

DĚLAT
DOBRÝ SOFTWARE
NÁS BAVÍ

PROFINIT

Spark SQL, Spark Streaming

Jan Hučín

18. listopadu 2020

Osnova

1. Spark SQL
2. Další rozšíření Sparku
 - Spark streaming
 - GraphX
 - Spark ML



Spark SQL

The background of the slide is a dark gray field filled with a complex, overlapping pattern of translucent, light gray polygons. These polygons vary in size and orientation, creating a sense of depth and movement, similar to a low-poly or crystalline aesthetic.

Spark SQL – proč?

- › Tradiční RDD přístup na strukturovaná data je nešikovný
- › Příklad – výpočet státu s nejvyšší prům. teplotou v létě:

```
AQW00061705,7,30,4,804,P,-14.3306,-170.7136,3.7,AS,PAGO PAGO WSO AP
parsing řádku
filter – jen léto, bez hlaviček sloupců
map → (stat, (teplota, 1))
reduceByKey → (stat, (soucet=sum(teplota), pocet=sum(1)))
map → (stat, (prumer=soucet/pocet))
seřadit, vypsát první
```

Spark SQL a DataFrames (DataSets)

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

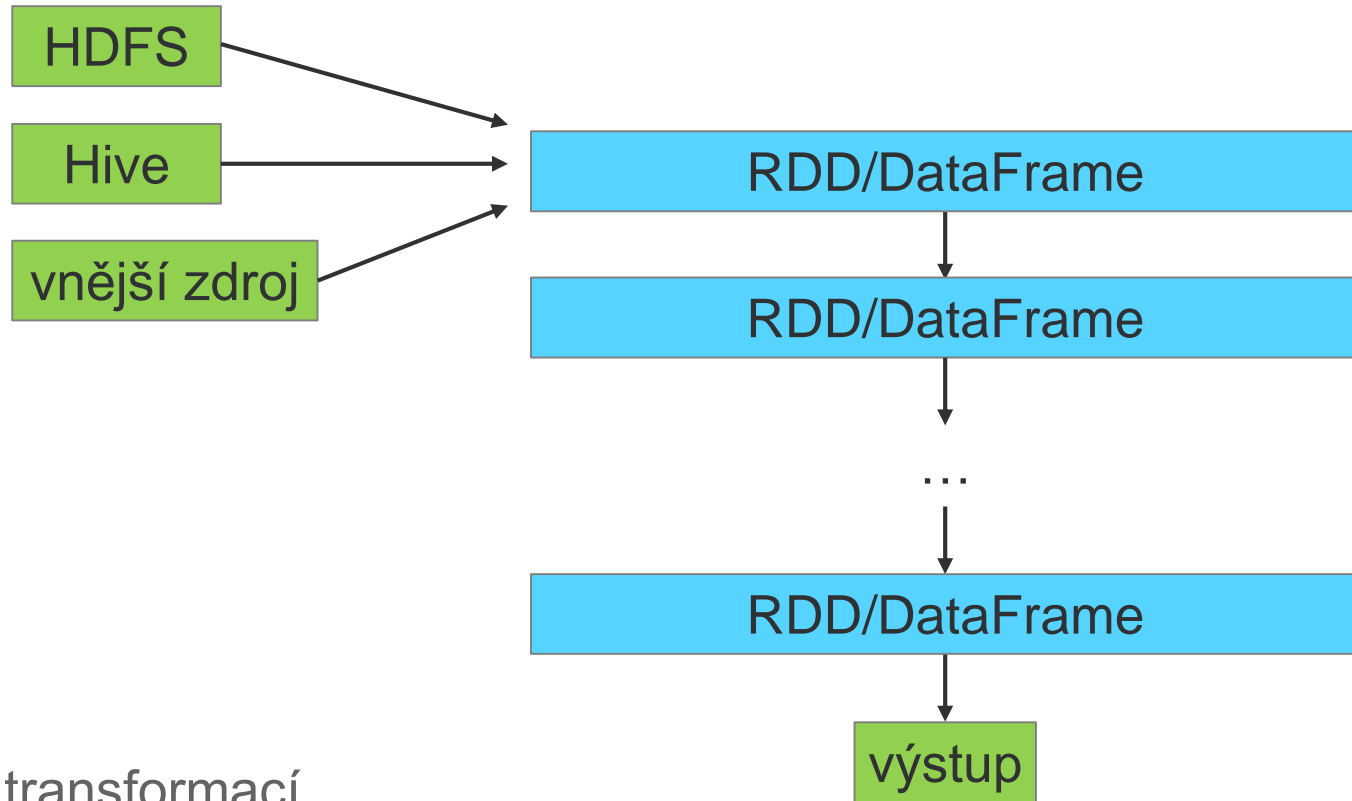
1;Andrea;35;64.3;Praha
2;Martin;43;87.1;Ostrava
3;Simona;18;57.8;Brno

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:
 - v podstatě žádné, je už dlouho součástí standardní instalace
 - od Sparku 2 využívá objekt `sparkSession` (u nás *spark*)
- › Kdy nelze použít?
 - úlohy nevhodné pro SQL ⇒ tradiční Spark
 - úlohy náročné na paměť ⇒ map-reduce, Hive

Spark RDD a SQL



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

Spark RDD a SQL

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

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 nebo jiné DB (JDBC konektor)

Jak vyrobit DataFrame?

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

Příklad

Data z meteostanic USA
(již jsme využili pro Hive a Spark RDD)

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

Postup 1 (přímé načtení CSV → DataFrame)

```
tpDF1 = spark.read \  
    .format("csv") \  
    .option("header", "true") \  
    .option("delimiter", ",") \  
    .option("inferSchema", "true") \  
    .load("/user/pascepel/data/teplota")
```

Postup 2 (Hive → DataFrame)

```
tpDF2 = spark.sql('select * from fel_bigdata.teplota')
```

Postup 3 (RDD → DataFrame)

- › řádek → Row(pole1=hodnota, pole2=hodnota, ...)
- › Row – něco jako *list*, umožňuje spojit data do sloupců
- › **CreateDataFrame** si určí typy podle hodnot v Row
- › můžeme ale při konverzi definovat i vlastní schéma
- › místo **CreateDataFrame** lze použít i **toDF**

```
from pyspark.sql import Row  
tp_prep = tp_raw.map(uprav_radek_df)  
tpDF3 = spark.createDataFrame(tp_prep)
```

Jak pracovat s DataFrame?

1. registrovat jako dočasnou tabulku + dotazování SQL
– výsledek je opět DataFrame
2. pseudo-SQL operace (podobné *pandas*)
– výsledek je opět DataFrame
3. tradiční operace pro RDD (map, flatMap...)
– nutno explicitně převést na obyčejné RDD

Jak pracovat s DataFrame?

1. registrovat jako dočasnou tabulku + dotazování SQL
 - `DF.registerTempTable("tabulka")`
 - `spark.sql("select * from tabulka")`
2. pseudo-SQL operace
 - `DF.operace`, např. select, filter, join, groupBy, sort...
3. operace RDD – nutno explicitně převést na obyčejné RDD
 - `DF.rdd.map(...)` apod.
 - řádek v DataFrame je typu **Row** – práce jako s typem **list**

Registrace dočasné tabulky + SQL

```
tpDF.registerTempTable("teploty")

tp_stDF = spark.sql("""select stat, avg(teplota) as tepl_prum
from teploty
where mesic in (6,7,8)
group by stat order by tepl_prum desc""")

tp_stDF.show(1)
```


Pseudo-SQL a další operace

Syntaxe i funkčnost je velmi podobná *pandas* pro Python.

- › **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)

Totéž pomocí pseudo-SQL

```
tpDF.registerTempTable("teploty")

tp_stDF = spark.sql("""select stat, avg(tepl) as tepl_prum
from teploty
where mesic in (6,7,8)
group by stat order by tepl_prum desc""")

tp_stDF.show(1)
```

```
tpDF2 = tpDF2.filter((tpDF2['mesic']>=6) & (tpDF2['mesic']<=8)) \
    .select('stat','teplota').dropna()

tpDF2 = tpDF2.groupBy('stat').avg() \
    .toDF('stat','tepl_prum')

tpDF2.orderBy(tpDF2['tepl_prum'].desc()).show(1)
```

Spark Streaming

Co to je a jak to využít



- › dávkové zpracování přicházejících dat
- › příchozí data neklepou na dveře, sedí v čekárně
- › near real-time, pevné nastavení časového okna

Možné využití:

- › filtrování logů, zpráv
- › monitorování, reakce na událost
- › vyhledávání v nakešovaných datech

Princip zpracování



- › streamovací modul dávkuje příchozí data – posloupnost RDD
- › klasický Spark postupně odbavuje RDD ve frontě
- › API pro Javu, Scalu, s malým omezením i Python

Příklad

Úkol: pro každou dávku ze socketu spočítat četnosti slov

```
sc = SparkContext(appName="Příklad")
ssc = StreamingContext(sc, 10)

lines = ssc.socketTextStream("localhost", 9999)
counts = lines.flatMap(lambda line: line.split(" ")) \
               .map(lambda word: (word, 1)) \
               .reduceByKey(lambda a, b: a+b)

counts.pprint()

ssc.start()
ssc.awaitTermination()
```



GraphX Spark ML

GraphX

- › rozšíření pro algoritmy prohledávající grafy
- › ve stadiu vývoje
- › připravené algoritmy:
 - PageRank
 - rozklad na podgrafy
 - počet trojúhelníků
 - label propagation
 - a další...
- › API pro iterativní procházení grafů (Pregel)

Spark ML (machine learning)

- › klasické algoritmy machine learning, např.:
 - regrese, lineární modely
 - rozhodovací stromy
 - naivní Bayesův klasifikátor
 - shluková analýza
- › algoritmy pro velká data, např.:
 - doporučovací systém
 - asociační pravidla, časté podmnožiny
- › statistické metody, např.:
 - popisná statistika
 - testování hypotéz
- › mnohorozměrné metody, např.:
 - hlavní komponenty
 - faktorová analýza

Spark ML (machine learning)

- › praktický smysl mají jen algoritmy pro velká data
- › u ostatních metod:
 - z velkých dat se vybere vzorek
 - na vzorku se tradičními nástroji modeluje
 - navržený model se naprogramuje ve Sparku (bez Spark ML)

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