TSVD 8

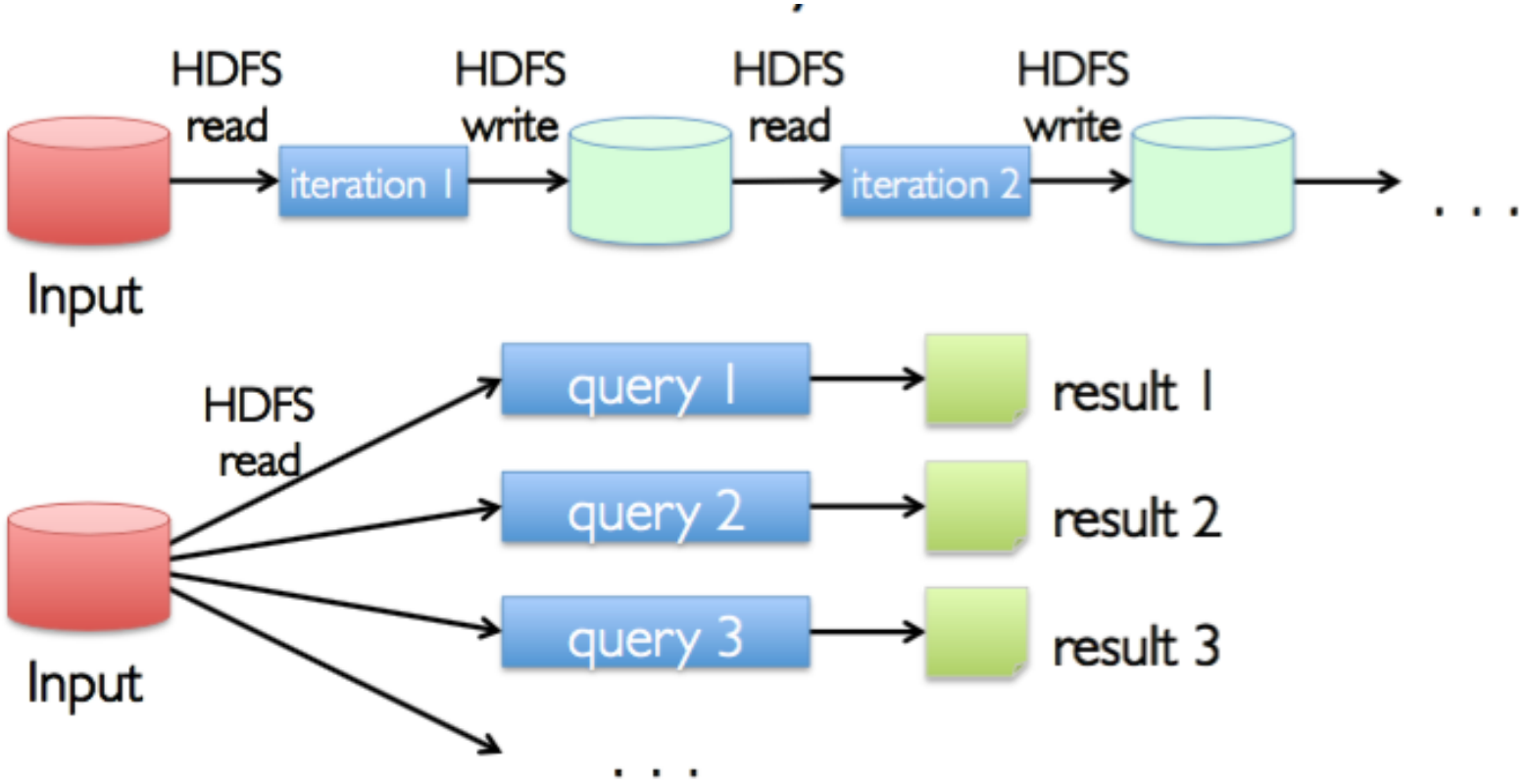
Technológie spracovania veľkých dát

Peter Bednár, Martin Sarnovský

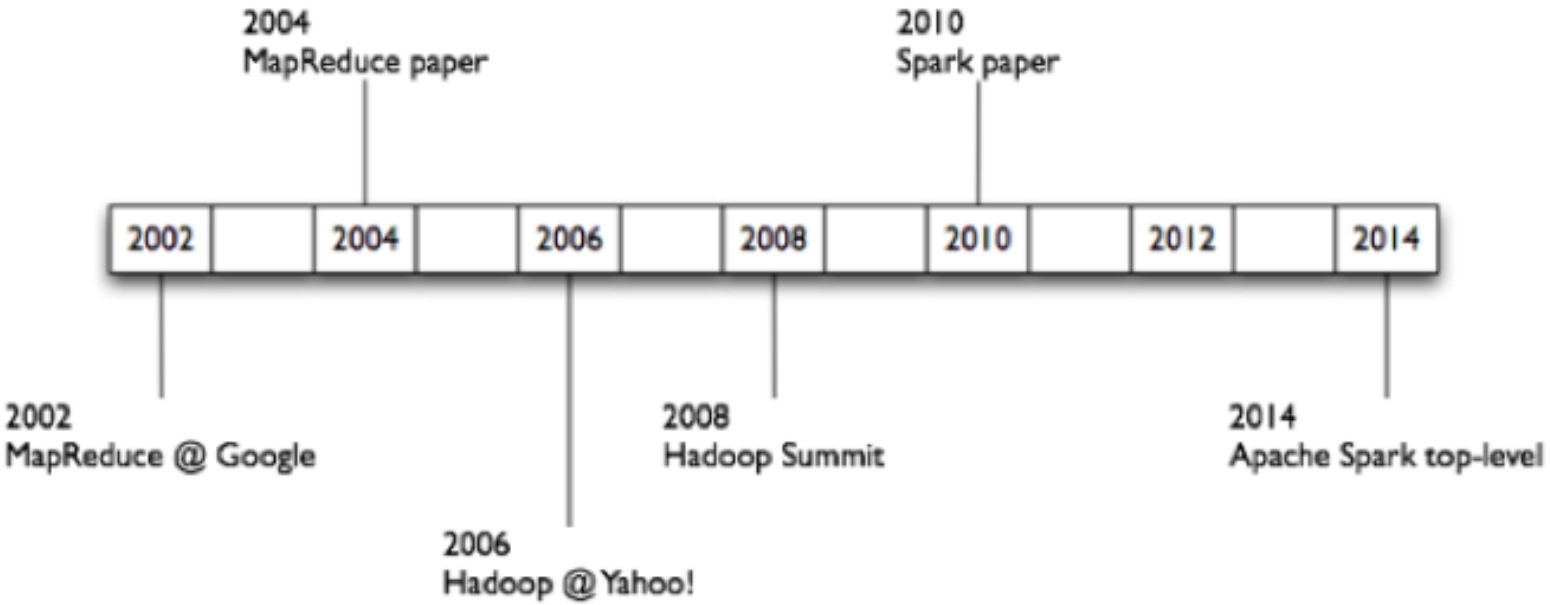
Nedostatky MapReduce + Hadoop

- MapReduce častokrát využívané v úlohách spracovania veľkého množstva dát na klastroch
- Postavené na acyklických dátových tokoch – neefektívne pri aplikáciách, ktoré opakovane používajú rovnakú množinu dát:
  - Iteratívne algoritmy (časté v úlohách strojového učenia)
  - Interaktívne úlohy dolovania v dátach (R, Excel, Python)
- Snaha o presun výpočtov do pamäte, in-memory computing – výpočty v pamäti, podpora cyklických dátových tokov
- Umožiť opakovane využívať množinu skôr použitých dát bez ich opätovného čítania z úložiska
- Ponechať niektoré vlastnosti použiteľné z MapReduce:
  - Fault tolerance
  - Data locality
  - Škálovateľnosť

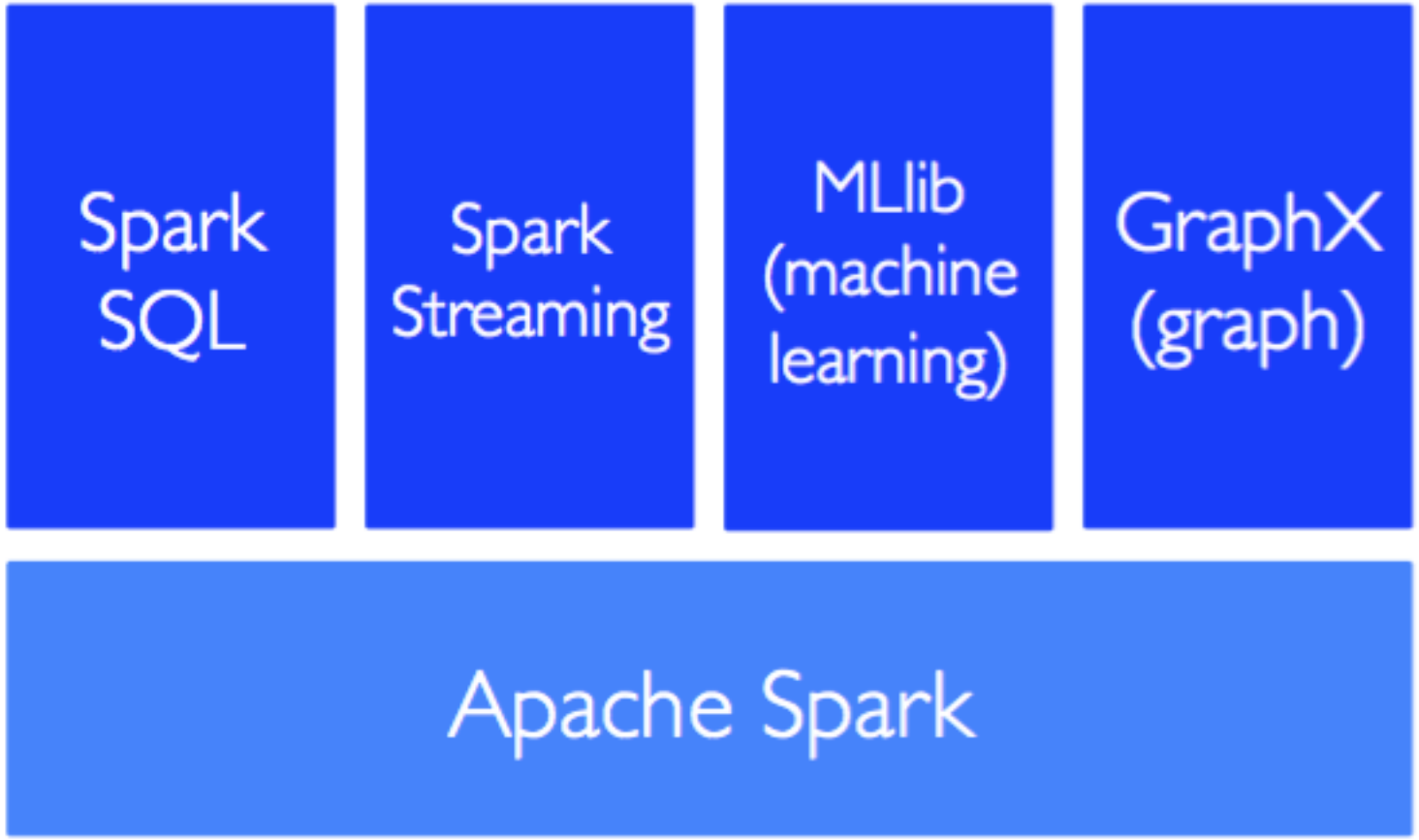
Použitie pamäte miesto disku



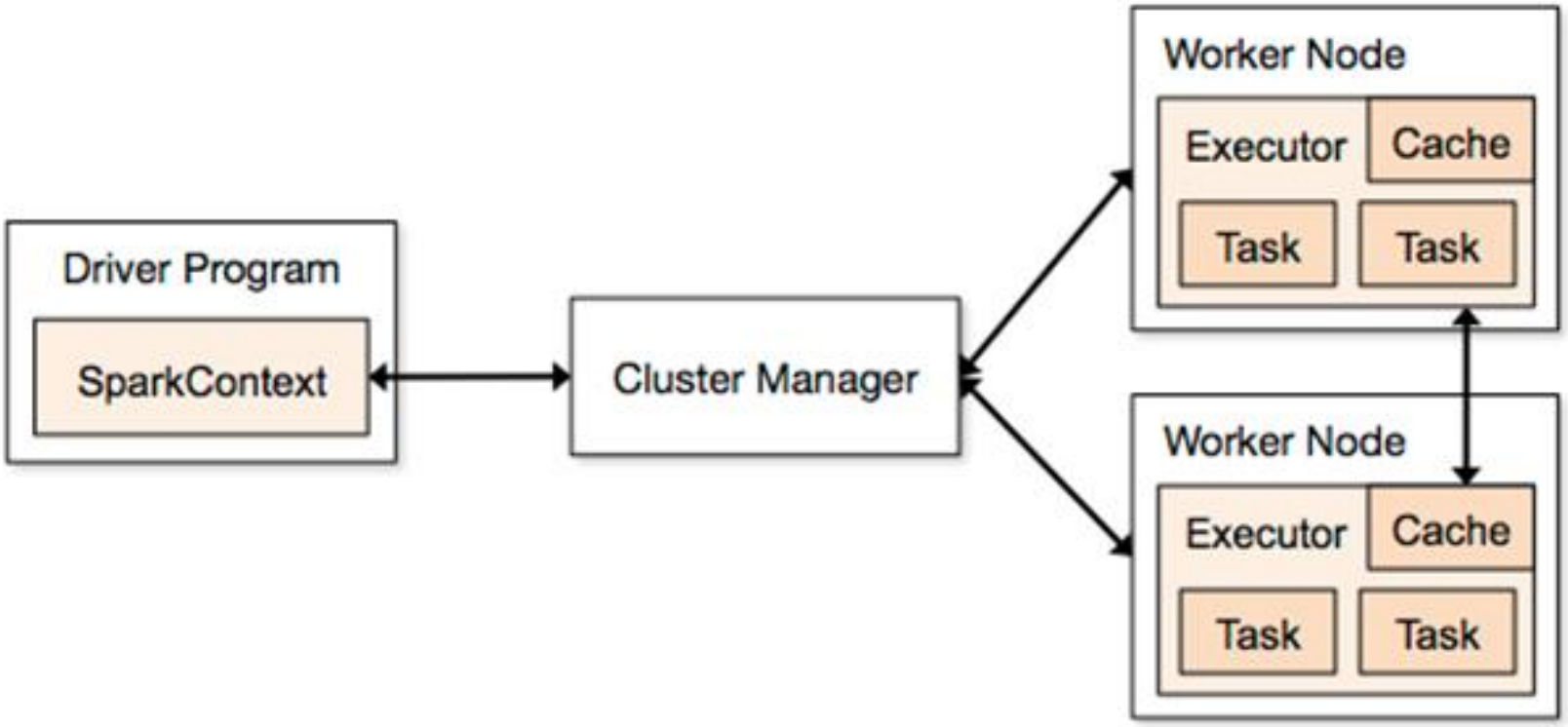
Apache Spark timeline



Spark platforma



Spark architektúra



Spark architektúra

- Každá Spark aplikácia pozostáva z Riadiaceho programu (Driver program) – pozostáva z main funkcie a spúšťa sériu paralelných operácií na klastri (tasks)
- Aplikácia vytvára SparkContext, ktorý koordinuje riadiaci program - aplikácie bežia ako nezávislé skupiny procesov na uzloch klastra
- SparkContext špecifikuje ako, kde a kedy sa pristúpiť ku klastru, vie využiť niekoľko druhov Manažéra zdrojov (Cluster managerov) – napr. YARN, Mesos, atď.
- Manažér zdrojov (Cluster manager) – alokuje zdroje v klastri pre aplikáciu
- Priradí Executor-y na pracovných uzloch klastra, čo sú procesy, ktoré vykonávajú výpočty a ukladajú dáta
- Následne na ne odošle úlohy na vykonanie a Executor spustí úlohu
- Ak niektorý z pracovných uzlov zlyhá, úlohy budú priradené na iný Executor
- Po ukončení úloh je SparkContext ukončený z Riadiaceho programu a zdroje sú Manžérom zdrojov uvoľnené

Spark master

- Master parameter špecifikuje spôsob ako sa SparkContext spustí

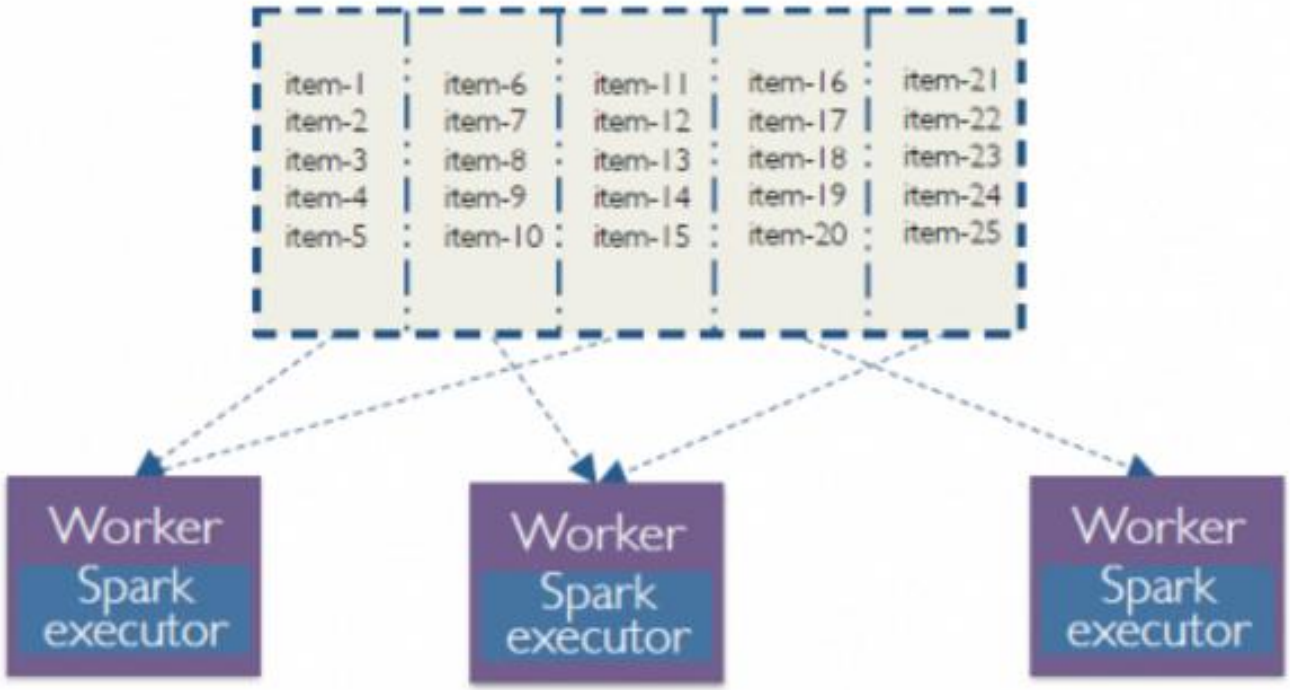
Master Parameter	Popis
`local`	Spustí Spark lokálne s jedným pracovným vláknom (žiaden paralelizmus)
`local[K]`	Spustí Spark lokálne s K pracovnými vláknami (ideálne ak K = počet jadier)
`spark://HOST:PORT`	Pripojenie na Spark klaster; PORT v závislosti od konfigurácie (7077 by default)
`mesos://HOST:PORT`	Pripojenie na Mesos klaster; v závislosti od konfigurácie (5055 by default)

Spark model

- Spark – dve hlavné abstrakcie pre paralelné programovanie – RDD, ktoré slúžia na ukladanie a prenos dát a paralelné operácie na týchto dátach

- Resilient distributed datasets (RDD) – primárna abstrakcia v Sparku
  - Kolekcie objektov určené na čítanie, distribuované na zariadeniach naprieč klastrom
  - RDD sú distribuované kolekcie, ale programátor k nim pristupuje ako k lokálnej premennej
  - Používateľ môže RDD cache-ovať v pamäti, vďaka čomu môže na ne opätovne aplikovať paralelné operácie bez potreby znova čítať dáta z úložiska
  - Tolerancia voči zlyhaniam – obsahujú informácie o spôsobe ako boli vytvorené, preto môžu byť znovuvytvorené, keď je niektorá z častí stratená (zlyhaním uzla)
  - Sú vytvárané a modifikované transformovaním dát z úložísk použitím operácií (map, filter, group-by, atď.)

Paralelizácia RDD



Tvorba RDD

- Spark umožňuje zostrojiť RDD niekoľkými spôsobmi:
  - Zo súboru v lokálnom alebo vzdialenom súborovom systéme, napr. HDFS
  - Paralelizovaním kolekcií v riadiacom programe, čo znamená ich rozdelenie na oddelené časti, ktoré sú odoslané na jednotlivé uzly klastra
  - Transformovaním existujúceho RDD
    - Datasety elementov typu A môžu byť zmenené na datasety s elementami typu B použitím Spark operácií. T
  - Zmenou životnosti existujúcej RDD
    - Implicitne RDD sú starnúce kolekcie. Partície datasetov sú materializované na žiadosť vo chvíli, keď sú použité v paralelných operáciách a sú odstránené z pamäte po použití.

RDD operácie

- Dva základné typy Spark operácií:
  - Transformácie – aplikujú sa na RDD vrátia transformované RDD
  - Akcie – aplikujú sa na RDD a vrátia výsledok

- Transformácie sú tzv. „lazy“ (nie sú vypočítané okamžite)
- Transformované RDD sa vykonajú až keď sa na nich vykoná akcia
- RDD zostávajú v pamäti, alebo na disku

RDD akcie

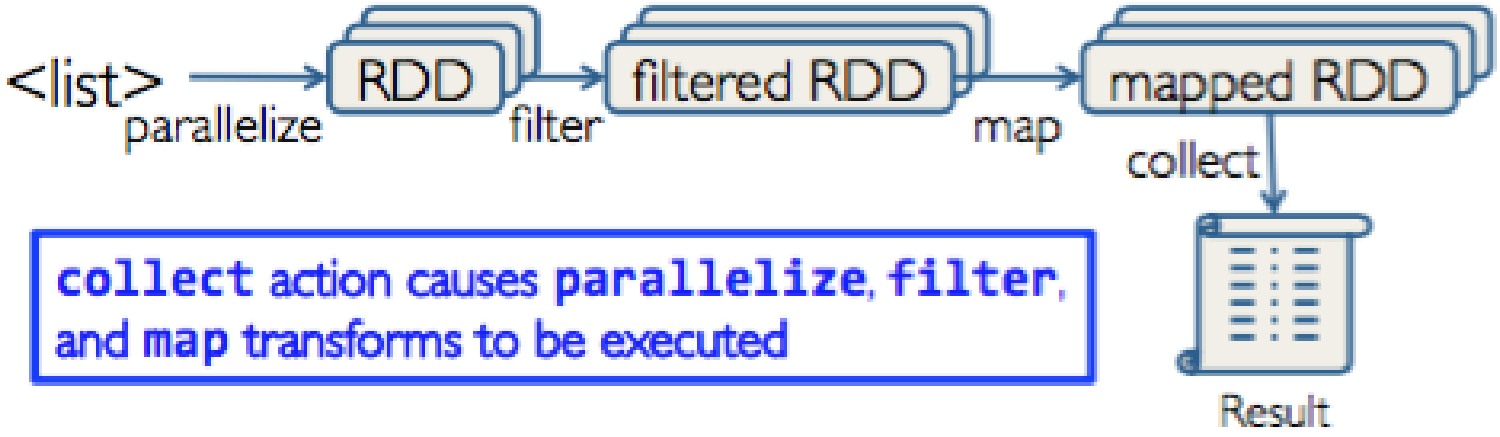
- reduce(*func*) – agreguje elementy RDD použitím funkcie *func*
- collect() – vráti elementy RDD ako pole
- count() – vráti počet elementov RDD
- first() – vráti prvý element RDD
- take(*n*) – vráti *n* prvých elementov RDD
- takeSample(*replacement*, *num*, *seed*) – vráti pole náhodného výberu *n* prvkov z RDD
- saveAsTextFile() – ukladá elementy do súboru (lokálny súborový systém, HDFS)
- countByKey() – pre RDD v tvare (key, value) vráti početnosti elementov podľa kľúča
- foreach(*func*) – aplikuje funkciu *func* na každý element RDD

RDD Transformácie

- map(*func*) – na každý element RDD aplikuje funkciu *func*
- filter(*func*) – vráti RDD, kde elementy spĺňajú podmienku funkcie *func* (vráti *true*)
- flatMap(*func*) – analogicky ako map, ale "rozbalí" vnorené objekty na jednu úroveň
- sample(*replacement*, *fraction*, *seed*) – vracia časť vstupného RDD
- union(*RDD2*) – vracia RDD ktoré vznikne zjednotením vstupného RDD s RDD2
- intersection(*RDD2*) –vráti prienik dvoch RDD
- distinct([*numTasks*]) – vráti RDD s ostatnými elementami, ako ktoré sú špecifikované v parametri
- groupByKey([*numTasks*]) – ak sa použije na RDD v tvare (key, value) vráti RDD zoskupené podľa kľúča (key, Iterable<value>)
- reduceByKey(*func*, [*numTasks*]) – vráti RDD v tvare (key, value), kde hodnoty sú agregované funkciou *func*
- sortByKey([*ascending*], [*numTasks*]) – vráti zoradené RDD podľa kľúča
- join(*RDD*, [*numTasks*]) – ak dva RDD sú (key, value1) a (key, value2), výsledné RDD potom bude (key, (value1, value2))

Práca s RDD

- Vytvorenie RDD z dátového zdroja
- Aplikácia transformácií na RDD: napr. “map”
- Použitie operácií na RDD: napr. “collect, count”



Rozdiely Spark a MapReduce

	MapReduce	Spark
Úložisko	Disk	In-memory
Operácie	Map a Reduce	Map, Reduce, Join, Sample…
Model vykonávania	Batch	Batch, micro-batch
Prostredie	Java	Scala, Java, R, Python

- Zovšeobecnené patterny
  - jednotný engine pre viacero prípadov použitia
- „Lazy“ vykonávanie
  - Redukuje prestoje, čakanie medzi akciami
- Nižšia cena komunikácie pri spúšťaní úloho

Spark knižnice

- Spark SQL
- Spark Streaming – spracovanie prúdov dát
- MLlib – knižnica strojového učenia
- GraphX – knižnica pre grafové výpočty

Spark SQL

- Spark interface pre prácu so štruktúrovanými dátami
- Umožňuje načítavať dáta z rôznych zdrojov (napr. Hive, Parquet, JSON, JDBC a RDD) a formulovať SQL dopyty
- Integruje Spark (Java, Scala a Python API) a SQL
- SparkSQL pridáva operácie pre SQL dopytovanie

Spark SQL

- Dataset/DataFrames API
- Dataset – distribuovaná kolekcia dát (interface od Spark 1.6 a vyššie, kombinuje RDD s Spark SQL enginom)
- DataFrame (dátový rámec) – Dataset organizovaný do stĺpcov (konceptuálne ekvivalentný k tabuľke relačných databáz)
- Dátový rámec je možné vytvoriť zo: štruktúrovaných súborov (csv, json, parquet atď), Hive tabuliek, databáz (JDBC), alebo z RDD

Spark SQL

- SparkSQL API potom umožňuje:
  - vykonávať operácie nad dátovými rámcami (prístup a operácie nad dátam podľa atribútov)
  - pristupovať ku množine funkcií nad dátovými rámcami
  - spúšťať SQL dopyty
- Výsledky jednotlivých operácií sú vrátené ako dátový rámec

Spark streaming

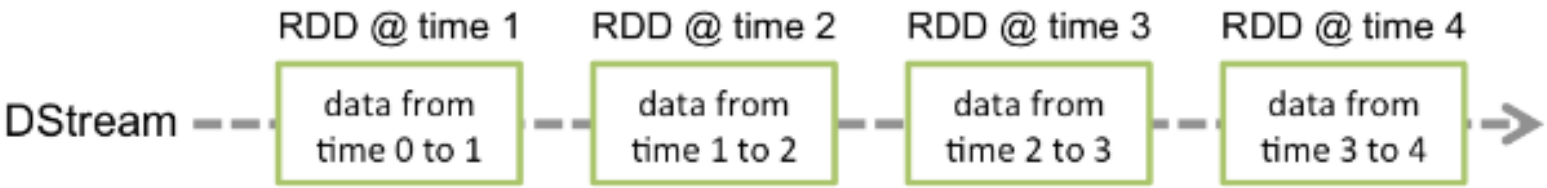
- Rozkúskovanie prúdu dát do malých dávok pozostávajúcich z okien časových okamihov (tzv. micro-batch processing)
- Spark potom považuje každú dávku dát ako RDD a spracuje ju použitím RDD operácií
- Nakoniec spracované výsledky RDD operácií budú vrátené v dávkach
- Možnosť kombinácie dávkového spracovania a spracovania prúdov dát jedným a tým istým systémom

Spark Streaming

- Diskretizované spracovanie dátových prúdov
- Hlavná abstrakcia - *DStream* (diskretizovaný stream)
- Zdroje
  - základné: priamo dostupné zo StreamingContext API (stream zo súborov, socket)
  - zdroje pomocou pripojených technológií ako napr. Kafka, Flume, Kinesis. Vyžadujú špecifické závislosti/nástroje.

DStream

- Hlavná abstrakcia Spark Streaming-u
- Reprezentuje kontinuálny prúd dát zo vstupného zdroja alebo prúd dát, ktorý vznikol jeho transformáciou
- Každý DStream je reprezentovaný ako sekvencia RDD

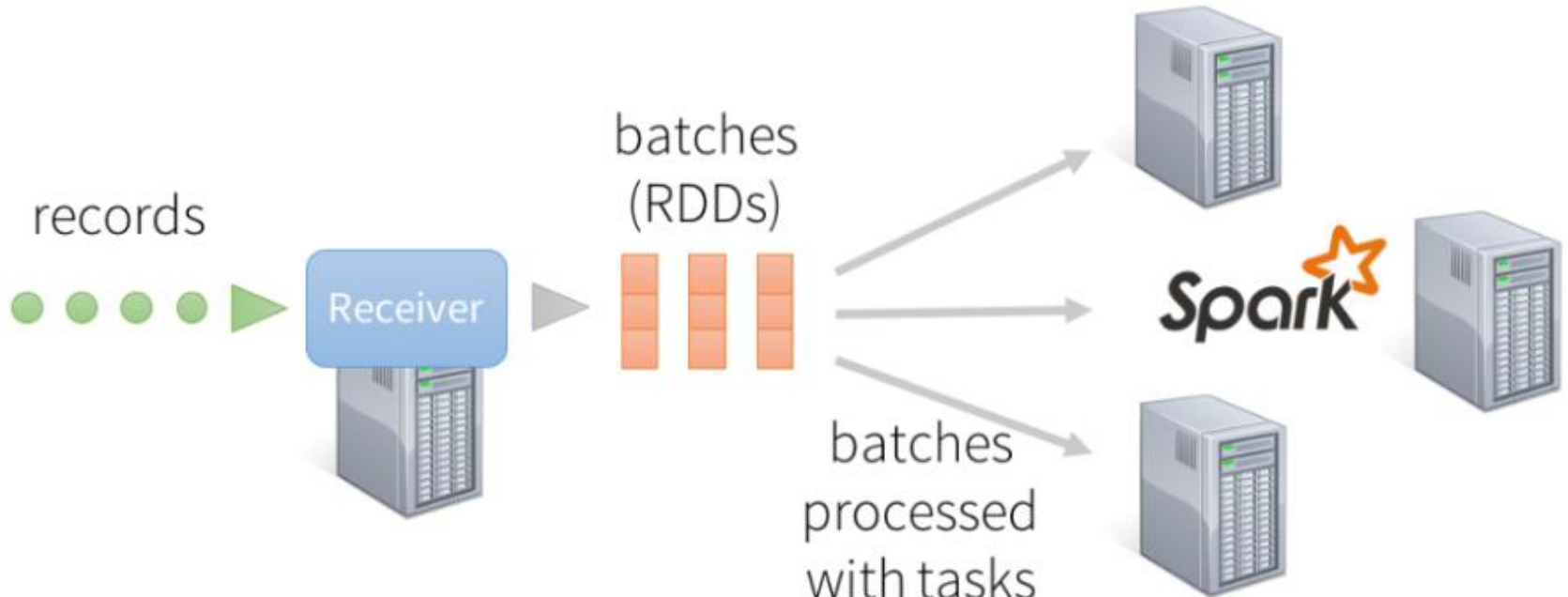


Spark Streaming

- Príjma dáta z rôznych zdrojov a preposiela ich do pamäte pracovných uzlov pre spracovanie

Spark Streaming

- Každý vstupný DStream má priradený Receiver
- Spark engine (optimalizovaný na nízku latenciu) spúšťa mnoho krátkych úloh na spracovanie mikro-dávok a produkuje výstupy
- Každá operácia aplikovaná na DStream sa transformuje na množinu operácií na jednotlivé RDD v DStreame



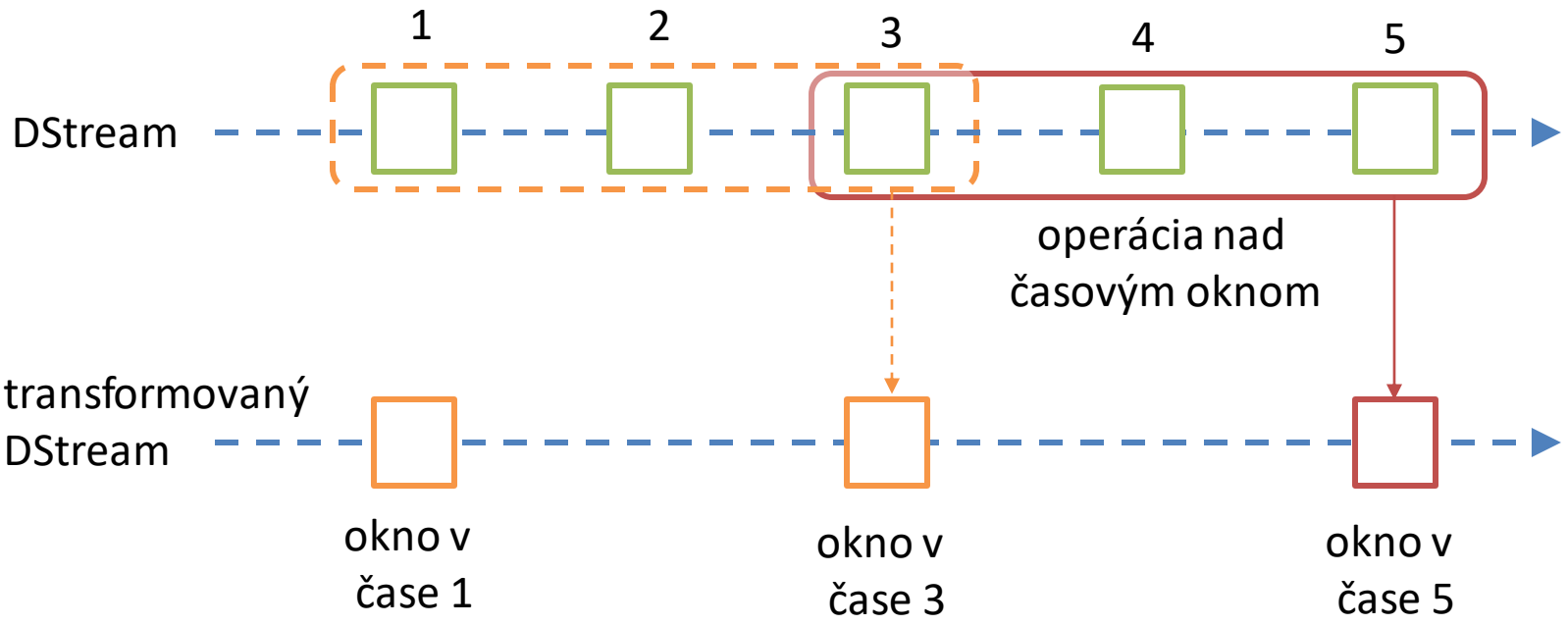
Spark Streaming

- Spark považuje každú dávku dát ako RDD
- To umožňuje spracovanie mikro-dávok Spark aplikáciami použitím RDD operácií vrátane rôznych knižníc (SparkSQL, GraphX, MLlib atď.)
- Nakoniec spracované výsledky RDD operácií budú vrátené v dávkach
- Veľkosť jednotlivých dávok je menej ako pol sekundy, latencia menej ako sekunda
- Možnosť kombinácie dávkového spracovania a spracovania prúdov dát jedným a tým istým systémom
- Úlohy sú prideľované dynamicky (podľa dostupnosti zdrojov a dát), čo zabezpečuje rovnomernú distribúciu úloh (a zaťaženia uzlov) v klastri

Práca s časovými oknami

- Spark Streaming poskytuje operácie pre tzv. *window-based computations*
- Umožňujú aplikovať operácie na na rôznych časových úsekoch dátového prúdu
- Špecifikujú sa 2 parametre:
  - *veľkosť posuvného okna* – dĺžka úseku
  - *interval posuvného okna* – interval, v ktorom sa majú jednotlivé operácie vykonávať

Operácie založené na časových oknách - príklad



- Vždy, keď sa posunie okno na zdrojovom DStreame, všetky RDD, ktoré spadajú do daného okna sú spracované použitím danej operácie

Operácie založené na časových oknách

- window(*dlzkaOkna*, *dlzkaIntervalu*)
- countByWindow(*dlzkaOkna*, *dlzkaIntervalu*)
- reduceByWindow(*func*, *dlzkaOkna*, *dlzkaIntervalu*)
- reduceByKeyAndWindow(*func*, *dlzkaOkna*, *dlzkaIntervalu*, [*numTasks*])
- countByValueAndWindow(*dlzkaOkna*, *dlzkaIntervalu*, [*numTasks*])

Príklad – Streaming wordcount

- Úloha – spočítať výskyty slov v streame textov

```
sc = SparkContext(“local[2]”, “StreamWordCount”)
ssc = StreamingContext(sc, 1)

lines = ssc.socketTextStream(“localhost”, 9999)
words = lines.flatMap(lambda line:line.split(“ “))

pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x,y: x+y)
```

Príklad – Streaming wordcount + posuvné okná

- Úloha – každých 10 sekúnd spočítať výskyty slov v prúdoch textov za posledných 30 sekúnd

```
sc = SparkContext(“local[2]”, “StreamWordCount”)
ssc = StreamingContext(sc, 1)

lines = ssc.socketTextStream(“localhost”, 9999)
words = lines.flatMap(lambda line:line.split(“ “))

pairs = words.map(lambda word: (word, 1))
windowedWordCounts = pairs.reduceByKeyAndWindow(lambda x,y:
x+y, 30, 10)
```