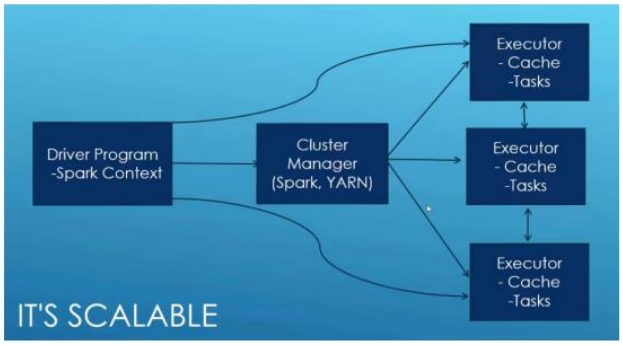
37. Apache Spark – popis a srovnání s Apache Hadoop, jednotlivé komponenty a jejich význam. Koncepty RDD a DataFrame- princip, popis a rozdíly. Transformace vs. akce.

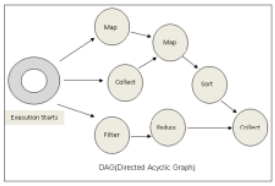
Apache Spark – popis a srovnání s Apache Hadoop, jednotlivé komponenty a jejich význam.

- rychlý a obecný engine pro zpracování dat ve velkém měřítku
  - umožňuje zpracovávat, distribuovat a analyzovat velká data na clusteru
  - open source
    - spadá pod Apache Software Foundation
  - v aktivním vývoji
    - prvotní verze v únoru 2013 vytvořená na UC Berkeley
    - aktuální verze 3.3.1 (říjen 2022)
- je škálovatelný
  - Spark skripty (Driver Programs) jsou obyčejné skripty napsané např. ve Scale
    - možné spouštět i lokálně
    - pod kapotou je ale Spark umí rozdělit na různé stroje (procesory) v clusteru
  - Spark běží nad správcem clusteru (Cluster Manager)
    - má vlastní, ale je možné použít i jiné správce, např. YARN (na Hadoop clusteru)
  - správce clusteru rozděluje a koordinuje práci mezi vykonávače (Executors)
    - více exekutorů na jednom stroji (ideálně 1 na 1 jádro CPU)
    - poskytuje také odolnost vůči chybám
      - výpadek jednoho exekutoru neznamená přerušení vykonávané práce
  - horizontální dělení a škálování
  - z vývojářského pohledu se jedná jen o program běžící na jednom stroji
    - o vše ostatní se stará Spark

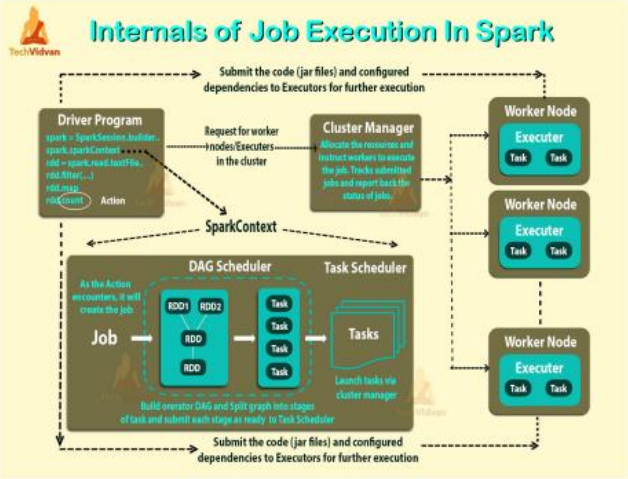
- je škálovatelný



- je rychlý
  - využívá orientované acyklické grafy
    - DAG engine (directed acyclic graph)
    - optimalizuje workflow
  - v praxi
    - Spark vyčkává do chvíle, než je požádán o výsledky
    - následně vybere optimální cestu pro zodpovězení otázky



POHLED DOVNITŘ



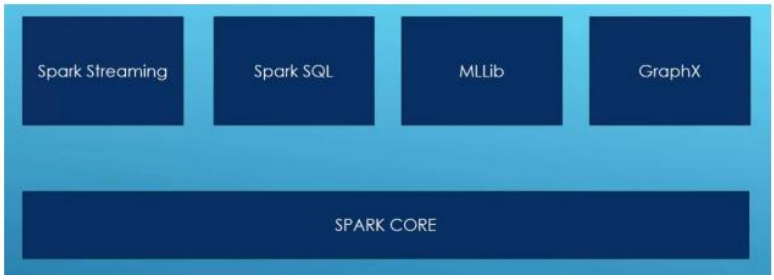
Porovnání s Hadoop

- Spark vs. Hadoop MapReduce
  - sdílí společné rysy
  - MapReduce vyžaduje soubory uložené na HDFS, Spark ne
    - Spark podporuje Cassandru, AWS S3, HDFS, …
  - Spark je výrazně rychlejší
    - až 100× rychlejší s daty v paměti a až 10× rychlejší při práci s daty na disku
    - v reálné aplikaci je rozdíl menší, ale stále výrazný
  - MapReduce po každé operaci map a reduce zapisuje většinu dat na disk
  - Spark udržuje většinu dat po každé transformaci v paměti
    - po naplnění paměti začne také používat disk

- není až tak těžký
  - programování v Pythonu, Javě, R nebo Scale
    - samotný Spark je napsaný ve Scale
  - postavený na jednom základním konceptu
    - resilient distributed dataset (RDD; odolný distribuovaný soubor dar)
  - ➢ programování je efektivnější než u Hadoop MapReduce
    - mappery a reducery v některých případech nahrazeny i jen jedním řádkem kódu

Komponenty a jejich význam

- je složený z komponent



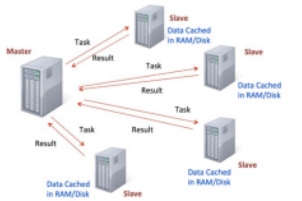
- je složený z komponent
  - Spark Core
    - práce s RDD, distribuce zpracování velkých dat, …
  - Spark Streaming
    - podpora pro analýzu streamovaných dat v reálném čase
  - Spark SQL
    - práce se strukturovanými daty a následné dotazování
  - MLLib
    - algoritmy pro strojové učení
  - GraphX
    - algoritmy pro teorii grafů

Koncepty RDD a DataFrame – princip, popis a rozdíly. Transformace vs. akce.

1.RDD

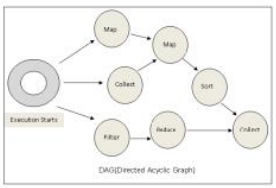
RESILIENT DISTRIBUTED DATASET

- odolný distribuovaný soubor dat (RDD)
  - základní koncept Apache Spark
  - dataset (soubor dat)
    - abstrakce pro velkou sadu dat
  - distributed (distribuovaný)
    - možnost objekty rozdělit po celém clusteru
  - resilient (odolný)
    - automatická obnova a přerozdělení práce v případě výpadku uzlu
  - Spark se o distribuci a odolnost stará na pozadí sám
    - podporuje také paralelní operace (partitioning)
  - ➢ z pohledu vývoje je důležité vědět, že se jedná o reprezentaci velkých dat
    - RDD objekty lze použít k transformaci jedné datové sady na druhou
    - nad datovou sadou lze provádět akce a získávat výsledky



https://www.udemy.com/course/spark-and-python-for-big-data-with-pyspa

- líně vyhodnocované
  - v Driver Programu se nic neděje, dokud není zavolána konkrétní akce
  - až po zavolání je sestaven orientovaný acyklický graf a akce je spuštěna
  - ➢ veškeré zpracování transformací je tedy až po zavolání dané akce
    - šetří výkon na obrovských datových sadách




```
>>> rdd = sc.parallelize([1, 2, 2, 1])
>>> nrdd = rdd.map(lambda x: x*x)
>>> nrdd.collect()
[1, 4, 4, 1]
```

https://www.udemy.com/course/taming-big-data-with-apache-spark-hands-on/ https://datafloq.com/read/apache-spark-a-basic-understanding/3493

- neměnné a kešované

Transformace

- transformace RDD's
  - operace proveditelné nad RDD's
  - reprezentují postup receptu
  - map
    - mění vstupní datovou sadu na výstupní pomocí zadané funkce nad RDD
    - vztah 1:1, každý vstupní záznam je mapován právě na jeden výstupní záznam
    - např. výpočet kvadrátu, map bude ukazovat na funkci, která násobí číslo samo sebou
  - flatmap
    - podobné operaci map, ale umožňuje více výstupních hodnot pro každý vstupní záznam
  - filter
    - pro vystřihnutí nezajímavých informací
    - např. webové logy, ale důležitá informace je jen v error lozích
    - filter odstraní záznamy neobsahující slovo error
  - distinct
    - odstranění duplikátů a zajištění unikátních hodnot
  - sample
    - pro výběr náhodné podmnožiny RDD
    - např. pro ladění skriptů na menší datové sadě pro urychlení
  - union, intersection, subtract, cartesian
    - operace s množinami
    - přijímají dvě RDD's a vrací finální RDD
    - konjunkce, disjunkce, odčítání, kartézský součin

- transformace RDD's
  - ukázka map
    - výpočet kvadrátu
    - s využitím lambda funkce
  - odbočka k funkcionálnímu programování
    - hodně RDD metod přijímá funkci jako parametr

```
rdd = sc.parallelize([1, 2, 3, 4])
rdd.map(lambda x: x*x)

This yields 1, 4, 9, 16
```

Many RDD methods accept a *function* as a parameter

```
rdd.map(lambda x: x*x)
```

Is the same thing as

```
def squareIt(x):
    return x*x

rdd.map(squareIt)
```

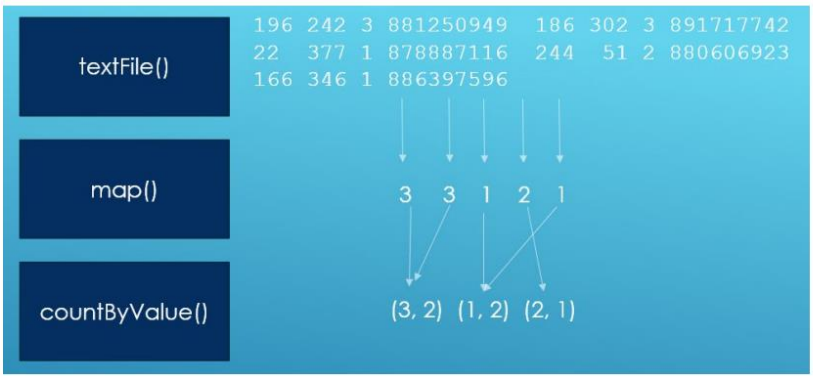
There, you now understand functional programming.

Akce

- akce nad RDD's
  - akce recept provádí a vrací zpátky výsledek
  - collect
    - výpis všech hodnot v RDD
  - count, countByValue
    - počet hodnot
    - počet výskytů jednotlivých hodnot
  - take, top
    - vzorek hodnot z finálního RDD
  - reduce
    - kombinace hodnot na základě funkce
  - a další…

Co se děje na pozadí

- co se ale děje na pozadí?
  - nejprve je na základě RDD's vytvořen plán provedení (DAG)



- úloha je rozdělena na etapy (stages) podle potřeby reorganizace dat
- každá etapa může být rozdělena na úkoly (tasky)
  - možnost distribuce tasků na clusteru

➢ úkoly jsou naplánovány na clusteru (scheduled) a zavoláním akce provedeny

- RDD's umožňují uchovávat i dvojice klíč – hodnota
  - kromě jednoduché hodnoty (řádek textu, hodnocení filmu)
- koncepčně připomínají NoSQL databáze
  - obří key-value úložiště
  - umožňují např. agregace podle klíče
- např. počet přátel podle věku
  - věk je klíčem, počet přátel hodnotou
  - ukládáno jako (věk, početKamarádů)
- key/value RDD's jsou vytvářena podobně jako klasické RDD's
  - např. namapování párů dat do RDD
  - podpora i pro list hodnot jako value

```
totalsByAge = rdd.map(lambda x: (x, 1))
```

https://www.udemy.com/course/taming-big-data-with-apache-spark-hands-on/

2. Dataframe

SPARK SQL

- v současnosti jedna z nejdůležitějších komponent Apache Spark
  - poskytuje snadné propojení SQL dotazů a Spark programů
  - ➢ zahrnuje DataFrame API
    - od verze 2.0 a postupně nahrazuje RDD API
  - rozšiřuje RDD na DataFrame objekt
  - ➢ modul pro práci se strukturovanými daty
    - uniformní přístup
    - nabízí vysoký výkon a škálovatelnost


```
spark.read.json("s3n://...")
  .registerTempTable("json")
results = spark.sql(
  """SELECT *
     FROM people
     JOIN json ..."""")
```

```
results = spark.sql(
  "SELECT * FROM people")
names = results.map(lambda p: p.name)
```

- DataFrame
  - odpovídá velké databázové tabulce
  - skládá se z řádkových objektů (Row) obsahujících sloupce
  - umožňuje snadné dotazování dat pomocí SQL dotazů
    - na celém clusteru
  - může, ale nemusí mít přiřazené schéma
    - efektivnější ukládání, pokud ano
  - snadný a uniformní zápis a čtení z různých formátů
    - JSON, Hive, Parquet, CSV, …
  - komunikace s venčím
    - JDBC/ODBC, Tableau
    - jeví se jako velká relační databáze

- provedení SQL dotazu na DataFramu
- dotaz na pohled pomocí SQL
  - jména sloupců musí odpovídat sloupcům v DataFrame
  - výsledkem je opět DataFrame

```
myResultDataFrame = spark.sql("SELECT foo FROM bar ORDER BY foobar")
```

https://www.udemy.com/course/taming-big-data-with-apache-spark-hands-on/

- ➢ extrémně snadné načtení strukturovaných dat
  - a snadné dotazování pomocí SQL

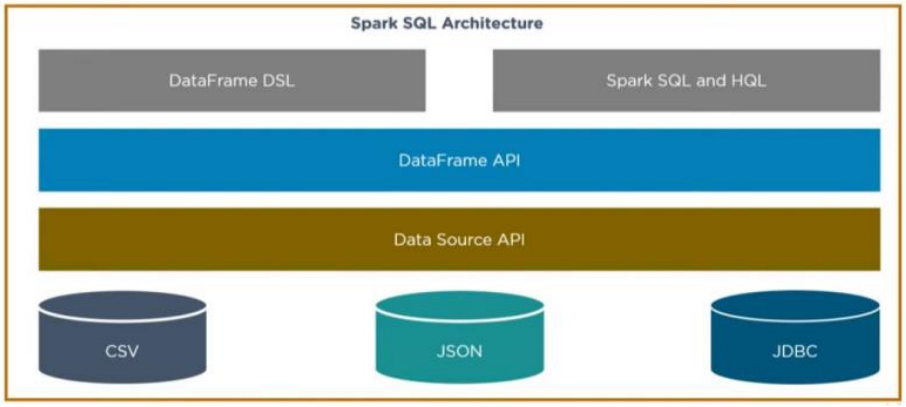
- SQL ale nemusí být použito přímo…
- možnost volat funkce přímo nad DataFramy
  - emulují SQL příkazy
  - místo SQL SELECT funkce select
  - filter na výběr určitých dat
  - show pro zobrazení výsledků
  - konverze na RDD pomocí funkce rdd

```
myResultDataFrame.show()
myResultDataFrame.select("someFieldName")
myResultDataFrame.filter(myResultDataFrame("someFieldName" > 200)
myResultDataFrame.groupBy(myResultDataFrame("someFieldName")).mean()
myResultDataFrame.rdd().map(mapperFunction)
```

- DataFramy jsou populární
  - trend používání směřuje od RDD's k DataFramům
  - DataFramy zajišťují vyšší vnitřní kompatibilitu
    - další Spark komponenty přechází primárně na DataFramy
    - MLLib od verze 3.0 podporuje již jen DataFramy
      - původní RDD verze již není udržovaná
    - Spark Streaming také primárně používá DataFrame API
      - structured streaming
    - ➢ DataFramy používány pro výměnu informací mezi jednotlivými komponentami
  - DataFramy ulehčují vývoj
    - velké množství SQL operací nad DataFramy jen jedním řádkem kódu
    - o dost snazší, než vymýšlení převodu na MapReduce problém a použití RDD's

Porovnání

- DataFrame vs. DataSet
  - dotýká se více Scaly než Pythonu
  - DataFrame je na pozadí DataSet objekt složený z řádkových objektů (Row)
    - nemají přiřazený datový typ
  - DataSet ale může zaobalit i známé strukturované datové typy
    - Python je ale netypový jazyk, takže toho nedokáže plně využít
    - ve Scale je ale použití DataSetů velmi prospěšné
    - efektivnější ukládání
    - optimalizace během kompilace
    - detekce chyb během kompilace



SESKUPENÍ A AGREGACE

- seskupení pomocí operace groupBy
  - shlukování řádků na základě určité hodnoty sloupce
  - např. seskupení dat z prodeje podle dne

```
+-------+-------+-----+
|Company| Person|Sales|
+-------+-------+-----+
|   GOOG|    Sam|200.0|
|   GOOG|Charlie|120.0|
|   GOOG|  Frank|340.0|
|   MSFT|   Tina|600.0|
|   MSFT|    Amy|124.0|
```

- na seskupená data je možné aplikovat agregační funkce
  - kombinuje vstupní řádky na jediný výstup
  - např. suma prodejů za konkrétní den
  - mean, sum, max, min, count

```
df.groupBy('Company').mean().show()
```

```
+-------+------------------+
|Company|       avg(Sales)|
+-------+------------------+
|   APPL|             370.0|
|   GOOG|             220.0|
|     FB|             610.0|
|   MSFT|322.3333333333333|
+-------+------------------+
```

- agregace je ale možné zavolat i na neseskupená data
  - pomocí funkce .agg
  - vstupem je slovník

```
df.agg({'Sales':'max'}).show()
```

```
+----------+
|max(Sales)|
+----------+
|     870.0|
+----------+
```

- definice schématu

```
schema = StructType([ \
                     StructField("stationID", StringType(), True), \
                     StructField("date", IntegerType(), True), \
                     StructField("measure_type", StringType(), True), \
                     StructField("temperature", FloatType(), True)])
```

```
ITE00100554,18000101,TMAX,-75,,,E,
ITE00100554,18000101,TMIN,-148,,,E,
GM000010962,18000101,PRCP,0,,,E,
EZE00100082,18000101,TMAX,-86,,,E,
EZE00100082,18000101,TMIN,-135,,,E,
```

https://www.udemy.com/course/taming-big-data-with-apache-spark-hands-on/