DĚLAT
DOBRÝ SOFTWARE
NÁS BAVÍ

PROFINIT

Rozšíření Sparku

Jan Hučín

22. března 2019

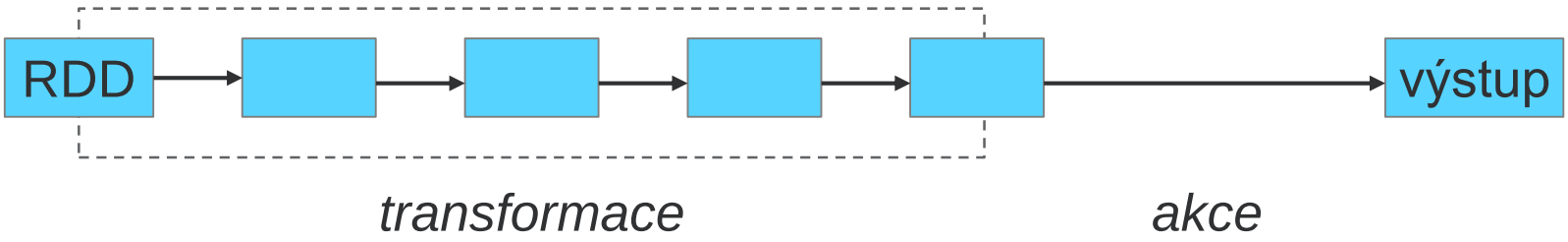
Osnova

1. Opakování Sparku RDD
2. Spuštění a konfigurace
3. Přehled rozšíření Sparku
4. Spark SQL

PROFINIT

Spark RDD – opakování

Spark RDD



- série transformací zakončená akcí
- transformace se **plánují** a **optimalizují**, ale zatím **neprovádějí**
- **lazy evaluation:** až první akce spustí celý proces

Co je to RDD?

- resilient distributed dataset
- kolekce prvků (např. řádky v textovém souboru, datová matice, posloupnost binárních dat)
- musí být dělitelné na části – místo rozdělení (spolu)určí Spark!

PROFINIT

Příklad 1 – word count

Úkol: spočítat četnosti slov v dokumentu

Vstup: textový soubor rozdělený do řádků (RDD)

```
lines = sc.textFile("/user/pascepet/bible.txt")
```

Postup:

- transformace řádků: řádek ⇨ jednotlivá slova (více prvků)

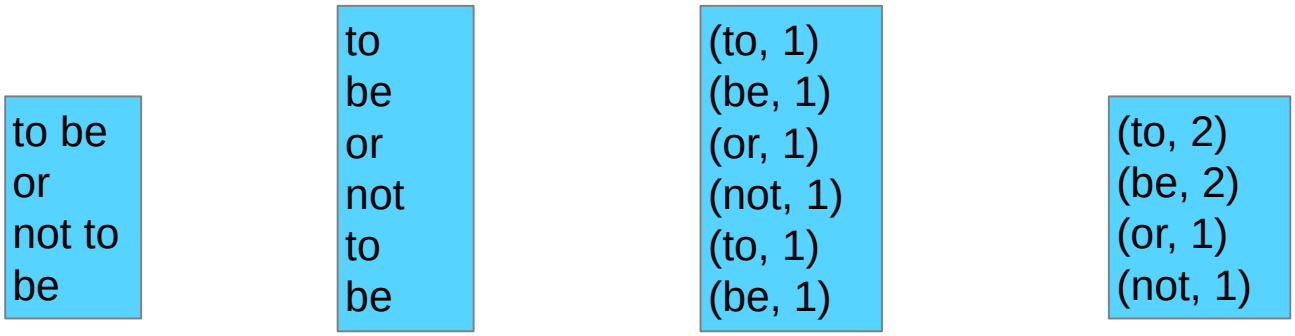
```
words = lines.flatMap(lambda line: line.split(" "))
```

- transformace řádků: řádek čili slovo ⇨ struktura (slovo, 1)

```
pairs = words.map(lambda word: (word, 1))
```

- seskupení prvků se stejným klíčem a sečtení jedniček

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



PROFINIT

Kešování

- Kešování: RDD se nezapomene, ale uchová v paměti / na disku.
- **Metody pro kešování:**
  - **cache** (pokusí se uchovat v paměti)
  - **persist** (obecnější, např. serializace, využití disku atd.)
  - **unpersist** (uvolnění RDD z paměti)
- **Typy kešování:**
  - MEMORY_ONLY
  - MEMORY_AND_DISK
  - MEMORY_ONLY_SER
  - MEMORY_AND_DISK_SER
- SER = serializace – úspora paměti, ale vyšší výpočetní náročnost
  - Volby se SER pouze Java a Scala, v Pythonu serializace vždy
- Kešování není akce!

Spuštění a konfigurace

Spuštění Sparku

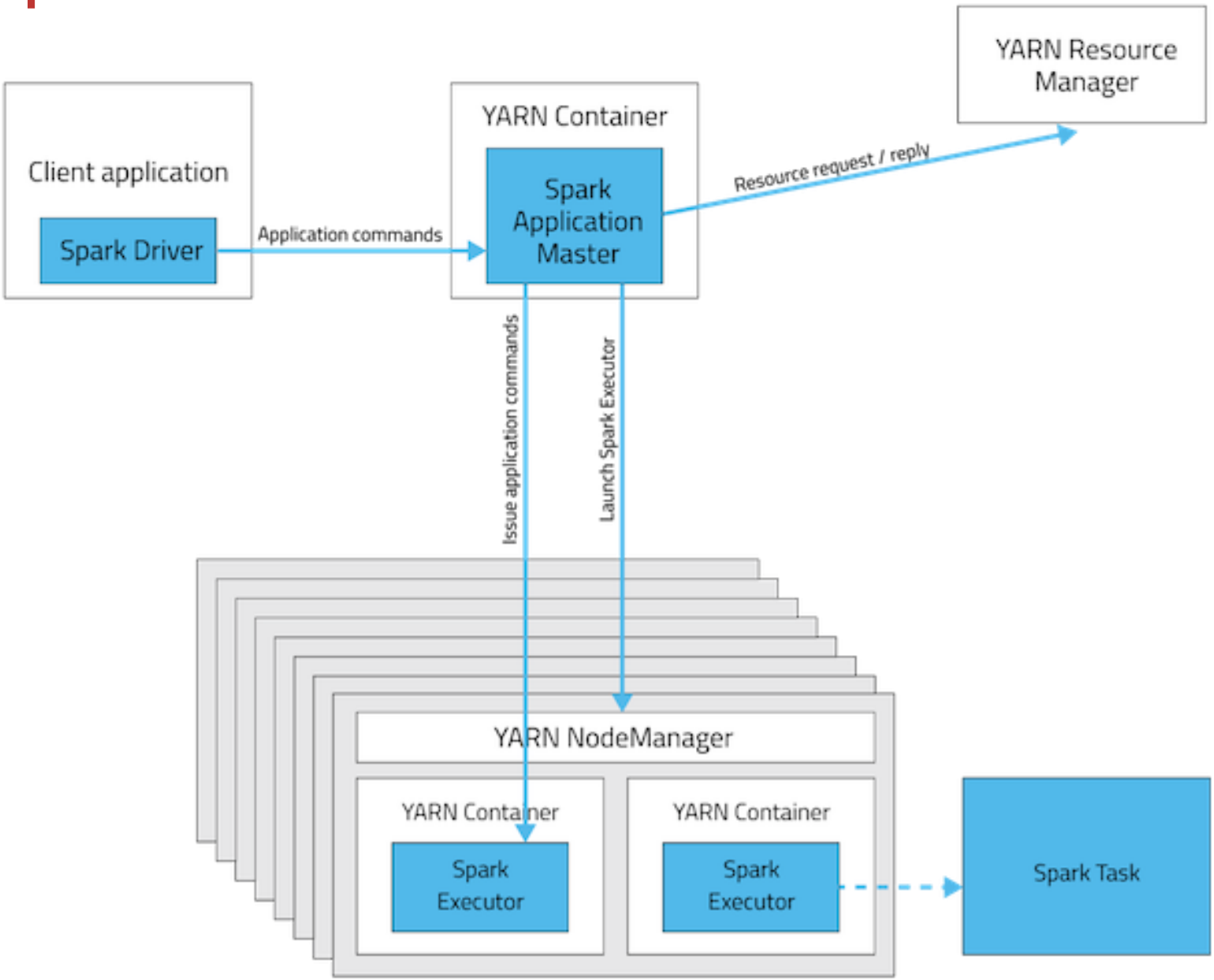
```
pyspark | spark-shell | spark-submit --param value
```

Kde a jak poběží

- na clusteru – plné využití paralelismu
  - mod client
  - mod cluster
- lokálně – paralelní běh na více jádrech
- určeno parametry --master a --deploy-mode

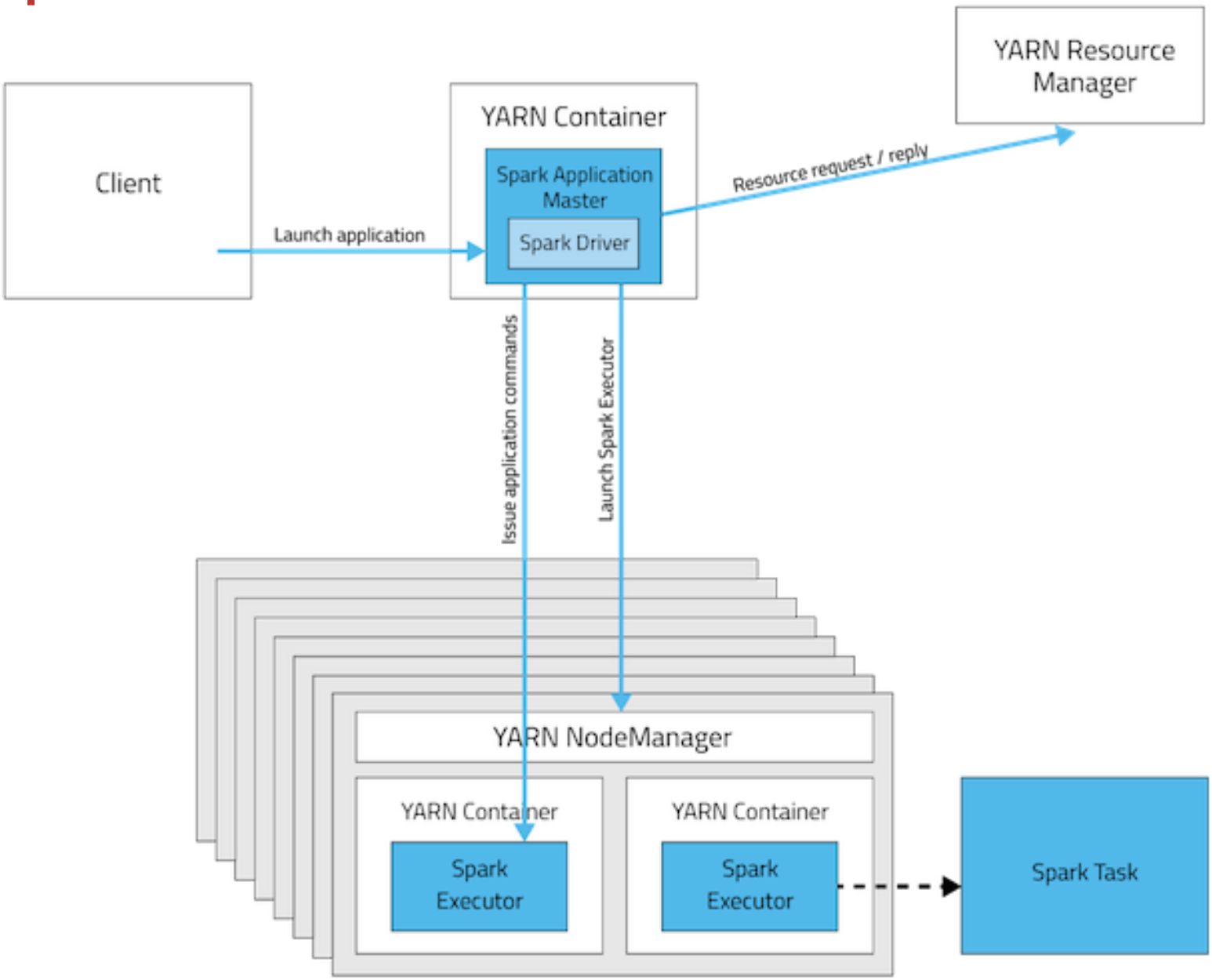
PROFINIT

Spark on YARN client mode



PROFINIT

Spark on YARN cluster mode



Mod client versus mod cluster

- default je client
- mod client je vhodný pro interaktivní práci a debugging (výstup jde na lokální konzolu)
- mod cluster je vhodný pro produkční účely

PROFINIT

Konfigurace běhu Sparku – požadavky na zdroje

- --name *jméno aplikace*
- --driver-memory *paměť pro driver*
- --num-executors *počet exekutorů*
- --executor-cores *počet jader pro exekutor*
- --executor-memory *paměť pro exekutor*

Příklad

```
pyspark --master yarn --deploy-mode client
--driver-memory 1G
--num-executors 3 --executor-cores 2
--executor-memory 3G
```

PROFINIT

Příklad plánu alokace zdrojů

Obecná doporučení:

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

Cluster 6 nodů, každý 16 jader a 64 GB RAM

- Rezervovat 1 jádro a 1GB /node pro OS
  zbývá 6 *15 jader a 63 GB
- 1 jádro pro Spark Driver: 6 * 15 – 1 = 89 jader.
- 89 / 5 ~ 17 exekutorů. Každý node (kromě toho s driverem) bude mít 3 exekutory.
- 63 GB / 3 ~ 21 GB paměti na exekutor. Navíc se musí počítat s memory overhead -> nastavit 19 GB na exekutor

PROFINIT

Rozšíření Sparku

Rozšíření Sparku

Spark SQL

- práce se strukturovanými daty pomocí SQL přístupu

GraphX

- rozšíření pro algoritmy prohledávající grafy

Spark ML

- klasické modely (regrese, stromy atd.)
- mnohorozměrná statistika (clustering, PCA atd.)
- algoritmy pro velká data (asociační pravidla, doporučování atd.)

Spark Streaming

- near real-time dávkové zpracování příchozích dat

PROFINIT

Spark SQL

Spark SQL a DataFrames (DataSets)

- Rozšíření k tradičnímu RDD přístupu
- Datová struktura **DataFrame** = RDD se sloupci
  - obdoba databázové relační tabulky
  - obsahuje i schéma
  - nad rámec RDD – práce se sloupci
  - možnost použití syntaxe podobné SQL nebo přímo SQL

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

id	jmeno	vek	hmotnost	mesto
1	Andrea	35	64.3	Praha
2	Martin	42	87.1	Ostrava
3	Simona	18	57.8	Brno

Spark SQL – výhody a nároky

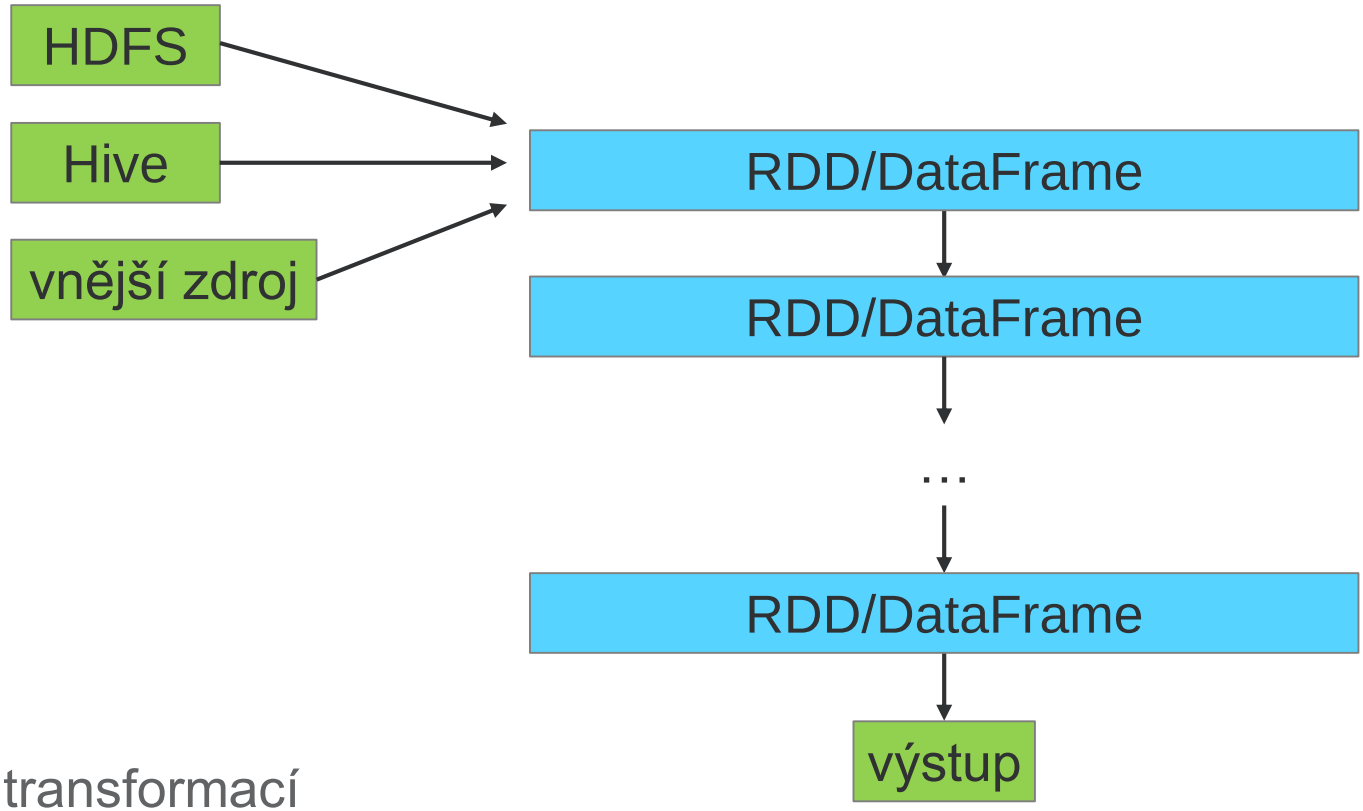
- Výhody oproti tradičnímu Sparku (RDD):
  - stručnější a jednodušší kód
  - využití Hive
  - snazší optimalizace
  - ⇨ rychlejší běh

- Nároky navíc:
  - rozšířené API: objekt **`sqlContext`**, ev. další

- Kdy nelze použít?
  - úlohy nevhodné pro SQL ⇨ tradiční Spark
  - úlohy náročné na paměť ⇨ map-reduce, Hive

PROFINIT

Spark RDD a SQL



- série transformací zakončená akcí
- lze transformovat RDD na DataFrame a obráceně

PROFINIT

Spark RDD a SQL

- Transformace RDD → RDD
  - už známe: map, flatMap, filter, …
- DataFrame → DataFrame
  RDD → DataFrame
  DataFrame → RDD
  - naučíme se

PROFINIT

Příklad – společné zadání

- Který stát USA má na meteostanicích nejvyšší průměrný normál v létě?
  (již jsme řešili pomocí Hive)

Struktura dat:

```
stanice,mesic,den,hodina,teplota,flag,latitude,longitude,vyska,stat,nazev
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
```

PROFINIT

Postup 0 (jen RDD)

```
tp_raw = sc.textFile('/user/pascepet/teplota')

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

tp = tp_raw.map(uprav_radek)

tp_st = tp.reduceByKey(soucty) \

    .map(lambda x: (x[0], x[1][0]/x[1][1])) \

    .sortBy(lambda y: y[1], False)

tp_st.take(1)
```

```
uprav_radek

AQW00061705,7,30,4,804,P,-14.3306,-170.7136,3.7,AS,PAGO PAGO WSO AP
                                  ↓
                          (AS, (26.89, 1))
```

PROFINIT

Jak vyrobit DataFrame?

- transformace z existujícího RDD
  - je-li převoditelné do sloupců
- přímé načtení souboru
  - s již definovanými sloupci (např. Parquet, ORC)
  - převoditelné do sloupců (např. CSV)
- výsledek dotazu do Hive
- výsledek dotazu do jiné DB (JDBC konektor)

PROFINIT

Jak vyrobit DataFrame?

- transformace z existujícího RDD
  - `sqlContext.createDataFrame(RDD, schema)`
- přímé načtení souboru
  - `sqlContext.read.format(formát).load(cesta)`
- výsledek dotazu do Hive
  - `sqlContext.sql(dotaz_sql)`

PROFINIT

Postup 1 (CSV → RDD → DataFrame)

```
from pyspark.sql.types import *

tp_raw = sc.textFile('/user/pascepet/data/teplota')

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

tp_prep = tp_raw.map(uprav_radek_df)

tpDF = sqlContext.createDataFrame(tp_prep, tp_schema)
```

```
uprav_radek_df

AQW00061705,7,30,4,804,P,-14.3306,-170.7136,3.7,AS,PAGO PAGO WSO AP
                                  ↓
                              (AS, 26.89)
```

```
tp_pole = [StructField('stat', StringType(), True),
StructField('teplota', DoubleType(), True)]

tp_schema = StructType(tp_pole)
```

PROFINIT

Postup 2 (přímé načtení CSV → DataFrame)

```
tpDF2 = sqlContext.read \
        .format("com.databricks.spark.csv") \
        .option("header", "true") \
        .option("delimiter", ",") \
   buď .schema(nazev_schematu) \
  nebo .option("inferSchema", "true") \
        .load("/user/pascepet/data/teplota")
```

Postup 3 (Hive → DataFrame)

```
tpDF3 = sqlContext.sql('select * from temperature')
```

PROFINIT

Jak pracovat s DataFrame?

1. registrovat jako dočasnou tabulku + dotazování SQL
2. pseudo-SQL operace
3. operace RDD – výsledek může být jen obyčejné RDD

Jak pracovat s DataFrame?

1. registrovat jako dočasnou tabulku + dotazování SQL
   - `DF.registerTempTable("tabulka")`
   - `sqlContext.sql("select * from tabulka")`
2. pseudo-SQL operace
   - `DF.operace`, např. select, filter, join, groupBy, sort...
3. operace RDD – výsledek může být jen obyčejné RDD
   - např. map, flatMap...
   - řádek v DataFrame je typu **Row** – práce jako s typem **list**

PROFINIT

Výpočet pomocí dočasné tabulky

```
tpDF.registerTempTable("teploty")

tp_stDF = sqlContext.sql("""select stat, avg(teplota) as
tepl_prum from teploty
group by stat order by tepl_prum desc""")

tp_stDF.show(1)
```

```
tp_pole = [StructField('stat', StringType(), True),
StructField('teplota', DoubleType(), True),
StructField('mesic', DoubleType(), True)]

tp_schema = StructType(tp_pole)
```

PROFINIT

Pseudo-SQL a další operace

- **select** (omezení na uvedené sloupce)
- **filter** (omezení řádků podle podmínky)
- **join** (připojení jiného DataFrame)
- **groupBy** (seskupení)
- **agg, avg, count** (agregační funkce)
- **toDF** (přejmenování sloupců)
- **withColumn** (transformace sloupců)
- **show** (hezčí výpis obsahu DataFrame)

Výpočet pomocí pseudo-SQL

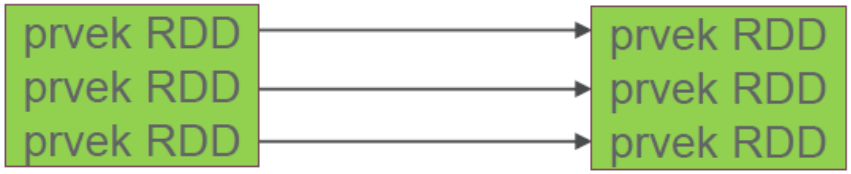
```
tpDF.registerTempTable("teploty")

tp_stDF = sqlContext.sql("""select stat, avg(teplota) as
tepl_prum from teploty
group by stat order by tepl_prum desc""")

tp_stDF.show(1)
```

```
tpDF2 = tpDF2.filter((tpDF2.mesic>5) & (tpDF2.mesic<9)) \
    .select('stat','teplota').na.drop()
tpDF2 = tpDF2.groupBy('stat').avg() \
    .toDF('stat','tepl_prum')
tpDF2.sort(tpDF2.tepl_prum.desc()).limit(1).show()
```

PROFINIT

Transformace v RDD a v DataFrame

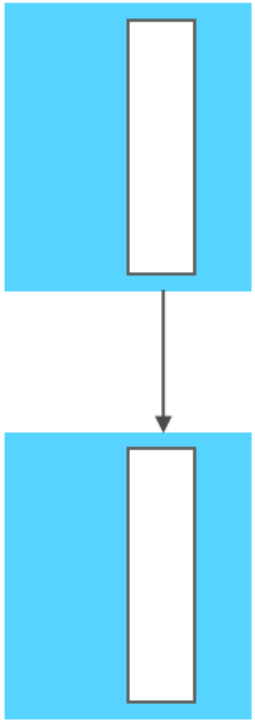
RDD map



transformační funkce:

Python
obvyklé knihovny a objekty
(string.lower, re.search atd.)

DataFrame withColumn



transformační funkce:

Pyspark SQL functions

manipulace se sloupci

nutno importovat, např.
from pyspark.sql import functions as F

F.lower, F.regexp_replace atd.

PROFINIT

Pyspark SQL functions – příklady

- **split** (rozdělení řetězce – výsledek je array)
- **size** (počet prvků array – odpovídá funkci len)
- **lower** (na malá písmena)
- **regexp_replace** (náhrada podle regulárního výrazu)
- **udf** (uživatelská funkce – pokud nelze použít funkci Spark SQL)
- **when... otherwise** (ifelse)
- **lit** (konstanta)

Příklad:

```
RDD2 = RDD1.map(lambda s: s[2].lower)


from pyspark.sql import functions as F
DF2 = DF1.withColumn('tepl_mala', F.lower(DF1.tepl))
```

PROFINIT

Díky za pozornost

PROFINIT

Profinit, s.r.o.
Tychonova 2, 160 00 Praha 6

Telefon
+ 420 224 316 016

Web
www.profinit.eu

LinkedIn
linkedin.com/company/profinit

Twitter
twitter.com/Profinit_EU