBy(*cols)
Window.rangeBetween(start, end)
Window.rowsBetween(start, end)
Window.unboundedFollowing
Window.unboundedPreceding
WindowSpec.orderBy(*cols)
WindowSpec.partitionBy(*cols)
WindowSpec.rangeBetween(start, end)
WindowSpec.rowsBetween(start, end)

Grouping
GroupedData.agg(*exprs)
GroupedData.apply(udf)
GroupedData.applyInPandas()
GroupedData.avg(*cols)
GroupedData.cogroup(other)
GroupedData.count()
GroupedData.max(*cols)
GroupedData.mean(*cols)
GroupedData.min(*cols)
GroupedData.pivot(pivot_col[, values])
GroupedData.sum(*cols)

Example

PROFINIT

› Which USA state has the highest average temperature in summer?

Data sample:

```
station,month,day,hour,temperature,flag,latit,longit,elevation,state,name
AQW00061705,1,1,1,804,P,-14.3306,-170.7136,3.7,AS,PAGO PAGO WSO AP
AQW00061705,1,2,1,804,P,-14.3306,-170.7136,3.7,AS,PAGO PAGO WSO AP
AQW00061705,1,3,1,803,P,-14.3306,-170.7136,3.7,AS,PAGO PAGO WSO AP
AQW00061705,1,4,1,802,P,-14.3306,-170.7136,3.7,AS,PAGO PAGO WSO AP
AQW00061705,1,5,1,802,P,-14.3306,-170.7136,3.7,AS,PAGO PAGO WSO AP
```

Solution by RDD only

PROFINIT

```
tp_raw = sc.textFile('7-12.csv')

tp_raw = tp_raw.filter(lambda r: (r.split(',')[1] in
set('678')) & (r.split(',')[4] !=''))

tp = tp_raw.map(adjust_row)

tp_st = tp.reduceByKey(sum).map(lambda x:
(x[0],x[1][0]/x[1][1])).sortBy(lambda y: y[1], False)

tp_st.take(1)
```

```
def adjust_row(rlist):
    stat = rlist[9]
    tepl = (int(rlist[4])/10.0 -
    32)*5/9
    return (stat, (tepl, 1))
```

```
def sums(a, b):
    sumA = a[0]
    sumB = b[0]
    cntA = a[1]
    cntB = b[1]
    return (sumA + sumB, cntA +
    cntB)
```

Solution by DataFrame

PROFINIT

```
import pyspark.sql.functions as F

df_temp = spark.read.format("com.databricks.spark.csv").load('teplota7-12.csv')

df_filtered = df_temp.filter((df_temp['mesic'] >= 6) &
                             (df_temp['mesic'] <= 8)).select('stat', 'teplota').dropna()


df_celsius = df_filtered.withColumn('teplota', (df_filtered['teplota'] / 10.0 - 32) * 5 / 9)


df_avg = df_celsius.groupBy('stat').avg('teplota').toDF('stat', 'temp_avg')


df_desc = df_avg.orderBy(F.col('temp_avg').desc())

df_desc.show(1)
```

Solution by DataFrame

PROFINIT

```
df_temp.registerTempTable("t")

df_sql= sqlContext.sql("
select stat, avg((teplota/ 10.0 - 32) * 5 / 9) as temp_avg
from t
where mesic>5 and mesic<9 and teplota is not null
group by stat order by temp_avg desc")

df_sql.limit(1).show()
```

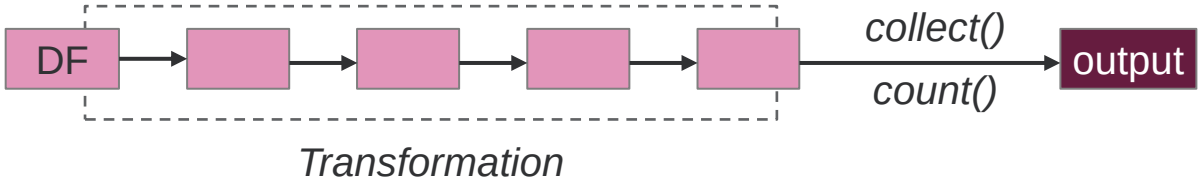
4

Spark actions

Spark Actions

PROFINIT

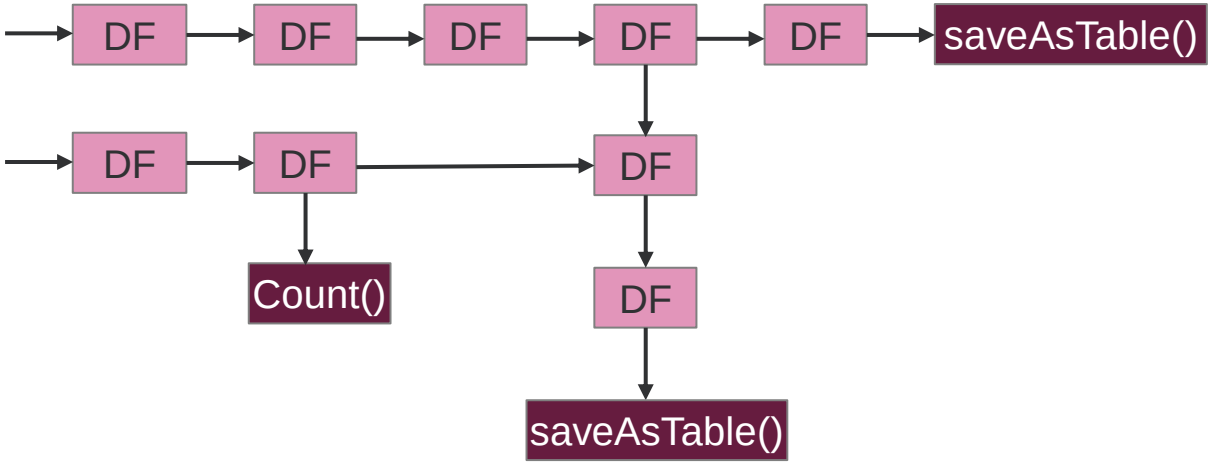
RDD	Dataframe	Description
take	take,show	Show first n rows
count	count	Count of rows
collect	collect	Show rdd/dataframe as list of rows
saveAsTextFile	saveAsTable, write	Save file/create table
…	…	



› Every action starts all steps of transformation from the beginning!

Spark caching

PROFINIT



HOW to CACHE

PROFINIT

- **Methods:**
  - `persist()` (several options)
  - `cache()` (use persist with MEMORY_ONLY option)
  - `unpersist()` (release persisted data)

- **Persist options:**
  - MEMORY_ONLY – Default –> deserialized JVM memory
  - MEMORY_AND_DISK –> excessed partitions into disk.
  - MEMORY_ONLY_SER -> serialized JVM memory
  - MEMORY_AND_DISK_SER -> etc.

- Persist is not an action!

Example

PROFINIT

- Which USA state has the lowest average temperature in summer?

```
df_asc=df_avg.orderBy(F.col('temp_avg').asc())
df_asc.show(5)
```

5

Spark Architecture

Important Terminologies of Apache Spark

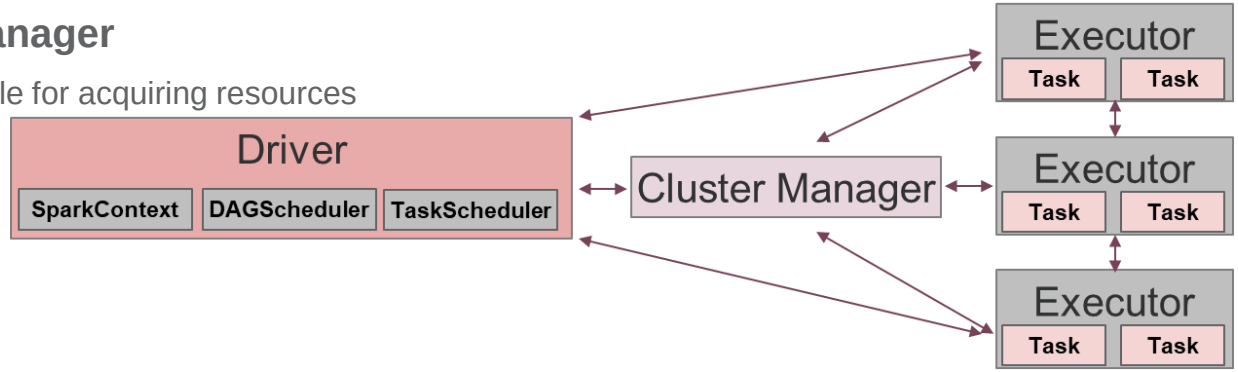
PROFINIT

- SparkContext
  - It is the main entry point to spark core. Act as master of spark application
  - Getting the current status of spark application
  - Canceling the job and the stage
  - Running job synchronously or asynchronously
- Spark Application
  - It is a self-contained computation that runs user-supplied code to compute a result.
- Job
  - It parallels computation consisting of multiple tasks.
- Stage
  - Each job is divided into small sets of tasks which are known as **stage.**
- Task
  - It is a unit of work, which we sent to the executor. It is a single operation (map, filter, …) applied to a single partition.

Components of Spark Architecture

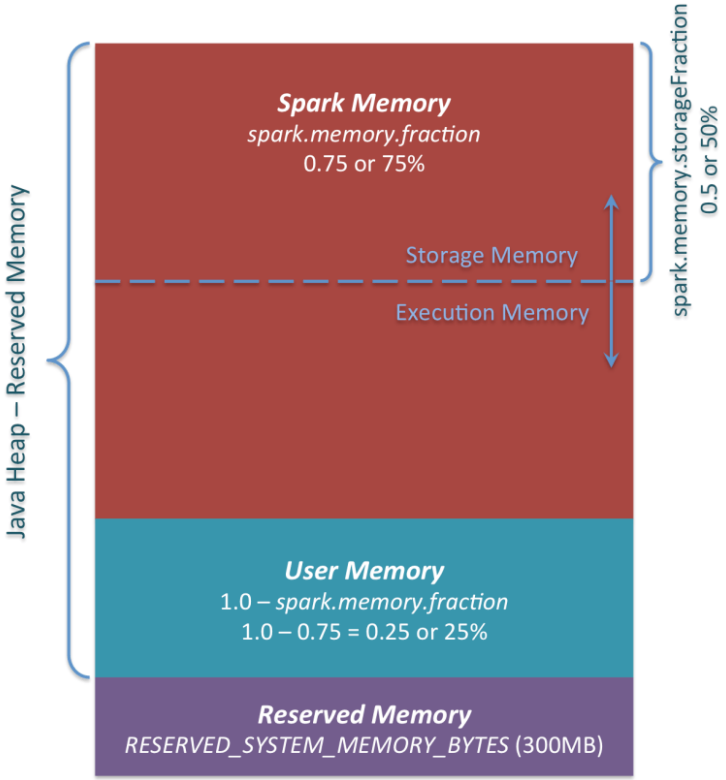
PROFINIT

- **Driver**
  - It is a **master node**.
  - Translates user code into a specified job.
  - Schedules the job execution and negotiates with the cluster manager.
  - Stores the metadata about all RDDs as well as their partitions.
  - The key component is a SparkContext, others are DAG Scheduler, task scheduler, backend scheduler and block manager.
- **Executors**
  - They are distributed agents those are responsible for the execution of tasks
  - They perform all the data processing
- **Cluster Manager**
  - Responsible for acquiring resources



Spark Memory

PROFINIT



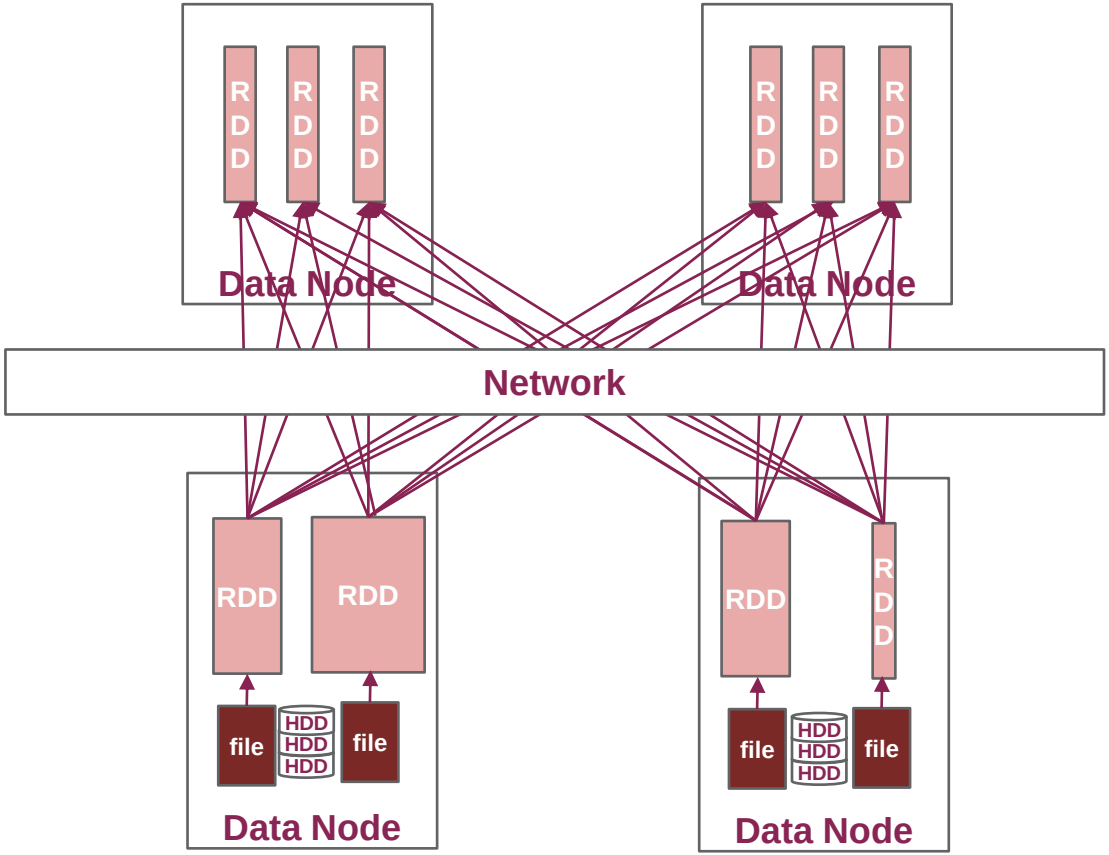
How are date data splitted – partitioning

PROFINIT

- **partition**
  - part of data managed in one task
  - default partition = 1 HDFS block = 1 task = 1 core
  - partition is ideally managed on the node where is stored
  - More partitions ⇨ more tasks ⇨ higher paralelization
    ⇨ smaller ⇨ lower efficiency ⇨ higher overhead
- **How to change it?**
  - in load: `sc.textFile(file, count_of_partitions)`
  - In the program:
    - `coalesce (count_of_partitions)`
    - `repartition(count_of_partitions)`
    - `partitionBy (count_of_partitions)`
- **Default for Joins:**
  - The default number of partitions to use when shuffling data for joins or aggregations.
  - `spark.sql.shuffle.partitions = 200`

Data locality & shuffling

PROFINIT



Example:

PROFINIT

› Count of rows for every USA state that is in source file?

```
df_temp.registerTempTable("t")
df_sql= sqlContext.sql("select * from t")


df_sql.groupBy('stat').count().show(100)
df_sql.repartition(2000).groupBy('stat').count().show(100)
```

6

Start and configuration

How to start Spark

PROFINIT

```
pyspark | spark-shell | spark-submit --param value
```

Useful parameters:

--name -> name of the application

--class -> The entry point for your application

--master -> The master URL for the cluster (local, Yarn, Mesos, ..)

--deploy_mode -> where the driver will be deployed (client/cluster)

--driver-memory -> memory for driver

--num-executors -> count of executors

--executor-cores -> count of cores for executor

--executor-memory -> memory for *executor*

How to start Spark

PROFINIT

Example:

```
./bin/spark-submit \
--master yarn
--deploy-mode client \
--name my_spark_application \
--driver-memory 1G \
--num-executors 3 \
--executor-cores 2 \
--executor-memory 3G
```

Example how to allocate resources

PROFINIT

General recomendation:

- --num-cores <= 5
- --executor-memory <= 64 GB

Cluster 6 nods, every 16 cores a 64 GB RAM

- Reserve 1 core a 1GB /node for OS
  rest is 6 *15 cores a 63 GB
- 1 core for spark Driver: 6 * 15 – 1 = 89 cores.
- 89 / 5 ~ 17 executors. Every node (except the one with driver) will have 3 executors.
- 63 GB / 3 ~ 21 GB memory per executor. Moreover count with memory overhead -> set up 19 GB per executor

Thank you

PROFINIT

Profinit EU, s.r.o., Tychonova 2, 160 00 Praha 6
Tel.: + 420 224 316 016, web: www.profinit.eu

LinkedIn
linkedin.com/company/profinit

Twitter
twitter.com/Profinit_EU

Facebook
facebook.com/Profinit.EU

Youtube
Profinit EU